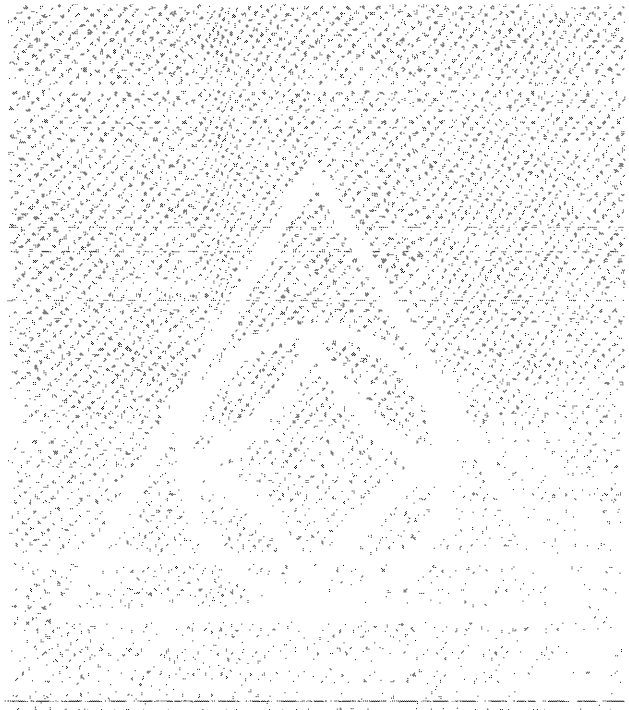


AFIPS

CONFERENCE PROCEEDINGS

VOLUME 42

1973

NATIONAL COMPUTER CONFERENCE AND EXPOSITION

June 4-8, 1973
New York, New York

AFIPS PRESS
210 SUMMIT AVENUE
MONTVALE, NEW JERSEY 07645

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1973 National Computer Conference or the American Federation of Information Processing Societies. Inc.

Library of Congress Catalog Card Number 55-44701
AFIPS PRESS
210 Summit Avenue
Montvale, New Jersey 07645

Printed in the United States of America

PART I

SCIENCE AND TECHNOLOGY

CONTENTS

Linguistics and the future of computation

by DAVID G. HAYS

State University of New York
Buffalo, New York

My subject is the art of computation: computer architecture, computer programming, and computer application. Linguistics provides the ideas, but the use I make of them is not the linguist's use, which would be an attempt at understanding the nature of man and of human communication, but the computer scientist's use. In ancient India, the study of language held the place in science that mathematics has always held in the West. Knowledge was organized according to the best known linguistic principles. If we had taken that path, we would have arrived today at a different science. Our scholarship draws its principles from sources close to linguistics, to be sure, but our science has rather limited itself to a basis in Newtonian calculus. And so a chasm separates two cultures.

The scientific reliance on calculus has been productive. Often understood as a demand for precision and rigor, it has simultaneously made theoreticians answerable to experimental observation and facilitated the internal organization of knowledge on a scale not imagined elsewhere in human history. Very likely, a reliance on linguistic laws for control of science during the same long period would have been less successful, because the principles of linguistic structure are more difficult to discover and manipulate than the principles of mathematical structure; or so it seems after two thousand years of attention to one and neglect of the other. How it will seem to our descendants a thousand years hence is uncertain; they may deem the long era of Western study of mathematical science somewhat pathological, and wonder why the easy, natural organization of knowledge on linguistic lines was rejected for so many centuries.

However that may be, the prospect for the near term is that important opportunities will be missed if linguistic principles continue to be neglected. Linguistics is enjoying a period of rapid growth, so a plethora of ideas await new uses; the computer makes it possible to manipulate even difficult principles. Traditional mathematics seems not to say how computers much beyond the actual state of the art can be organized, nor how programs can be made much more suitable to their applications and human users, nor how many desirable fields of application can be conquered. I think that linguistics has something to say.

THREE LINGUISTIC PRINCIPLES

Since I cannot treat the entire field of linguistics. I have chosen to sketch three principles that seem most basic and far-reaching. Two of them are known to every linguist and applied automatically to every problem that arises. The third is slightly less familiar; I have begun a campaign to give it due recognition.

As everyone knows, the capacity for language is innate in every human specimen, but the details of a language are acquired by traditional transmission, from senior to younger. As everyone knows, language is a symbolic system, using arbitrary signs to refer to external things, properties, and events. And, as everyone certainly knows by now, language is productive or creative, capable of describing new events by composition of sentences never before uttered. Hockett[9] and Chomsky[3] explain these things. But of course these are not principles; they are problems for which explanatory principles are needed.

My first principle is stratification.[12] This principle is often called duality of patterning, although in recent years the number of levels of patterning has grown. The original observation is that language can be regarded as a system of sounds or a system of meaningful units; both points of view are essential. One complements the other without supplanting it. Phonology studies language as sound. It discovers that each of the world's languages uses a small alphabet of sounds, from a dozen to four times that many, to construct all its utterances. The definition of these unit sounds is not physical but functional. In one language, two sounds with physically distinct manifestations are counted as functionally the same; speakers of this language do not acquire the ability to distinguish between the sounds, and can live out their lives without knowing that the sounds are unlike. English has no use for the difference between [p] and [p'], the latter having a little puff of air at the end, yet both occur: [p] in *spin*, [p'] in *pin*. Since other languages, notably Thai, use this difference to distinguish utterances, it is humanly possible not only to make the two forms of /p/ but also to hear it. Thus languages arbitrarily map out their alphabets of sounds.[10]

Languages also differ in the sequences of sounds that they permit. In Russian, the word *vzbalmoshnyj* 'extrava-

gant' is reasonable, but the English speaker feels that the initial sequence /vzb/ is extravagant, because initial /v/ in English is not followed by another consonant, and furthermore initial /z/ is not. The Russian word violates English rules, which is perfectly satisfactory to Russian speakers, because they are unacquainted with English restrictions. As Robert Southey put it, speaking of a Russian,

> And last of all an Admiral came,
> A terrible man with a terrible name,
> A name which you all know by sight very well
> But which no one can speak, and no one can spell.

(Robert Southey, 'The March to Moscow.') Phonology, with its units and rules of combinations, is one level of patterning in language.

That languages are patterned on a second level is so well known as to require little discussion. English puts its subject first, verb second, object last—in simple sentences. Malagasy, a language of Madagascar, puts the verb first, then the object, last the subject.[11] Other orders are found elsewhere. English has no agreement in gender between nouns and adjectives, but other languages such as French and Russian, Navaho and Swahili, do; nor is gender controlled by semantics, since many gender classes are without known semantic correlation. Gender is as arbitrary as the English rejection of initial /vzb/.

The units that enter into grammatical patterns are morphemes; each language has its own stock, a vocabulary that can be listed and found once more to be arbitrary. It seems true that some color names are universal —needed in all languages to symbolize genetic capacities —but other color names are also coined, such as the English *scarlet* and *crimson*, on arbitrary lines.[1,8]

The existence of a third level of symbolic patterning is best shown by psychological experiments. Memory for a story is good, but not verbatim. Only the shortest stretches of speech can be remembered word for word; but the ideas in quite a long stretch can be recited after only one hearing if the hearer is allowed to use his own words and grammatical structures.[5] The comparison of pictures with sentences has been investigated by several investigators; they use models in which *below* is coded as *not above, forget* is coded as *not remember*, and so on, because they need such models to account for their subjects' latencies (times to respond measured in milliseconds). Using such models, they can account for the differences between times of response to a single picture, described with different sentences, to an impressive degree of precision.[12]

Each level of symbolic patterning should have both units and rules of construction. On the phonological level the units are functional sounds; the rules are rules of sequence, for the most part. On the grammatical level the units are morphemes and the rules are the familiar rules of sequence, agreement, and so on. On the third level, which can be called semological or cognitive, the units are often called sememes; the morpheme 'forget' corresponds to the sememes 'not' and 'remember'. The rules of organization of this level have not been investigated adequately. Many studies of paradigmatic organization have been reported, sometimes presenting hierarchical classifications of items (a canary is a bird, a dog is a quadruped, etc.), but this is only one of several kinds of organization that must exist. Classification patterns are not sentences, and there must be sentences of some kind on the semological level. Chomsky's deep structures might be suitable, but Fillmore[6] and McCawley[13] have proposed different views. What is needed is rapidly becoming clearer, through both linguistic and psychological investigations. The relations that help explain grammar, such as subject and object, which control sequence and inflection, are not the relations that would help most in explaining the interpretation of pictures or memory for stories; for such purposes, notions of agent, instrument, and inert material are more suitable. But the organization of these and other relations into a workable grammar of cognition is unfinished.

Up to this point I have been arguing only that language is stratified, requiring not one but several correlated descriptions. Now I turn to my second principle, that language is internalized. Internalization is a mode of storage in the brain, intermediate between innateness and learning. Concerning the neurology of these distinctions I have nothing to say. Their functional significance is easy enough to identify, however.

What is innate is universal in mankind, little subject to cultural variation. Everyone sees colors in about the same way, unless pathologically color blind. Everyone on earth has three or more levels of linguistic patterning. The distinctions among things (nouns), properties (adjectives), and events (verbs) are so nearly universal as to suggest that this threefold organization of experience is innate. To have grammar is universal, however much the grammars of particular languages vary. The innate aspects of thought are swift, sure, and strong.

What is internalized is not the same in every culture or every person. But whatever a person internalizes is reasonably swift, sure, and strong; less than what is innate, more than what is learned. Besides the mechanisms of linguistic processing, various persons internalize the skills of their arts and crafts; some internalize the strategies of games; and all internalize the content of at least some social roles.

The contrast between learning and internalization is apparent in knowledge of a foreign language. A person who has learned something of a foreign language without internalization can formulate sentences and manage to express himself, and can understand what is said to him, although slowly and with difficulty. A person who has internalized a second language is able to speak and understand with ease and fluency.

Similarly in games, the difference between a master of chess, bridge, or go is apparent. But something more of the difference between learning and internalization is also to be seen here. The novice has a more detailed awareness of how he is playing; he examines the board or the cards step by step, applying the methods he has learned, and can report how he arrives at his decision. Awareness goes with learned skills, not with internalized abilities.

Internalized abilities are the basis of more highly organized behavior. The high-school learner of French is not able to think in French, nor is the novice in chess able to construct a workable strategy for a long sequence of moves. When a language has been internalized, it becomes a tool of thought; when a chess player has internalized enough configurations of pieces and small sequences of play, he can put them together into strategies. The musician internalizes chords, melodies, and ultimately passages and whole scores; he can then give his attention to overall strategies, making his performance lyrical, romantic, martial, or whatever.

What makes learning possible is the internalization of a system for the storage and manipulation of symbolic matter. If a person learns a story well enough to tell it, he uses the facilities of symbolic organization—his linguistic skills—to hold the substance of the story. Much of the content of social roles is first learned in this way; the conditions of behavior and the forms of behavior are learned symbolically, then come to be, as social psychologists put it, part of the self—that is, internalized. In fact, the conversion of symbolic learned material into internalized capacities is a widespread and unique fact of human life. It must be unique, since the symbol processing capacity required is limited to man. This ability gives man a great capacity for change, for adaptation to different cultures, for science: by internalizing the methods of science, he becomes a scientist.

The amount that a person internalizes in a lifetime is easy to underestimate. A language has thousands of morphemes and its users know them. Certainly their semantic and grammatical organization requires tens—more plausibly hundreds—of thousands of linkages. A high skill such as chess takes internalize knowledge of the same order of magnitude, according to Simon and Barenfeld.[15]

I think that internalized units are more accurately conceived as activities than as inert objects. All tissue is metabolically active, including the tissue that supports memory. Memory search implies an activity, searching, in an inactive medium, perhaps a network of nodes and arcs. More fruitfully we can imagine memory as a network of active nodes with arcs that convey their activity from one to another. A morpheme, then, is an activity seeking at all times the conditions of its application.

My third principle in linguistics is the principle of metalinguistic organization. Language is generally recognized as able to refer to itself; one can mention a word in order to define it, or quote a sentence in order to refute it. A very common occurrence in grammar is the embedding of one sentence within another. A sentence can modify a word in another sentence, as a relative clause:

The boy who stole the pig ran away.

A sentence can serve as the object of a verb of perception, thought, or communication:

I saw him leave. I know that he left.
You told me that she had left.

And two sentences can be embedded in temporal, spatial, or causal relation:

He ran away because he stole the pig.
He stole the pig and then ran away.
He is hiding far from the spot where he stole the pig.

An embedded sentence is sometimes taken in the same form as if it were independent, perhaps introduced by a word like the English *that*, and sometimes greatly altered in form as in *his running away*.

The definition of abstract terms can be understood by metalingual linkages in cognitive networks. The definition is a structure, similar to the representation of any sentence or story in a cognitive network. The structure is linked to the term it defines, and the use of the term governed by the content of the structure. Science and technology are replete with terms that cannot be defined with any ease in observation sentences; they are defined, I think, through metalingual linkages. What kind of program is a compiler? What kind of device can correctly be called heuristic? These questions can be answered, but useful answers are complicated stories about the art of programming, not simple statements of perceptual conditions, and certainly not classificatory statements using elementary features such as *human, male*, or *concrete*.

I can indicate how vast a difference there is between metalingual operations and others by proposing that all other operations in cognitive networks are performed by path tracing using finite-state automata, whereas metalingual operations are performed by pattern matching using pushdown automata. These two systems differ in power; a finite-state machine defines a regular language and a pushdown automaton defines a context-free language.

A path in a cognitive network is defined as a sequence of nodes and arcs; to specify a path requires only a list of node and arc types, perhaps with mention that some are optional, some can be repeated. A more complex form of path definition could be described, but I doubt that it would enhance the effectiveness of path tracing procedures. In Quillian's work, for example, one needs only simple path specifications to find the relation between *lawyer* and *client* (a client employs a lawyer). To know that a canary has wings requires a simple form of path involving paradigmatic (a canary is a kind of bird) and syntagmatic relations (a bird has wings). The limiting factor is not the complexity of the path that can be

defined from one node to another, but the very notion that node-to-node paths are required.[4,14]

Pattern matching means fitting a template. The patterns I have in mind are abstract, as three examples will show. The first, familiar to linguists, is the determination of the applicability of a grammatical transformation. The method, due to Chomsky, is to write a template called a structure description. The grammatical structure of any sentence is described by some tree; if the template fits the tree of a certain sentence, then the transformation applies to it, yielding a different tree. These templates contain symbols that can apply to nodes in grammatical trees, and relations that can connect the nodes. Chomsky was, I think, the first to recognize that some rules of grammar can be applied only where structure is known; many phenomena in language are now seen to be of this kind. Thus linguists today ask for tree-processing languages and cannot do with string processors.

My second example is the testing of a proof for the applicability of a rule of inference. A proof has a tree structure like the structure of a sentence; whether a rule of inference can be applied has to be tested by reference to the structure. 'If p and q then p' is a valid inference, provided that in its application p is one of the arguments of the conjunction; one cannot assert that p is true just because p, q, and a conjunction symbol all occur in the same string.

Finally, I come to metalingual definition. The definition is itself a template. The term it defines is correctly used in contexts where the template fits. As in the first two examples, the template is abstract. A structure description defines a class of trees; the transformation it goes with applies to any tree in the class. A rule of inference defines a class of proofs; it applies to each of them. And a metalingual definition defines a class of contexts, in each of which the corresponding term is usable. Charity has many guises; the story-template that defines *charity* must specify all of the relevant features of charitable activity, leaving the rest to vary freely.

When the difference in power between finite-state and context-free systems was discovered, it seemed that this difference was a fundamental reason for preferring context-free grammars in the study of natural language. Later it became evident that the need to associate a structural description with each string was more important, since context-free grammars could do so in a natural way and finite-state automata could not. Today linguists and programmers generally prefer the form of context-free rules even for languages known to be finite state, just because their need for structure is so urgent. It may prove the same with pattern matching. In proofs, in transformations, and in definitions it is necessary to mark certain elements: the conclusions of inferences, the elements moved or deleted by transformation, and the key participating elements in definition. (The benefactor is charitable, not the recipient.) Until I see evidence to the contrary, however, I will hold the view that pattern matching is more powerful than path tracing.

Pattern matching is, surely, a reflective activity in comparison with path tracing. To trace a path through a maze, one can move between the hedges, possibly marking the paths already tried with Ariadne's thread. To see the pattern requires rising above the hedges, looking down on the whole from a point of view not customarily adopted by the designers of cognitive networks. That is, they often take such a point of view themselves, but they do not include in their systems a component capable of taking such a view.

COMPUTER ARCHITECTURE

I turn now to the art of computation, and ask what kind of computer might be constructed which followed the principles of stratification, internalization, and metalingual operation.

Such a computer will, I freely admit, appear to be a special-purpose device in comparison with the general-purpose machines we know today. The human brain, on close inspection, also begins to look like a special-purpose machine. Its creativity is of a limited kind, yet interesting nevertheless. The prejudice in favor of mathematics and against linguistics prefers the present structure of the computer; but a special-purpose machine built to linguistic principles might prove useful for many problems that have heretofore been recalcitrant.

Stratification is not unknown in computation, but it deserves further attention. The difficulties of code translation and data-structure conversion that apparently still exist in networks of different kinds of computers and in large software systems that should be written in a combination of programming languages are hard to take. The level of morphemics in language is relatively independent of both cognition and phonology. In computer architecture, it should be possible to work with notational schemes independent of both the problem and the input-output system. Whether this level of encoding both data and their organization should be the medium of transmission, or specific to the processor, I do not know. But it is clear that translators should be standard hardware items, their existence unknown in high-level languages. Sophistication in design may be needed, but the problems seem not insurmountable, at least for numerical, alphabetic, and pictorial data. The design of translators for data structures is trickier, and may even prove not to be possible on the highest level.

The lesson to be learned from the separation of grammar and cognition is more profound. Language provides a medium of exchange among persons with different interests and different backgrounds; how they will understand the same sentence depends on their purposes as well as their knowledge. Much difficulty in computer programming apparently can be traced to the impossibility of separating these two levels in programming languages. Programs do not mean different things in different contexts; they mean the same thing always. They are there-

fore called unambiguous, but a jaundiced eye might see a loss of flexibility along with the elimination of doubt. Many simple problems of this class have been solved; in high-level languages, addition is generally not conditioned by data types, even if the compiler has to bring the data into a common type before adding. More difficult problems remain. At Buffalo, Teiji Furugori is working on a system to expand driving instructions in the context of the road and traffic. He uses principles of safe driving to find tests and precautions that may be needed, arriving at a program for carrying out the instruction safely. Current computer architecture is resistant to this kind of work; it is not easy to think of a program on two levels, one of them providing a facility for expanding the other during execution. An interpreter can do something of the sort; but interpretive execution is a high price to pay. If computer hardware provided for two simultaneous monitors of the data stream, one executing a compiled program while the other watched for situations in which the compiled version would be inadequate, the separation of morphemics and cognition might better be realized.

In teaching internalization to students who are mainly interested in linguistics, I use microprogramming as an analogy. What can be seen in the opposite direction is the fantastic extent to which microprogramming might be carried with corresponding improvement in performance. If the meaning of every word in a language (or a large fraction of the words) is internalized by its users, then one may hope that microprogramming of a similar repertory of commands would carry possibilities for the computer somewhat resembling what the speaker gains, to wit, speed.

A computer could easily be built with a repertory of 10,000 commands. Its manual would be the size of a desk dictionary; the programmer would often find that his program consisted of one word, naming an operation, followed by the necessary description of a data structure. Execution would be faster because of the intrinsically higher speed of the circuitry used in microprogramming. Even if some microprograms were mainly executive, making numerous calls to other microprograms, overall speed should be increased. At one time it would have been argued that the art could not supply 10,000 widely used commands, but I think that time is past. If someone were inclined, I think he could study the literature in the field and arrive at a list of thousands of frequently used operations.

Parallel processing adds further hope. If a computer contains thousands of subcomputers, many of them should be operating at each moment. Even the little we know about the organization of linguistic and cognitive processing in the brain suggests how parallel processing might be used with profit in systems for new applications.

A morphemic unit is the brain seems to be an activity, which when successful links a phonological string with one or more points in a cognitive network. If these units had to be tested sequentially, or even by binary search, the time to process a sentence would be great. Instead all of them seem to be available at all times, watching the input and switching from latency to arousal when the appropriate phonological string appears. If each were a microprogram, each could have access to all input. Conflicts inevitably arise, with several units aroused at the same time. Grammar serves to limit these conflicts; a grammatical unit is one with a combination of inputs from morphemic units. When a morphemic unit is aroused, it signals its activation to one or several grammatical units. When a proper combination of morphemic units is aroused, the grammatical unit is in turn aroused and returns feedback to maintain the arousal of the morphemic unit which is thereupon enabled to transmit also to the cognitive level. Thus the condition for linkage between phonology and cognition is a combination of grammatical elements that amounts to the representation of a sentence structure. This is Lamb's model of stratal organization, and shows how grammar reduces lexical ambiguity. The problem it poses for computer architecture is that of interconnection; unless the morphemic units (like words) and the grammatical units (like phrase-structure rules) are interconnected according to the grammar of a language, nothing works. The computer designer would prefer to make his interconnections on the basis of more general principles; but English is used so widely that a special-purpose computer built on the lines of its grammar would be acceptable to a majority of the educated persons in the world—at least, if no other were on the market.

A similar architecture could be used for other purposes, following the linguistic principle but not the grammar of a natural language. Ware[16] mentions picture processing and other multidimensional systems as most urgently needing increased computing speed. Models of physiology, of social and political systems, and of the atmosphere and hydrosphere are among these. Now, it is in the nature of the world as science knows it that local and remote interactions in these systems are on different time scales. A quantum of water near the surface of a sea is influenced by the temperature and motion of other quanta of water and air in its vicinity; ultimately, but in a series of steps, it can be influenced by changes at remote places. Each individual in a society is influenced by the persons and institutions close to him in the social structure. Each element of a picture represents a portion of a physical object, and must be of a kind to suit its neighbors.

To be sure, certain factors change simultaneously on a wide scale. If a person in a picture is wearing a striped or polka-dotted garment, the recognition of the pattern can be applied to the improvement of the elements throughout the area of the garment in the picture. A new law or change in the economy can influence every person simultaneously. Endocrine hormones sweep through tissue rapidly, influencing every point almost simultaneously. When a cloud evaporates, a vast area is suddenly exposed to a higher level of radiation from the sun.

These situations are of the kind to make stratification a helpful mode of architecture. Each point in the grid of

picture, physiological organism, society, or planet is connected with its neighbors on its own stratum and with units of wide influence on other strata; it need not be connected with remote points in the same stratum.

Depending on the system, different patterns of interaction have to be admitted. Clouds are formed, transported, and evaporated. Endocrine glands, although they vary in their activity, are permanent, as are governments. Both glands and governments do suffer revolutionary changes within the time spans of useful simulations. In a motion picture, objects enter and depart.

How the elements of the first stratum are to be connected with those of the next is a difficult problem. It is known that the cat's brain recognizes lines by parallel processing; each possible line is represented by a cell or cells with fixed connections to certain retinal cells. But this does not say how the cat recognizes an object composed of several lines that can be seen from varying orientations. Switching seems unavoidable in any presently conceivable system to connect the level of picture elements with the level of objects, to connect the level of persons with the level of institutions, to connect the elements of oceans with the level of clouds, or to connect the elements of the morphemic stratum with the level of cognition in linguistic processing.

In computation, it seems that path tracing should be implicit, pattern matching explicit. The transmission of activity from a unit to its neighbors, leading to feedback that maintains or terminates the activity of the original unit, can be understood as the formation of paths. Something else, I think, happens when patterns are matched.

A typical application of a linguistically powerful computer would be the discovery of patterns in the user's situation. The user might be a person in need of psychiatric or medical help; an experimenter needing theoretical help to analyze his results and formulate further experiments; a lawyer seeking precedents to aid his clients; or a policy officer trying to understand the activities of an adversary. In such cases the user submits a description of his situation and the computer applies a battery of patterns to it. The battery would surely have to be composed of thousands of possibilities to be of use; with a smaller battery, the user or a professional would be more helpful than the computer.

If the input is in natural language, I assume that it is converted into a morphemic notation, in which grammatical relations are made explicit, as a first step.

On the next level are thousands of patterns, each linked metalingually to a term; the computer has symbolic patterns definitive of *charity, ego strength, heuristics, hostility,* and so on. Each such pattern has manifold representations on the morphemic stratum; these representations may differ in their morphemes and in the grammadical linkages among them. Some of these patterns, in fact, cannot be connected to the morphemic stratum directly with any profit whatsoever, but must instead be linked to other metalingual patterns and thence ultimately to morphemic representations. In this way the cognitive patterns resemble objects in perception that must be recognized in different perspectives.

Grammatical theory suggests an architecture for the connection of the strata that may be applicable to other multistratal systems. The two strata are related through a bus; the object on the lower stratum is a tree which reads onto the bus in one of the natural linearizations. All of the elements of all of the patterns on the upper stratum are connected simultaneously to the bus and go from latent to aroused when an element of their class appears; these elements include both node and arc labels. When the last item has passed, each pattern checks itself for completeness; all patterns above a threshold transmit their arousal over their metalingual links to the terms they define. Second-order patterns may come to arousal in this way, and so on.

If this model has any validity for human processing, it brings us close to the stage at which awareness takes over. In awareness, conflicts are dealt with that cannot be reduced by internalized mechanisms. The chess player goes through a few sequences of moves to see what he can accomplish on each of them; the listener checks out those occasional ambiguities that he notices, and considers the speaker's purposes, the relevance of what he has heard to himself, and so on. The scientist compares his overall theoretical views with the interpretations of his data as they come to mind and tries a few analytic tricks. In short, this is the level at which even a powerful computer might open a dialogue with the user.

Would a sensible person build a computer with architecture oriented to a class of problems? I think so, in a few situations. Ware listed some problems for which the payoff function varies over a multibillion-dollar range: foreign policy and arms control, weather and the environment, social policy, and medicine. With such payoffs, an investment of even a large amount in a more powerful computer might be shown to carry a sufficient likelihood of profit to warrant a gamble. In the case of language-oriented architecture, it is not hard to develop a composite market in which the users control billions of dollars and millions of lives with only their own brains as tools to link conceptualization with data. A president arrives at the moment of decision, after all the computer simulations and briefings, with a yellow pad and a pencil; to give him a computer which could help his brain through multistratal and metalingual linkages of data and theories would be worth a substantial investment.

Can these applications be achieved at optimal levels without specialized architecture? I doubt it. Parallel processing with general-purpose computers linked through generalized busses will surely bring an improvement over serial processing, but raises problems of delay while results are switched from one computer to another and does nothing to solve software problems. Specialized architecture is a lesson to be learned from linguistics with consequences for ease of programming, time spent in compilation or interpretation, and efficiency of parallel processing.

COMPUTATIONAL LINGUISTICS

I have delayed until the end a definition of my own field, which I have presented before.[7] It should be more significant against the background of the foregoing discussion.

The definition is built upon a twofold distinction. One is the distinction, familiar enough, between the infinitesimal calculus and linguistics. The calculus occupies a major place in science, giving a means of deduction in systems of continuous change. It has developed in two ways: Mathematical analysis, which gives a time-independent characterization of systems including those in which time itself is a variable—time does not appear in the metasystem of description. And numerical analysis, in which time is a variable of the metasystem; numerical analysis deals in algorithms.

Linguistics, also, has developed in two ways. The time-independent characterizations that Chomsky speaks of as statements of competence are the subject of what is called linguistics, with no modifier. This field corresponds to the calculus, or to its applications to physical systems. Time-dependent characterizations of linguistic processes are the subject matter of computational linguistics, which also has two parts. Its abstract branch is purely formal, dealing with linguistic systems whether realized, or realizable, in nature; its applied branch deals with algorithms for the processing of naturally occurring languages.

I have undertaken to show that the concepts of abstract computational linguistics provide a foundation for nonnumerical computation comparable to that provided by the calculus for numerical computation. The work is still in progress, and many who are doing it would not be comfortable to think of themselves as computational linguists. I hope that the stature of the field is growing so that more pride can attach to the label now and hereafter than in earlier days.

REFERENCES

1. Berlin, Brent, Kay, Paul, *Basic Color Terms: Their Universality and Evolution*, Berkeley, University of California Press, 1969.
2. Chase, William G., Clark, Herbert H., "Mental Operations in the Comparison of Sentences and Pictures," in *Cognition in Learning and Memory*, edited by Lee W. Gregg, New York, Wiley, 1972, pp. 205-232.
3. Chomsky, Noam, *Language and Mind*, New York, Harcourt Brace Jovanovich, enlarged edition, 1972.
4. Collins, Allan M., Quillian, M. Ross, "Experiments on Semantic Memory and Language Comprehension," in Gregg, op. cit., pps. 117-137.
5. Fillenbaum, S., "Memory for Gist: Some Relevant Variables," *Language and Speech*, 1966, Vol. 9, pp. 217-227.
6. Fillmore, Charles J., "The Case for Case," in *Universals in Linguistic Theory*, edited by Emmon Bach and Robert T. Harms, New York, Holt, Rinehart and Winston, 1968, pp. 1-88.
7. Hays, David G., *The Field and Scope of Computational Linguistics*, 1971 International Meeting on Computational Linguistics, Debrecen.
8. Hays, David G., Margolis, Enid, Naroll, Raoul, Perkins, Revere Dale. "Color Term Salience," *American Anthropologist*, 1972, Vol. 74, pp. 1107-1121.
9. Hockett, Charles F., "The Origin of Speech," *Scientific American*, 1960, Vol. 203, pp. 88-96.
10. Ladefoged, Peter, *Preliminaries to Linguistic Phonetics*, Chicago, University of Chicago Press, 1971.
11. Keenan, Edward L., "Relative Clause Formation in Malagasy," in *The Chicago Which Hunt*, edited by Paul M. Peranteau, Judith N. Levi, and Gloria C. Phares, Chicago Linguistic Society, 1972.
12. Lamb, Sydney M., *Outline of Stratificational Grammar*, Washington, Georgetown University Press, revised edition, 1966.
13. McCawley, James D., "The Role of Semantics in a Grammar," in *Bach and Harms*, op. cit., pp. 125-169.
14. Quillian, M. Ross, "The Teachable Language Comprehender," *Communications of the ACM*, 1969, Vol. 12, pp. 459-476.
15. Simon, Herbert A., Barenfeld, M., "Information-processing Analysis of Perceptual Processes in Problem Solving," *Psychological Review*, 1969, Vol. 76, pp. 473-483.
16. Ware, Willis H., "The Ultimate Computer," *IEEE Spectrum*, March 1972.

Speech understanding

by DONALD E. WALKER

Stanford Research Institute
Menlo Park, California

ABSTRACT

Research on speech understanding is adding new dimensions to the analysis of speech and to the understanding of language. The accoustic, phonetic, and phonological processing of speech recognition efforts are being blended with the syntax, semantics, and pragmatics of question-answering systems. The goal is the development of capabilities that will allow a person to have a conversation with a computer in the performance of a shared task. Achievement of this goal will both require and contribute to a more comprehensive and powerful model of language —with significant consequences for linguistics, for computer science, and especially for computational linguistics.

Syntax and computation

by JANE J. ROBINSON

The University of Michigan
Ann Arbor, Michigan

ABSTRACT

Algorithms have been developed for generating and parsing with context-sensitive grammars. In principle, the contexts to which a grammar is sensitive can be syntactic, semantic, pragmatic, or phonetic. This development points up the need to develop a new kind of lexicon, whose entries contain large amounts of several kinds of contextual information about each word or morpheme, provided in computable form. Ways in which both the form and content of the entries differ from those of traditional dictionaries are indicated.

Literary text processing

by SALLY YEATES SEDELOW

University of Kansas
Lawrence, Kansas

ABSTRACT

To date, computer-based literary text processing bears much greater similarity to techniques used for information retrieval and, to some degree, for question-answering, than it does to techniques used in, for example, machine translation of 'classical' artificial intelligence. A literary text is treated not as 'output' in a process to be emulated nor as a string to be transformed into an equivalent verbal representation, but, rather, as an artifact to be analyzed and described.

The absence of process as an integrating concept in computer-based literary text processing leads to very different definitions of linguistic domains (such as semantics and syntactics) than is the case with, for example, artificial intelligence. This presentation explores some of these distinctions, as well as some of the implications of more process-oriented techniques for literary text processing.

The augmented knowledge workshop

by DOUGLAS C. ENGELBART, RICHARD W. WATSON, and JAMES C. NORTON

Stanford Research Institute
Menlo Park, California

CONCEPT OF THE KNOWLEDGE WORKSHOP

This paper discusses the theme of augmenting a knowledge workshop. The first part of the paper describes the concept and framework of the knowledge workshop. The second part describes aspects of a prototype knowledge workshop being developed within this framework.

The importance and implications of the idea of knowledge work have been described by Drucker.[3,4] Considering knowledge to be the systematic organization of information and concepts, he defines the knowledge worker as the person who creates and applies knowledge to productive ends, in contrast to an "intellectual" for whom information and concepts may only have importance because they interest him, or to the manual worker who applies manual skills or brawn. In those two books Drucker brings out many significant facts and considerations highly relevant to the theme here, one among them (paraphrased below) being the accelerating rate at which knowledge and knowledge work are coming to dominate the working activity of our society:

> In 1900 the majority and largest single group of Americans obtained their livelihood from the farm. By 1940 the largest single group was industrial workers, especially semiskilled machine operators. By 1960, the largest single group was professional, managerial, and technical—that is, knowledge workers. By 1975-80 this group will embrace the majority of Americans. The productivity of knowledge has already become the key to national productivity, competitive strength, and economic achievement, according to Drucker. It is knowledge, not land, raw materials, or capital, that has become the central factor in production.

In his provocative discussions, Drucker makes extensive use of such terms as "knowledge organizations," "knowledge technologies," and "knowledge societies." It seemed a highly appropriate extension for us to coin "knowledge workshop" for re-naming the area of our special interest: the place in which knowledge workers do their work. Knowledge workshops have existed for centuries, but our special concern is their systematic improvement, toward increased effectiveness of this new breed of craftsmen.

Workshop improvement involves systematic change not only in the tools that help handle and transform the materials, but in the customs, conventions, skills, procedures, working methods, organizational roles, training, etc., by which the workers and their organizations harness their tools, their skills, and their knowledge.

Over the past ten years, the explicit focus in the Augmentation Research Center (ARC) has been upon the effects and possibilities of new knowledge workshop tools based on the technology of computer timesharing and modern communications.[18-41] Since we consider automating many human operations, what we are after could perhaps be termed "workshop automation." But the very great importance of aspects other than the new tools (i.e., conventions, methods, roles) makes us prefer the "augmentation" term that hopefully can remain "wholescope." We want to keep tools in proper perspective within the total system that augments native human capacities toward effective action.[1-3,10,16,18,24]

Development of more effective knowledge workshop technology will require talents and experience from many backgrounds: computer hardware and software, psychology, management science, information science, and operations research, to name a few. These must come together within the framework of a new discipline, focused on the systematic study of knowledge work and its workshop environments.

TWO WAYS IN WHICH AUGMENTED KNOWLEDGE WORKSHOPS ARE EVOLVING

Introduction

First, one can see a definite evolution of new workshop architecture in the trends of computer application systems. An "augmented workshop domain" will probably emerge because many special-purpose application systems are evolving by adding useful features outside their immediate special application area. As a result, many will tend to overlap in their general knowledge work supporting features.

Second, research and development is being directed toward augmenting a "Core" Knowledge Workshop domain. This application system development is aimed expressly at supporting basic functions of knowledge

work. An important characteristic of such systems is to interface usefully with specialized systems. This paper is oriented toward this second approach.

NATURAL EVOLUTION BY SCATTERED NUCLEI EXPANDING TOWARD A COMMON "KNOWLEDGE WORKSHOP" DOMAIN

Anderson and Coover[15] point out that a decade or more of application-system evolution is bringing about the beginning of relatively rational user-oriented languages for the control interfaces of advanced applications software systems. What is interesting to note is that the functions provided by the "interface control" for the more advanced systems are coming to include editors and generalized file-management facilities, to make easier the preparation, execution, and management of the special-purpose tools of such systems.

It seems probable that special application-oriented systems (languages) will evolve steadily toward helping the user with such associated work as formulating models, documenting them, specifying the different trial runs, keeping track of intermediate results, annotating them and linking them back to the users' model(s), etc. When the results are produced by what were initially the core application programs (e.g., the statistical programs), he will want ways to integrate them into his working notes, illustrating, labeling, captioning, explaining and interpreting them. Eventually these notes will be shaped into memoranda and formal publications, to undergo dialogue and detailed study with and by others.[15]

Once a significant user-oriented system becomes established, with a steady growth of user clientele, there will be natural forces steadily increasing the effectiveness of the system services and steadily decreasing the cost per unit of service. And it will also be natural that the functional domain of an application system will steadily grow outward: "as long as the information must be in computer form anyway for an adjacent, computerized process, let's consider applying computer aid to Activity X also."

> Because the boundary of the Application System has grown out to be "next to" Activity X, it has become relatively easy to consider extending the computerized-information domain a bit so that a new application process can support Activity X. After all, the equipment is already there, the users who perform Activity X are already oriented to use integrated computer aid, and generally the computer facilitation of Activity X will prove to have a beneficial effect on the productivity of the rest of the applications system.

This domain-spreading characteristic is less dependent upon the substantive work area a particular application system supports than it is upon the health and vitality of its development and application (the authors of Reference 15 have important things to say on these issues); however, it appears that continuing growth is bound to occur in many special application domains, inevitably bringing about overlap in common application "sub-domains" (as seen from the center of any of these nuclei). These special subdomains include formulating, studying, keeping track of ideas, carrying on dialogue, publishing, negotiating, planning, coordinating, learning, coaching, looking up in the yellow pages to find someone who can do a special service, etc.

CONSIDERING THE CORE KNOWLEDGE WORKSHOP AS A SYSTEM DOMAIN IN ITS OWN RIGHT

A second approach to the evolution of a knowledge workshop is to recognize from the beginning the amount and importance of human activity constantly involved in the "core" domain of knowledge work—activity within which more specialized functions are embedded.

If you asked a particular knowledge worker (e.g., scientist, engineer, manager, or marketing specialist) what were the foundations of his livelihood, he would probably point to particular skills such as those involved in designing an electric circuit, forecasting a market based on various data, or managing work flow in a project. If you asked him what tools he needed to improve his effectiveness he would point to requirements for aids in designing circuits, analyzing his data, or scheduling the flow of work.

> But, a record of how this person used his time, even if his work was highly specialized, would show that specialized work such as mentioned above, while vital to his effectiveness, probably occupied a small fraction of his time and effort.
>
> The bulk of his time, for example, would probably be occupied by more general knowledge work: writing and planning or design document; carrying on dialogue with others in writing, in person, or on the telephone; studying documents; filing ideas or other material; formulating problem-solving approaches; coordinating work with others; and reporting results.

There would seem to be a promise of considerable payoff in establishing a healthy, applications oriented systems development activity within this common, "core" domain, meeting the special-application systems "coming the other way" and providing them with well-designed services at a natural system-to-system interface.

It will be much more efficient to develop this domain explicitly, by people oriented toward it, and hopefully with resources shared in a coordinated fashion. The alternative of semi-random growth promises problems such as:

(1) Repetitive solutions for the same functional problems, each within the skewed perspective of a particular special-applications area for which these problems are peripheral issues,

(2) Incompatibility between diferent application software systems in terms of their inputs and outputs,

(3) Languages and other control conventions inconsistent or based on different principles from one system to another, creating unnecessary learning barriers or other discouragements to cross usage.

In summary, the two trends in the evolution of knowledge workshops described above are each valuable and are complementary. Experience and specific tools and techniques can and will be transferred between them.

There is a very extensive range of "core" workshop functions, common to a wide variety of knowledge work, and they factor into many levels and dimensions. In the sections to follow, we describe our developments, activities, and commitments from the expectation that there soon will be increased activity in this core knowledge workshop domain, and that it will be evolving "outward" to meet the other application systems "heading inward."

BASIC ASSUMPTIONS ABOUT AUGMENTED KNOWLEDGE WORKSHOPS EMBEDDED IN A COMPUTER NETWORK

The computer-based "tools" of a knowledge workshop will be provided in the environment of a computer network such as the ARPANET.[7,8,14] For instance, the core functions will consist of a network of cooperating processors performing special functions such as editing, publishing, communication of documents and messages, data management, and so forth. Less commonly used but important functions might exist on a single machine. The total computer assisted workshop will be based on many geographically separate systems.

Once there is a "digital-packet transportation system," it becomes possible for the individual user to reach out through his interfacing processor(s) to access other people and other services scattered throughout a "community," and the "labor marketplace" where he transacts his knowledge work literally will not have to be affected by geographical location.[27]

Specialty application systems will exist in the way that specialty shops and services now do—and for the same reasons. When it is easy to transport the material and negotiate the service transactions, one group of people will find that specilization can improve their cost/effectiveness, and that there is a large enough market within reach to support them. And in the network-coupled computer-resource marketplace, the specialty shops will grow—e.g., application systems specially tailored for particular types of analyses, or for checking through text for spelling errors, or for doing the text-graphic document typography in a special area of technical portrayal, and so on. There will be brokers, wholesalers, middle men, and retailers.

Coordinated set of user interface principles

There will be a common set of principles, over the many application areas, shaping user interface features such as the language, control conventions, and methods for obtaining help and computer-aided training.

This characteristic has two main implications. One, it means that while each domain within the core workshop area or within a specialized application system may have a vocabulary unique to its area, this vocabulary will be used within language and control structures common throughout the workshop system. A user will learn to use additional functions by increasing vocabulary, not by having to learn separate "foreign" languages. Two, when in trouble, he will invoke help or tutorial functions in a standard way.

Grades of user proficiency

Even a once-in-a-while user with a minimum of learning will want to be able to get at least a few straightforward things done. In fact, even an expert user in one domain will be a novice in others that he uses infrequently. Attention to novice-oriented features is required.

But users also want and deserve the reward of increased proficiency and capability from improvements in their skills and knowledge, and in their conceptual orientation to the problem domain and to their workshop's system of tools, methods, conventions, etc. "Advanced vocabularies" in every special domain will be important and unavoidable.

A corollary feature is that workers in the rapidly evolving augmented workshops should continuously be involved with testing and training in order that their skills and knowledge may harness available tools and methodology most effectively.

Ease of communication between, and addition of, workshop domains

One cannot predict ahead of time which domains or application systems within the workshop will want to communicate in various sequences with which others, or what operations will be needed in the future. Thus, results must be easily communicated from one set of operations to another, and it should be easy to add or interface new domains to the workshop.

User programming capability

There will never be enough professional programmers and system developers to develop or interface all the tools that users may need for their work. Therefore, it must be possible, with various levels of ease, for users to add or interface new tools, and extend the language to meet their needs. They should be able to do this in a variety of programming languages with which they may have training, or in the basic user-level language of the workshop itself.

Availability of people support services

An augmented workshop will have more support services available than those provided by computer tools.

There will be many people support services as well: besides clerical support, there will be extensive and highly specialized professional services, e.g., document design and typography, data base design and administration, training, cataloging, retrieval formulation, etc. In fact, the marketplace for human services will become much more diverse and active.[27]

Cost decreasing, capabilities increasing

The power and range of available capabilities will increase and costs will decrease. Modular software designs, where only the software tools needed at any given moment are linked into a person's run-time computer space, will cut system overhead for parts of the system not in use. Modularity in hardware will provide local configurations of terminals and miniprocessors tailored for economically fitting needs. It is obvious that cost of raw hardware components is plummeting; and the assumed large market for knowledge workshop support systems implies further help in bringing prices down.

The argument given earlier for the steady expansion of vital application systems to other domains remains valid for explaining why the capabilities of the workshop will increase. Further, increasing experience with the workshop will lead to improvements, as will the general trend in technology evolution.

Range of workstations and symbol representations

The range of workstations available to the user will increase in scope and capability. These workstations will support text with large, open-ended character sets, pictures, voice, mathematical notation, tables, numbers and other forms of knowledge representation. Even small portable hand-held consoles will be available.[13]

Careful development of methodology

As much care and attention will be given to the development, analysis, and evaluation of procedures and methodology for use of computer and people support services as to the development of the technological support services.

Changed roles and organizational structure

The widespread availability of workshop services will create the need for new organizational structures and roles.

SELECTED DESCRIPTION OF AUGMENTED WORKSHOP CAPABILITIES

Introduction

Within the framework described above, ARC is developing a prototype workshop system. Our system does not meet all the requirements outlined previously, but it does have a powerful set of core capabilities and experience that leads us to believe that such goals can be achieved.

Within ARC we do as much work as possible using the range of online capabilities offered. We serve not only as researchers, but also as the subjects for the analysis and evaluation of the augmentation system that we have been developing.

Consequently, an important aspect of the augmentation work done within ARC is that the techniques being explored are implemented, studied, and evaluated with the advantage of intensive everyday usage. We call this research and development strategy "bootstrapping."

In our experience, complex man-machine systems can evolve only in a pragmatic mode, within real-work environments where there is an appropriate commitment to conscious, controlled, exploratory evolution within the general framework outlined earlier. The plans and commitments described later are a consistent extension of this pragmatic bootstrapping strategy.

To give the reader more of a flavor of some of the many dimensions and levels of the ARC workshop, four example areas are discussed below in more detail, following a quick description of our physical environment.

The first area consists of mechanisms for studying and browsing through NLS files as an example of one functional dimension that has been explored in some depth.

The second area consists of mechanisms for collaboration support—a subsystem domain important to many application areas.

The third and fourth areas, support for software engineers and the ARPANET Network Information Center (NIC), show example application domains based on functions in our workshop.

General physical environment

Our computer-based tools run on a Digital Equipment Corporation PDP-10 computer, operating with the Bolt, Beranek, and Newman TENEX timesharing system.[9] The computer is connected via an Interface Message Processor (IMP) to the ARPANET.[7,8] There is a good deal of interaction with Network researchers, and with Network technology, since we operate the ARPA Network Information Center (see below).[39]

There is a range of terminals: twelve old, but serviceable, display consoles of our own design,[26] an IMLAC display, a dozen or so 30 ch/sec portable upper/lower case typewriter terminals, five magnetic tape-cassette storage units that can be used either online or offline, and a 96-character line printer. There are 125 million characters of online disk storage.

The display consoles are equipped with a typewriter-like keyboard, a five-finger keyset for one-handed character input, and a "mouse"—a device for controlling the position of a cursor (or pointer) on the display screen and for input of certain control commands. Test results on the mouse as a screen-

selection device have been reported in Reference 25, and good photographs and descriptions of the physical systems have appeared in References 20 and 21.

The core workshop software system and language, called NLS, provides many basic tools, of which a number will be mentioned below. It is our "core-workshop application system."

During the initial years of workshop development, application and analysis, the basic knowledge-work functions have centered around the composition, modification, and study of structured textual material.[26] Some of the capabilities in this area are described in detail in Reference 26, and are graphically shown in a movie available on loan[41]—

> The structured-text manipulation has been developed extensively because of its high payoff in the area of applications-system development to which we have applied our augmented workshop. We have delayed addition of graphic-manipulation capabilities because there were important areas associated with the text domain needing exploration and because of limitations in the display system and hardcopy printout.

To build the picture of what our Core Knowledge Workshop is like, we first give several in-depth examples, and then list in the section on workshop utility service some "workshop subsystems" that we consider to be of considerable importance to general knowledge work.

STUDYING ONLINE DOCUMENTS

Introduction

The functions to be described form a set of controls for easily moving one around in an information space and allowing one to adjust the scope, format, and content of the information seen.[26,41]

Given the addition of graphical, numerical, and vocal information, which are planned for addition to the workshop, one can visualize many additions to the concepts below. Even for strictly textual material there are yet many useful ideas to be explored.

View specifications

One may want an overview of a document in a table-of-contents like form on the screen. To facilitate this and other needs, NLS text files are hierarchically structured in a tree form with subordinate material at lower levels in the hierarchy.[26]

> The basic conceptual unit in NLS, at each node of the hierarchical file, is called a "statement" and is usually a paragraph, sentence, equation, or other unit that one wants to manipulate as a whole.
>
> A statement can contain many characters—presently, up to 2000. Therefore, a statement can contain many lines of text. Two of the "view-specification" parameters—depth in the hierarchy, and lines per statement—can be controlled during study of a document to give various overviews of it. View specifications are given with highly abbreviated control codes, because they are used very frequently and their quick specification and execution make a great deal of difference in the facility with which one studies the material and keeps track of where he is.

Examples of other view specifications are those that control spacing between statements, and indentation for levels in the hierarchy, and determine whether the identifications associated with statements are to be displayed, which branch(es) in the tree are to be displayed, whether special filters are to be invoked to show only statements meeting specified content requirements or whether statements are to be transformed according to special rules programmed by the user.

Moving in information space

A related viewing problem is designating the particular location (node in a file hierarchy) to be at the top of the screen. The computer then creates a display of the information from that point according to the view specifications currently in effect.

The system contains a variety of appropriate commands to do this; they are called jump commands because they have the effect of "jumping" or moving one from place to place in the network of files available as a user's information space.[26,33-39]

> One can point at a particular statement on the screen and command the system to move on to various positions relative to the selected one, such as up or down in the hierarchical structure, to the next or preceding statement at the same hierarchical level, to the first or last statement at a given level, etc.
>
> One can tell the system to move to a specifically named point or go to the next occurrence of a statement with a specific content.
>
> Each time a jump or move is made, the option is offered of including any of the abbreviated view specifications—a very general, single operation is "jump to that location and display with this view."

As one moves about in a file one may want to quickly and easily return to a previous view of the path as one traverses through the file and the specific view at each point, and then allowing return movement to the most recent points saved.

Another important feature in studying or browsing in a document is being able to quickly move to other documents cited.

There is a convention (called a "link") for citing documents that allows the user to specify a particular file, statement within the file and view specification for initial display when arriving in the cited file.

A single, quickly executed command (Jump to Link) allows one to point at such a citation, or anywhere in the statement preceding the citation, and the system will go to the specific file and statement cited and show the associated material with the specified view parameters. This allows systems of interlinked documents and highly specific citations to be created.

A piece of the path through the chain of documents is saved so that one can return easily a limited distance back along his "trail," to previously referenced documents. Such a concept was originally suggested by Bush[1] in a fertile paper that has influenced our thinking in many ways.

Multiple windows

Another very useful feature is the ability to "split" the viewing screen horizontally and/or vertically in up to eight rectangular display windows of arbitrary size. Generally two to four windows are all that are used. Each window can contain a different view of the same or different locations, within the same or different files.[39]

COLLABORATIVE DIALOGUE AND TELECONFERENCING

Introduction

The approach to collaboration support taken at ARC to date has two main thrusts:

(1) Support for real-time dialogue (teleconferencing) for two or more people at two terminals who want to see and work on a common set of material. The collaborating parties may be further augmented with a voice telephone connection as well.
(2) Support for written, recorded dialogue, distributed over time.

These two thrusts give a range of capabilities for support of dialogue distributed over time and space.

Teleconferencing support

Consider two people or groups of people who are geographically separated and who want to collaborate on a document, study a computer program, learn to use a new aspect of a system, or perform planning tasks, etc.

The workshop supports this type of collaboration by allowing them to link their terminals so that each sees the same information and either can control the system. This function is available for both display and typewriter terminal users over the ARPANET.

The technique is particularly effective between displays because of the high speed of information output and the flexibility of being able to split the screen into several windows, allowing more than one document or view of a document to be displayed for discussion.

When a telephone link is also established for voice communication between the participants, the technique comes as close as any we know to eliminating the need for collaborating persons or small groups to be physically together for sophisticated interaction.

A number of other healthy approaches to teleconferencing are being explored elsewhere.[11,12,16,17] It would be interesting to interface to such systems to gain experience in their use within workshops such as described here.

RECORDED DIALOGUE SUPPORT

Introduction

As ARC has become more and more involved in the augmentation of teams, serious consideration has been given to improving intra- and inter-team communication with whatever mixture of tools, conventions, and procedures will help.[27,36,39]

If a team is solving a problem that extends over a considerable time, the members will begin to need help in remembering some of the important communications—i.e., some recording and recalling processes must be invoked, and these processes become candidates for augmentation.

If the complexity of the team's problem relative to human working capacity requires partitioning of the problem into many parts—where each part is independently attacked, but where there is considerable interdependence among the parts—the communication between various people may well be too complex for their own accurate recall and coordination without special aids.

Collaborating teams at ARC have been augmented by development of a "Dialogue Support System (DSS)," containing current and thoroughly used working records of the group's plans, designs, notes, etc. The central feature of this system is the ARC Journal, a specially managed and serviced repository for files and messages.

The DSS involves a number of techniques for use by distributed parties to collaborate effectively both using general functions in the workshop and special functions briefly described below and more fully in Reference 39. Further aspects are described in the section on Workshop Utility Service.

Document or message submission

The user can submit an NLS file, a part of a file, a file prepared on another system in the ARPANET (document), or text typed at submission time (message)

to the Journal system. When submitted, a copy of the document or message is transferred to a read-only file whose permanent safekeeping is guaranteed by the Journal system. It is assigned a unique catalog number, and automatically cataloged. Later, catalog indices based on number, author, and "titleword out of context" are created by another computer process.

Nonrecorded dialogue for quick messages or material not likely to be referenced in the future is also permitted.

One can obtain catalog numbers ahead of time to interlink document citations for related documents that are being prepared simultaneously. Issuing and controlling of catalog numbers is performed by a Number System (an automatic, crash-protected computer process).

At the time of submission, the user can contribute such information as: title, distribution list, comments, keywords, catalog numbers of documents this new one supersedes (updates), and other information.

The distribution is specified as a list of unique identification terms (abbreviated) for individuals or groups. The latter option allows users to establish dialogue groups. The system automatically "expands" the group identification to generate the distribution list of the individuals and groups that are its members. Special indices of items belonging to subcollections (dialogue groups) can be prepared to aid their members in keeping track of their dialogue. An extension of the mechanisms available for group distribution could give a capability similar to one described by Turoff.[17]

Entry of identification information initially into the system, group expansion, querying to find a person's or group's identification, and other functions are performed by an Identification System.

Document distribution

Documents are distributed to a person in one, two, or all of three of the following ways depending on information kept by the Identification System.

(1) In hardcopy through the U.S. or corporation mail to those not having online access or to those desiring this mode,
(2) Online as citations (for documents) or actual text (for messages) in a special file assigned to each user.
(3) Through the ARPANET for printing or online delivery at remote sites. This delivery is performed using a standard Network wide protocol.

Document distribution is automated, with online delivery performed by a background computer process that runs automatically at specified times. Printing and mailing are performed by operator and clerical support. With each such printed document, an address cover sheet is automatically printed, so that the associated printout pages only need to be folded in half, stapled, and stamped before being dropped in the mail.

Document access

An effort has been made to make convenient both online and offline access to Journal documents. The master catalog number is the key to accessing documents. Several strategically placed hardcopy master and access collections (libraries) are maintained, containing all Journal documents.

Automatic catalog-generation processes generate author, number, and titleword indices, both online and in hardcopy.[38] The online versions of the indices can be searched conveniently with standard NLS retrieval capabilities.[37,39,41]

Online access to the full text of a document is accomplished by using the catalog number as a file name and loading the file or moving to it by pointing at a citation and asking the system to "jump" there as described earlier.

SOFTWARE ENGINEERING AUGMENTATION SYSTEM

Introduction

One of the important application areas in ARC's work is software engineering. The economics of large computer systems, such as NLS, indicate that software development and maintenance costs exceed hardware costs, and that software costs are rising while hardware costs are rapidly decreasing. The expected lifetime of most large software systems exceeds that of any piece of computer hardware. Large software systems are becoming increasingly complex, difficult to continue evolving and maintain. Costs of additional enhancements made after initial implementation generally exceed the initial cost over the lifetime of the system. It is for these reasons that it is important to develop a powerful application area to aid software engineering. Areas of software engineering in which the ARC workshop offers aids are described below.

Design and review collaboration

During design and review, the document creation, editing, and studying capabilities are used as well as the collaboration, described above.

Use of higher level system programming languages

Programming of NLS is performed in a higher level ALGOL-like system programming language called L-10 developed at ARC. The L-10 language compiler takes its input directly from standard NLS structured files. The PDP-10 assembler also can obtain input from NLS files.

It is planned to extend this capability to other languages, for example, by providing an interface to the BASIC system available in our machine for knowledge workers wishing to perform more complex numerical tasks.

We are involved with developing a modular runtime-linkable programming system (MPS), and with planning a redesign of NLS to utilize MPS capabilities, both in cooperation with the Xerox Palo Alto Research Center. MPS will:

(1) Allow a workshop system organization that will make it easier for many people to work on and develop parts of the same complex system semi-independently.
(2) Make it easier to allow pieces of the system to exist on several processors.
(3) Allow individual users or groups of users to tailor versions of the system to their special needs.
(4) Make it easier to move NLS to other computers since MPS is written in itself.
(5) Speed system development because of MPS's improved system building language facilities, integrated source-level debugging, measurement facilities, the ability to construct new modules by combining old ones, and to easily modify the system by changing module interconnection.

System documentation and source-code creation

Source-code creation uses the standard NLS hierarchical file structures and allows documentation and other programming conventions to be established that simplify studying of source-code files.

Debugging

A form of source-level debugging is allowed through development of several tools, of which the following are key examples:

(1) A user program compilation and link loading facility that allows new or replacement programs to be linked into the running system to create revised versions for testing or other purposes.
(2) NLS-DDT, a DDT like debugging facility with a command language more consistent with the rest of NLS, and simplifies display of system variables and data structures, and allows replacement of system procedures by user supplied procedures.
(3) Use of several display windows so as to allow source code in some windows and control of DDT in others for the setting of breakpoints and display of variables and data structures.

Measurement and analysis

A range of measurement tools has been developed for analyzing system operation. These include the following:

(1) Capabilities for gathering and reporting statistics on many operating system parameters such as utilization of system components in various modes, queue lengths, memory utilization, etc.
(2) The ability to sample the program counter for intervals of a selectable area of the operating system or any particular user subsystem to measure time spent in the sampled areas;
(3) Trace and timing facilities to follow all procedure calls during execution of a specified function.
(4) The ability to study page-faulting characteristics of a subsystem to check on its memory use characteristics.
(5) The ability to gather NLS command usage and timing information.
(6) The ability to study user interaction on a task basis from the point of view of the operating-system scheduler.
(7) The ability to collect sample user sessions for later playback to the system for simulated load, or for analysis.

Maintenance

Maintenance programmers use the various functions mentioned above. The Journal is used for reporting bugs; NLS structured source code files simplify the study of problem areas and the debugging tools permit easy modification and testing of the modifications.

THE ARPA NETWORK INFORMATION CENTER (NIC)

Introduction

The NIC is presently a project embedded within ARC.[39] Workshop support for the NIC is based on the capabilities within the total ARC workshop system.

As useful as is the bootstrapping strategy mentioned earlier, there are limits to the type of feedback it can yield with only ARC as the user population. The NIC is the first of what we expect will be many activities set up to offer services to outside users. The goal is to provide a useful service and to obtain feedback on the needs of a wider class of knowledge workers. Exercised within the NIC are also prototypes of information services expected to be normal parts of the workshop.

The NIC is more than a classical information center, as that term has come to be used, in that it provides a wider range of services than just bibliographic and "library" type services.

The NIC is an experiment in setting up and running a general purpose information service for the ARPANET community with both online and offline services. The services offered and under development by the NIC have as their initial basic objectives:

(1) To help people with problems find the resources (people, systems, and information) available within the network community that meet their needs.

(2) To help members of geographically distributed groups collaborate with each other.

Following are the NIC services now provided to meet the above goals in serving the present clientele:

Current online services

(1) Access to the typewriter version (TNLS) and display version (DNLS) of the Augmentation Research Center's Online System (NLS) for communique creation, access, and linking between users, and for experimental use for any other information storage and manipulation purpose suitable for NLS and useful to Network participants.
(2) Access to Journal, Number, and Identification Systems to allow messages and documents to be transmitted between network participants.
(3) Access to a number of online information bases through a special Locator file using NLS link mechanisms and through a novice-oriented query system.

Current offline services

(1) A Network Information Center Station set up at each network site.
(2) Techniques for gathering, producing and maintaining data bases such as bibliographic catalogs, directories of network participants, resource information, and user guides.
(3) Support of Network dialogue existing in hardcopy through duplication, distribution, and cataloging.
(4) General Network referral and handling of document requests.
(5) Building of a collection of documents potentially valuable to the Network Community. Initial concentration has been on obtaining documents of possible value to the Network builders.
(6) As yet primitive selective document distribution to Station Collections.
(7) Training in use of NIC services and facilities.

Conclusion

The Network Information Center is an example prototype of a new type of information service that has significant future potential. Even though it is presently in an experimental and developmental phase, it is providing useful online and offline services to the ARPANET community.

PLANS FOR A WORKSHOP UTILITY SERVICE

Motivation

It is now time for a next stage of application to be established. We want to involve a wider group of people so that we can begin to transfer the fruits of our past work to them and with their assistance, to others, and so that we can obtain feedback needed for further evolution from wider application than is possible in our project alone.[28] We want to find and support selected groups who are willing to take extra trouble to be exploratory, but who:

(1) Are not necessarily oriented to being core-workshop developers (they have their own work to do).
(2) Can see enough benefit from the system to be tried and from the experience of trying it so that they can justify the extra risk and expense of being "early birds."
(3) Can accept assurance that system reliability and stability, and technical/application help will be available to meet their conditions for risk and cost.

ARC is establishing a Workshop Utility Service, and promoting the type of workshop service described above as part of its long-term commitment to pursue the continued development of augmented knowledge workshops in a pragmatic, evolutionary manner.

It is important to note that the last few years of work have concentrated on the means for delivering support to a distributed community, for providing teleconferencing and other basic processes of collaborative dialogue, etc. ARC has aimed consciously toward developing experience and capabilities especially applicable to support remote and distributed groups of exploratory users for this next stage of wider-application bootstrapping.

One aspect of the service is that it will be an experiment in harnessing the new environment of a modern computer network to increase the feasibility of a wider community of participants cooperating in the evolution of an application system.

Characteristics of the planned service

The planned service offered will include:

(1) Availability of Workshop Utility computer service to the user community from a PDP-10 TENEX system operated by a commercial supplier.
(2) Providing training as appropriate in the use of Display NLS (DNLS), Typewriter NLS (TNLS), and Deferred Execution (DEX) software subsystems.
(3) Providing technical assistance to a user organization "workshop architect" in the formulation, development, and implementation of augmented knowledge work procedures within selected offices at the user organization.[6]

This assistance will include help in the development of NLS use strategies suitable to the user environments, procedures within the user organization for implementing these strategies, and possible special-application NLS extensions (or

simplifications) to handle the mechanics of particular user needs and methodologies.

(4) Providing "workshop architect" assistance to help set up and assist selected geographically distributed user groups who share a special discipline or mission orientation to utilize the workshop utility services and to develop procedures, documentation, and methodology for their purposes.

GENERAL DESCRIPTION OF SOME WORKSHOP UTILITY SUBSYSTEMS

Introduction

Within a particular professional task area (mission- or discipline-oriented) there are often groups who could be benefited by using special workshop subsystems. These subsystems may be specialized for their specific application or research domain or for support of their more general knowledge work. Our goal is to offer a workshop utility service that contains a range of subsystems and associated methodology particularly aimed at aiding general knowledge work, and that also supports in a coordinated way special application subsystems either by interfacing to subsystems already existing, or by developing new subsystems in selected areas.

In the descriptions to follow are a number of workshop subsystem domains that are fundamental to a wide range of knowledge work in which ARC already has extensive developments or is committed to work. For each subsystem we include some general comments as well as a brief statement of current ARC capabilities in the area.

Document development, production, and control

Here a system is considered involving authors, editors, supervisors, typists, distribution-control personnel, and technical specialists. Their job is to develop documents, through successive drafts, reviews, and revisions. Control is needed along the way of bibliography, who has checked what point, etc. Final drafts need checkoff, then production. Finally distribution needs some sort of control. If it is what we call a "functional document" such as a user guide, then it needs to be kept up to date.[39] There is a further responsibility to keep track of who needs the documents, who has what version, etc.

Within the ARC workshop, documents ranging from initial drafts to final high-quality printed publications can be quickly produced with a rich set of creation and editing functions. All of ARC's proposals, reports, designs, letters, thinkpieces, user documentation, and other such information are composed and produced using the workshop.

Documents in a proof or finished form can be produced with a limited character set and control on a line printer or typewriter, or publication-quality documents can be produced on a photocomposer microfilm unit.

Presently there are on the order of two hundred special directives that can be inserted in text to control printing. These directives control such features as typefont, pagination, margins, headers, footers, statement spacing, typefont size and spacing, indenting, numbering of various hierarchical levels, and many other parameters useful for publication quality work. Methodology to perform the creation, production, and controlling functions described above has been developed, although much work at this level is still needed.

In terms of future goals, one would like to have display terminals with a capability for the range of fonts available on the photocomposer so that one could study page layout and design interactively, showing the font to be used, margins, justification, columnization, etc. on the screen rather than having to rely on hardcopy proofsheets.

To prepare for such a capability, plans are being made to move toward an integrated portrayal mechanism for both online and hardcopy viewing.

Collaborative dialogue and teleconferencing

Effective capabilities have already been developed and are in application, as discussed above. There is much yet to do. The Dialogue Support System will grow to provide the following additional general online aids:

Link-setup automation; back-link annunciators and jumping; aids for the formation, manipulation, and study of sets of arbitrary passages from among the dialogue entries; and integration of cross-reference information into hardcopy printouts. Interfaces will probably be made to other teleconferencing capabilities that come into existence on the ARPANET.

It also will include people-system developments: conventions and working procedures for using these aids effectively in conducting collaborative dialogue among various kinds of people, at various kinds of terminals, and under various conditions; working methodology for teams doing planning, design, implementation coordination; and so on.

Meetings and conferences

Assemblies of people are not likely for a long time, if ever, to be supplanted in total by technological aids. Online conferences are held at ARC for local group meetings and for meetings where some of the participants are located across the country.

Use is made of a large-screen projection TV system to provide a display image that many people in a conference room can easily see. This is controlled locally or remotely by participants in the meeting, giving access to the entire recorded dialogue data base as needed during the meeting and also providing the capability of recording real-time

meeting notes and other data. The technique also allows mixing of other video signals.

Management and organization

The capabilities offered in the workshop described in this paper are used in project management and administration.[39] Numerical calculations can also be performed for budget and other purposes, obtaining operands and returning results to NLS files for further manipulation.

Where an organization has conventional project management operations, their workshop can include computer aids for techniques such as PERT and CPM. We want to support the interfacing that our Core Workshop can provide to special application systems for management processes.

We are especially interested at this stage, in management of project teams—particularly, of application-systems development teams.

Handbook development

Capabilities described above are being extended toward the coordinated handling of a very large and complex body of documentation and its associated external references. The goal is that a project or discipline of ever-increasing size and complexity can be provided with a service that enables the users to keep a single, coordinated "superdocument" in their computer; that keeps up to date and records the state of their affairs; and provides a description of the state of the art in their special area.

Example contents would be glossaries, basic concept structure, special analytic techniques, design principles, actual design, and implementation records of all developments.

Research intelligence

The provisions within the Dialogue Support System for cataloging and indexing internally generated items also support the management for externally generated items, bibliographies, contact reports, clippings, notes, etc. Here the goal is to give a human organization (distributed or local) an ever greater capability for integrating the many input data concerning its external environment; processing (filtering, transforming, integrating, etc.) the data so that it can be handled on a par with internally generated information in the organization's establishing of plans and goals; and adapting to external opportunities or dangers.[38]

Computer-based instruction

This is an important area to facilitate increasing the skills of knowledge workers. ARC has as yet performed little direct work in this area. We hope in the future to work closely with those in the computer-based instruction area to apply their techniques and systems in the workshop domain.

In training new and developing users in the use of the system, we have begun using the system itself as a teaching environment. This is done locally and with remote users over the ARPANET.

Software engineering augmentation

A major special application area described above, that has had considerable effort devoted to it, is support of software engineers. The software-based tools of the workshop are designed and built using the tools previously constructed. It has long been felt[24,29] that the greatest "bootstrapping" leverage would be obtained by intensively developing the augmented workshop for software engineers, and we hope to stimulate and support more activity in this area.

Knowledge workshop analysis

Systematic analysis has begun of the workshop environment at internal system levels, at user usage levels, and at information-handling procedure and methodology levels. The development of new analytic methodology and tools is a part of this process. The analysis of application systems, and especially of core-workshop systems, is a very important capability to be developed. To provide a special workshop subsystem that augments this sort of analytic work is a natural strategic goal.

CONCLUSION—THE NEED FOR LONG-TERM COMMITMENT

As work progresses day-to-day toward the long-term goal of helping to make the truly augmented knowledge workshop, and as communities of workshop users become a reality, we at ARC frequently reflect on the magnitude of the endeavor and its long-term nature.[22]

> Progress is made in steps, with hundreds of short-term tasks directed to strategically selected subgoals, together forming a vector toward our higher-level goals.

To continue on the vector has required a strong commitment to the longer-range goals by the staff of ARC.

In addition, we see that many of the people and organizations we hope to enlist in cooperative efforts will need a similar commitment if they are to effectively aid the process.

> One of ARC's tasks is to make the long-term objectives of the workshop's evolutionary development, the potential value of such a system, and the strategy for realizing that value clear enough to the collaborators we seek, so that they will have a strong commitment to invest resources with understanding and patience.

One key for meeting this need will be to involve them in serious use of the workshop as it develops. The plans for the Workshop Utility are partly motivated by this objective.

Although the present ARC workshop is far from complete, it does have core capabilities that we feel will greatly aid the next communities of users in their perception of the value of the improved workshops of the future.

ACKNOWLEDGMENTS

During the 10 year life of ARC many people have contributed to the development of the workshop described here. There are presently some 35 people—clerical, hardware, software, information specialists, operations researchers, writers, and others—all contributing significantly toward the goals described here.

The work reported here is currently supported primarily by the Advanced Research Projects Agency of the Department of Defense, and also by the Rome Air Development Center of the Air Force and by the Office of Naval Research.

REFERENCES

1. Bush, V., "As We May Think," *Atlantic Monthly*, pp. 101-108, July 1945 (SRI-ARC Catalog Item 3973).
2. Licklider, J. C. R., "Man-Computer Symbiosis," *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-1, pp. 4-11, March, 1960 (SRI-ARC Catalog Item 6342).
3. Drucker, P. F., *The Effective Executive*, Harper and Row, New York, 1967 (SRI-ARC Catalog Item 3074).
4. Drucker, P. F., *The Age of Discontinuity—Guidelines to our Changing Society*, Harper and Row, New York, 1968 (SRI-ARC Catalog Item 4247).
5. Dalkey, N., *The Delphi Method—An Experimental Study of Group Opinion*, Rand Corporation Memorandum RM-5888-PR, 1969 (SRI-ARC Catalog Item 3896).
6. Allen, T. J., Piepmeier, J. M., Cooney, S., "Technology Transfer to Developing Countries—The International Technological Gatekeeper," Proceedings of the ASIS, Vol. 7, pp. 205-210, 1970 (SRI-ARC Catalog Item 13959).
7. Roberts, L. G., Wessler, B. D., "Computer Network Development to Achieve Resource Sharing," *AFIPS Proceedings*, Spring Joint Computer Conference, Vol. 36, pp. 543-549, 1970 (SRI-ARC Catalog Item 4564).
8. Roberts, L. G., Wessler, B. D., *The ARPA Network*, Advanced Research Projects Agency, Information Processing Techniques, Washington, D.C. May 1971 (SRI-ARC Catalog Item 7750).
9. Bobrow, D. G., Burchfiel, J. D., Murphy, D. L., Tomlinson, R. S., "TENEX—A Paged Time Sharing System for the PDP-10," presented at *ACM Symposium on Operating Systems Principles*, October 18-20, 1971. Bolt Beranek and Newman Inc., August 15, 1971 (SRI-ARC Catalog Item 7736).
10. Weinberg, G. M., *The Psychology of Computer Programming*, Van Nostrand Reinhold Company, New York, 1971 (SRI-ARC Catalog Item 9036).
11. Hall, T. W., "Implementation of an Interactive Conference System," *AFIPS Proceedings*, Spring Joint Computer Conference, Vol. 38, pp. 217-229, 1971 (SRI-ARC Catalog Item 13962).
12. Turoff, M., "Delphi and its Potential Impact on Information Systems," *AFIPS Proceedings*, Fall Joint Computer Conference, Vol. 39, pp. 317-326, 1971 (SRI-ARC Catalog Item 7966).
13. Roberts, L. G., *Extensions of Packet Communication Technology to a Hand Held Personal Terminal*, Advanced Research Projects Agency, Information Processing Techniques, January 24, 1972 (SRI-ARC Catalog Item 9120).
14. Kahn, R. E., "Resource-Sharing Computer Communication Networks," *Proceedings of the IEEE*, Vol. 147, pp. 147, September 1972 (SRI-ARC Catalog Item 13958).
15. Anderson, R. E., Coover, E. R., "Wrapping Up the Package—Critical Thoughts on Applications Software for Social Data Analysis," *Computers and Humanities*, Vol. 7, No. 2, pp. 81-95, November 1972 (SRI-ARC Catalog Item 13956).
16. Lipinski, A. J., Lipinski, H. M., Randolph, R. H., "Computer Assisted Expert Interrogation—A Report on Current Methods Development," *Proceedings of First International Conference on Computer Communication*, Winkler, Stanley (ed), October 24-26, 1972, Washington, D.C., pp. 147-154 (SRI-ARC Catalog Item 11980).
17. Turoff, M., " 'Party-Line' and 'Discussion' Computerized Conference Systems," *Proceedings of First International Conference on Computer Communication*, Winkler, Stanley (ed), October 24-26, 1972, Washington, D. C., pp. 161-171, (SRI-ARC Catalog Item 11983).

BY OTHER PEOPLE, WITH SUBSTANTIVE DESCRIPTION OF ARC DEVELOPMENTS

18. Licklider, J. C. R., Taylor, R. W., Herbert, E., "The Computer as a Communication Device," *International Science and Technology*, No. 76, pp. 21-31, April 1968 (SRI-ARC Catalog Item 3888).
19. Engelbart, D. C., "Augmenting your Intellect," (Interview with D. C. Engelbart), *Research/Development*, pp. 22-27, August 1968 (SRI-ARC Catalog Item 9698).
20. Haavind, R., "Man-Computer 'Partnerships' Explored," *Electronic Design*, Vol. 17, No. 3, pp. 25-32, February 1, 1969 (SRI-ARC Catalog Item 13961).
21. Field, R. K., "Here Comes the Tuned-In, Wired-Up, Plugged-In, Hyperarticulate Speed-of-Light Society—An Electronics Special Report: No More Pencils, No More Books—Write and Read Electronically," *Electronics*, pp. 73-104, November 24, 1969 (SRI-ARC Catalog Item 9705).
22. Lindgren, N., "Toward the Decentralized Intellectural Workshop," *Innovation*, No. 24, pp. 50-60, September 1971 (SRI-ARC Catalog Item 10480).

OPEN-LITERATURE ITEMS BY ARC STAFF

23. Engelbart, D. C., "Special Considerations of the Individual as a User, Generator, and Retriever of Information," *American Documentation*, Vol. 12, No. 2, pp. 121-125, April 1961 (SRI-ARC Catalog Item 585).
24. Engelbart, D. C., "A Conceptual Framework for the Augmentation of Man's Intellect," *Vistas in Information Handling*, Howerton and Weeks (eds), Spartan Books, Washington, D.C., 1963, pp. 1-29 (SRI-ARC Catalog Item 9375).
25. English, W. K., Engelbart, D. C., Berman, M. A., "Display-Selection Techniques for Text Manipulation," *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-8, No. 1, pp. 5-15, March 1967 (SRI-ARC Catalog Item 9694).
26. Engelbart, D. C., English, W. K., "A Research Center for Augmenting Human Intellect," *AFIPS Proceedings*, Fall Joint Computer Conference, Vol. 33, pp. 395-410, 1968 (SRI-ARC Catalog Item 3954).
27. Engelbart, D. C., "Intellectual Implications of Multi-Access Computer Networks," paper presented at *Interdisciplinary Conference on Multi-Access Computer Networks*, Austin, Texas, April 1970, preprint (SRI ARC Journal File 5255).

28. Engelbart, D. C., *Coordinated Information Services for a Discipline- or Mission-Oriented Community,* Stanford Research Institute Augmentation Research Center, December 12, 1972 (SRI-ARC Journal File 12445). Also published in "Time Sharing—Past, Present and Future," *Proceedings of the Second Annual Computer Communications Conference* at California State University, San Jose, California, January 24-25, 1973, pp. 2.1-2.4, 1973. Catalog Item 5139).

RELEVANT ARC REPORTS

29. Engelbart, D. C., *Augmenting Human Intellect—A Conceptual Framework*, Stanford Research Institute Augmentation Research Center, AFOSR-3223, AD-289 565, October 1962 (SRI-ARC Catalog Item 3906).
30. Engelbart, D. C., Huddart, B., *Research on Computer-Augmented Information Management (Final Report)*, Stanford Research Institute Augmentation Research Center, ESD-TDR-65-168, AD 622 520, March 1965 (SRI-ARC Catalog Item 9690).
31. Engelbart, D. C., *Augmenting Human Intellect—Experiments, Concepts, and Possibilities—Summary Report*, Stanford Research Institute Augmentation Research Center, March 1965 (SRI-ARC Catalog Item 9691).
32. English, W. K., Engelbart, D. C., Huddart, B., *Computer Aided Display Control—Final Report*, Stanford Research Institute Augmentation Research Center, July 1965 (SRI-ARC Catalog Item 9692).
33. Engelbart, D. C., English, W. K., Rulifson, J. F., *Development of a Multidisplay*, Time-Shared Computer Facility and Computer-Augmented Management-System Research, Stanford Research Institute Augmentation Research Center, AD 843 577, April 1968 (SRI-ARC Catalog Item 9697).
34. Engelbart, D. C., *Human Intellect Augmentation Techniques—Final Report*, Stanford Research Institute Augmentation Research Center, CR-1270 N69-16140, July 1968 (SRI-ARC Catalog Item 3562).
35. Engelbart, D. C., English, W. K., Evans, D. C., *Study for the Development of Computer Augmented Management Techniques—Interim Technical Report*, Stanford Research Institute Augmentation Research Center, RADC-TR-69-98, AD 855 579, March 8, 1969 (SRI-ARC Catalog Item 9703).
36. Engelbart, D. C., SRI-ARC Staff, *Computer-Augmented Management-System Research and Development of Augmentation Facility—Final Report*, Stanford Research Institute Augmentation Research Center, RADC-TR-70-82, April 1970 (SRI-ARC Catalog Item 5139).
38. Engelbart, D. C., *Experimental Development of a Small Computer-Augmented Information System—Annual Report,* Stanford Research Institute Augmentation Research Center, April 1972 (SRI-ARC Catalog Item 10045).
39. *Online Team Environment—Network Information Center and Computer Augmented Team Interaction*, Stanford Research Institute Augmentation Research Center, RADC-TR-72-232, June 8, 1972 (SRI-ARC Journal File 13041).

RELEVANT ARTICLES IN ARC/NIC JOURNAL

40. Engelbart, D. C., *SRI-ARC Summary for IPT Contractor Meeting*, San Diego, January 8-10, 1973, Stanford Research Institute Augmentation Research Center, January 7, 1973 (SRI-ARC Journal File 13537).

MOVIE AVAILABLE FROM ARC FOR LOAN

41. *Augmentation of the Human Intellect*—A Film of the SRI-ARC Presentation at the 1969 ASIS Conference, San Francisco (A 3-Reel Movie, Stanford Research Institute Augmentation Research Center, October 1969 (SRI-ARC Catalog Item 9733).

Graphics, problem solving and virtual systems

by R. M. DUNN

U.S. Army Electronics Command
Fort Monmouth, New Jersey

INTRODUCTION

Man naturally uses many languages when he thinks creatively. Interactive computing mechanisms intended to augment man's higher faculties must provide for appropriate man-machine dialogues. The mechanisms must not constrain man's linguistic expressiveness for communication in the dialogues. To do so is to limit or retard the creative activity.

This paper explores some basic concepts for problem solving through interactive computing. Characteristics of the interactive access process and over-all system concepts are discussed. The evolution of a recognition automaton is proposed based on current work toward a multiconsole, interactive graphics Design Terminal.

BASIC CONCEPTS

Certain notions about problem solving, virtual systems, use-directed specification and interactive graphics are central to the concluding proposal of this discussion. These notions do not all reflect the current state-of-the-art or even that of the very near future. However, they do characterize the objectives and capabilities that should be the goals of interactive computing mechanism research development and use.

Problem solving

"Problem solving" is considered to be a process that involves creative thinking and discovery. Problem solving in a computer-based system is considered to be the activity of a human exploring a concept or system for which a computer-based description has been or is being devised. The human tries to perceive, alter and/or assess the description, behavior, performance or other quality of the concept or system. Very often the system or concept has time-dependent characteristics which add to its complexity.

The essence of the problem solving process is variation and observation of the system under study. Alexander[1] pointed out that human cognition functions to discover and identify the "misfit variable" that is the cause of the problem. To do so, the human needs to "toy" with the system so that he has a "feel" for its characteristics in terms of personal judgments that may be quite subjective.

Interactive computing mechanisms in problem solving situations should extend and amplify man's basic abilities for creative thinking and discovery. These mechanisms should improve his ability to perceive previously unrecognized characteristics. They should permit and support man's definition of new and meaningful symbols by which he designates his perceptions. They should aid in making any specification of values he chooses to assign to his symbolized perceptions. And, interactive computing mechanisms should aid in specifying and retaining any combination of evaluated, symbolized perceptions. Of particular interest are the combinations that man's creative faculty perceives as being related so as to form a higher order entity.

Virtual systems

A virtual system is considered to be an organized, temporary collection of resources that is created for a specific transient purpose.

Computer-based virtual systems combine processes, processors, data storage mechanisms. Interactive, computer-based virtual systems are considered to include people as another type of resource. The specific purposes that generate virtual systems are considered to be functionally classifiable. As a result, one can associate a specific purpose with a type of virtual system in terms of the function of the virtual system, or the process that carried out its formation, or the structure of its physical or functional organization or any combination of these attributes.

The resources that are available for use in a computer-based virtual system may be centralized or widely distributed. Today's trend points to the general case for the future as being distributed resources interconnected by communications networks. Network-oriented, computer-based virtual systems are extensible by the simple expedient of interconnecting more and/or different resources. The problems associated with the design and control of extensible distributed computing systems were investigated as early as 1965 by Cave and Dunn,[2] and since then by many others.[3,4,5,6,7,8,9,10]

For problem solving processes that incorporate interactive computing mechanisms, a particular type of computer-based, network-oriented virtual system is of specific interest. This system type exhibits two hierarchical characteristics. First, it allows and supports a hierarchy of functional uses to which it may be put. And second, it also embodies the capacity to interconnect and support access to resources on a hierarchical basis.

Use-directed specification

Use-directed specification is considered to be a process within a time-ordered activity. The subjects and history of activity determine the semantics and pragmatics of the specification.[11] In this context, semantics is taken to be the process of identifying relations between elements of a specification and the intents that are to be signified by those elements. Pragmatics is taken to be the process of identifying the extent and manner by which the signified intents can be of value to the time-ordered activity. For activities that are not deterministic, the semantic and pragmatic processes establish the operational context and effect of use-directed specifications on a probablistic basis.

Use-directed specification presumes an identified system of pragmatic values based upon the goals of the activity. For many time-ordered activities, the goals are unclear. Therefore, the associated system of pragmatic values are poorly defined or may not exist at all at the start of the activity. Such activities require rules of thumb, strategies, methods or tricks that are used as guides until the goals and pragmatic value system are established. This heuristic[12] approach requires a feedback mechanism as part of the means by which the activity is conducted. Feedback is used to provide information which may lead to adjustments in the heuristics and clarification of the activity's goals and pragmatic value system.

Adaptive, use-directed specification will be used to characterize activities that operate in the manner just described. Adaptive, use-directed specifications are of particular interest for problem solving activities that incorporate interactive mechanisms in the environment of network-oriented, computer-based virtual systems with hierarchical characteristics.

Interactive graphics

Interactive graphics is considered to be a computer-based process with the human "in-the-loop." "Interactive" describes the relation between the human and the computer-based process. The interactive relation is characterized by a *rate of response to human service requests that is both useful and satisfying to the human.* If the rate is too fast, the substance of the response may not be useful to the human. If the rate is too slow, the human may not be satisfied. Dunn, [13] Boehm, et al.,[14] and many others have explored detailed characteristics of interaction in a graphics environment.

Interactive graphics is considered to have three principal purposes.[15] One purpose is to improve the quality and rate of the input/output relation between people and machines. Another purpose is to provide assistance to people during detailed specification of some particular abstract representation. The remaining purpose is to provide assistance to people in visualizing and evaluating some attribute, behavior or performance of a specified abstract representation.

All three purposes, but especially the latter two, are of particular interest for problem solving activities that incorporate interactive computing mechanisms.

VIRTUAL SYSTEM ACCESS AND INTERACTIVE GRAPHICS

Most interactive computing systems contain an inherent assumption about certain knowledge required of the users. In some systems, the assumption is open and stated. In others, a less obvious, more troublesome situation may exist. Users of interactive computing systems rarely can consider the system as a "black box" into which parameter identification and values are entered and from which problem solving results are received. Most often the user is minimally required to explicitly know: the "black box" function to be used; the identification of the shared main computer that supports the "black box" function; the way in which the function must be requested; the way in which service on the supporting computer must be requested; and the type of information that must be provided to the function and the supporting computer. Some interactive systems require even more of the user.

The user of most of today's interactive systems can reasonably be required to have knowledge of the kind referred to above. However, when one considers the types of interactive systems that are likely to exist tomorrow, these requirements are not merely unreasonable, they may be impossible to be satisfied by typical users.

Consider the use of interactive computing mechanisms by problem solving activities via an extensible, network-oriented, distributed resource computing system. Over time, such a system will undergo significant changes in the number, type and pattern of dispersion of resources that are inter-connected. For an individual user, as his efforts progress or change, the combinations of resources appropriate to his purposes will also change. Any economic implementation of such a system will not be free of periods or instances when "busy signals" are encountered in response to requests for service. Therefore, it is likely that most resources will have some level of redundancy in the system.

The following conclusion must be drawn. If the human user of interactive computing systems must continue to satisfy today's requirements in the environment of tomorrow's systems, then the enormous potential of these sys-

tems will be lost to the user. It appears that this conclusion can be obviated. If one analyzes interactive access characteristics along with system functions and relations in a certain way, it appears feasible to reduce the burden of system knowledge upon the user to a manageable level in the environment of sophisticated interactive networks of the future.

Interactive access performance characteristics

The basic motivation for interactive access is to allow people to function as on-line controllers and participants in the computing process. Consequently, we must consider characteristics of interactive access mechanisms from the view of both human and system performance. Further, if we consider performance characteristics in the context of a complex process such as "problem solving" then, in a very loose sense, we have taken a "worst case" approach.

The first thing to consider is that interaction is carried on by means of a dialogue. This implies the existence of a language known to both parties. The question is—what should be the scope of reference of this language? Should it be the mechanisms of computing? Or the functioning of the interactive device? Or the topics which give rise to the body of information pertinent to the problem to be solved?

Ideally, one should not need to be concerned with computing mechanisms or interactive devices, but only with information relevant to the problem. Practically, one may want or need at least initial and, perhaps, refresher information on mechanisms and devices. One can then conclude that the principal concern of the language should be the topics which relate to the problem. The discourse should permit tutorial modes or inquiry dialogues on other issues *only at the specific request of the user.* Raphael's[16] work and that of others have established a solid foundation for the inquiry capability.

But, what of the problem solving topics? Should a separate language exist for each one? Could that be feasible as the domain of interactive problem solving expands? Clearly, it is not even feasible with today's primitive use. Tomorrow's uses will make this matter worse. It may be equally unreasonable to expect that machine systems can be provided with a human's linguistic faculty for some time. However, there are at least two feasible approximations.

The first is exemplified by MATHLAB.[17] In this effort, the machine is being programmed with the rules of analytical mathematics. Then the user interactively writes a mathematical equation on a machine sensible surface, the equation is solved analytically by the machine and the graph of the solution is displayed on an interactive graphics device. The process also requires that the machine is programmed to recognize handwritten entries. It does this task imperfectly and has to be corrected through re-entry of the symbols. The sensible input surface and the visible output surface together form the interactive mechanism of feedback until man and machine have reached agreement. A related example is the "turtle" language of Papert's[18] Mathland.

This first type of approximation provides a linguistic mechanism for a broad topic of discourse *and in addition,* provides an interactive feedback mechanism that allows man and machine to work out misunderstandings in the dialogue.

The second approximation is exemplified by the work of Pendergraft.[11,19] In this effort, the main concern became the evolution of computer-based, linguistic systems of a certain kind—a semiotic system. These systems are based on semiotics, the science of linguistic and other signs and how they are used (first identified by Charles Sanders Pierce and later elaborated upon by Morris [21]).

"A semoitic system can be precisely specified as a system of acts rather than of *things*. Such a specification describes what the system does, not what it is in a physical sense. The specification of acts consists of two basic parts:

"(a) Potential acts. Individually, these may be thought of as mechanical analogues of habits. Collectively, they constitute a body of knowledge delimiting what the system can do, what is within its competence.

"(b) Actual acts. These are individually the analogues of behaviors realizing the potential acts or habits. They relate to one another within a single taxonomic structure that centers on a history of the success or failure of elementary senso-motor behaviors, or inferred projections of that history. Together and in their relations, these actual acts constitute a pattern of experience delimiting what the system observes in the present, remembers of the past, or anticipates for the future.

"Among the potential acts, there must be certain acts which determine how the current pattern of experience is to be "deduced" from the current body of knowledge. The very realization of habits as behaviors depends upon this logical part of the specification. Deductive behaviors realizing these logical habits themselves, may appear in the experimental pattern being constructed; usually the system will not be aware of its logical behaviors.[19]

An automatic classification system was defined and constructed[11] that provided the mechanism for the unifying taxonomic structure. It was demonstrated to be capable of assessing probability of class membership for an occurrence. It also was demonstrated to be capable of detecting the need and carrying out effort to reclassify the data base upon the occurrence of a "misfit."

This second type of approximation provides a mechanism for man and machine to interactively teach each other what is relevant in their dialogue. It also provides a capability for both partners to learn useful lessons for the problem solving activity based on their actual history of success and failure. This latter point is particularly relevant for the situation when another user wishes to do problem solving in an area in which the system has already had some "experience."

In summary, the interactive access mechanisms for problem solving ought to have the following performance

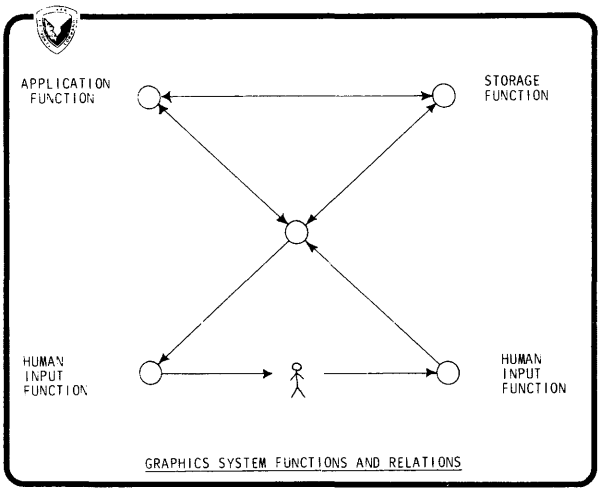


Figure 1

characteristics. The mechanisms should be oriented to discourse on the subject of the problem. Recourse to subsidiary dialogues, e.g. tutorial inquiry, etc., at the request of the human, should be provided to facilitate the operation of the mechanism by the user. The mechanisms should bear the burden of trying to deduce the system implications of a human's service request, rather than the human needing to direct or set up the implementation of service in response to the request. Use of the wide bandwidth channel provided by interactive graphics for man-machine communication at a rate comfortable to the human is the concluding feature of this characterization.

Interactive graphics systems functions and relations

Figure 1 illustrates the relations that exist among the five essential functions in any interactive graphics system. The human output function presents images in a form compatible with human viewing and cognition. The human input function mechanizes requests for attention and provides a means for entry of position, notation or transition data that will affect the graphics and other processes. The storage function retains images or their coded abstractions for subsequent processing. The application function is the set of higher order, "black box" processes that will utilize information inherent in images as input data. Finally, the graphics function performs the three types of sub-processes that are the heart of the graphics process. One type of sub-process provides for composition, construction or formation of images. The second type of sub-process provides for manipulation or transformation of images. The third type of sub-process (attention handling) links the composition and/or manipulation sub-processes to interaction, intervention or use by higher order processes being performed either by the application function or by the user. Notice that the *relations between the graphics system functions are ones of data flow* within an overall system.

We observe the following. The data that flows from one function of the system to another can always be envisioned as having at least two distinct types of components. One component type contains information about variables and their values. The other component type contains information about the identity and parameters of the function that is to utilize the variable data.

Considered this way, inter-function data are messages between elements of the graphics system. They convey service requests for specified functions. Message structured service requests of a limited kind for computer graphics has been considered in the environment of distributed resource, network-oriented systems.[20]

In order to successfully support interactive graphics access in a network environment, careful distribution of the graphics system functions must be accomplished within the network facilities. And equally important, the relationships between graphics system functions must be preserved.

DESIGN TERMINAL

One approach for network-oriented, interactive graphics is illustrated by a Design Terminal configuration[13] under development at the author's installation. Some limited experience with it in conjunction with network access has been gained.[22] The Design Terminal is basically a multi-console terminal with the ability to independently and concurrently interconnect graphics consoles, graphics functions and network-based application and storage functions.

Objectives

Two issues have motivated the development of the Design Terminal. First, in certain types of installations, there is considered to be a need to insure that interactive facilities for problem solving and design specification are functionally matched and economically operated with a multi-source capability from hardware suppliers. This issue involves concern for the relation between types of

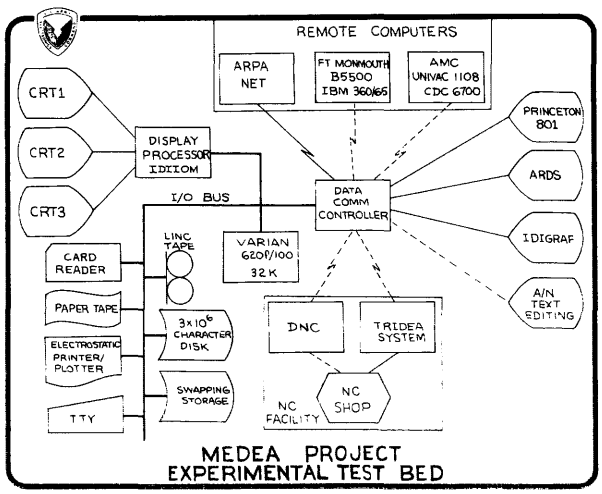


Figure 2

display mechanisms (e.g. refresh CRT's, DVST's, printer/plotters, etc.), the types of graphics' use and the probability of occurrence and volume need for each type of use. Second, for an installation that requires many intelligent terminals, there is the concern for the total system implementation and support costs.

The solution that is being pursued is a little farther around the "wheel of reincarnation"[23] than other related configurations. A general purpose mini-processor and a special purpose display processor form the heart of a terminal with remote access to many shared computers. The general purpose mini-processor is being multi-programmed in a certain way to support concurrent, independent graphics activities emanating from the terminal. The main thrust is to expand the number of concurrently active graphics consoles at the terminal so as to achieve a satisfactory distribution of the total cost of the terminal over each concurrently active console. Figure 2 illustrates the test bed on which this effort is being conducted.

The Design Terminal configuration is concerned with providing an interactive graphics terminal with the following capabilities: (a) in its simplest form, it is a single-console intelligent terminal; (b) the local mini-processor and special purpose processor facilities for providing the graphics function are shared by many interactive graphics consoles; (c) the graphics consoles currently active may employ widely different display mechanisms; (d) a majority of all the connected consoles can be concurrently active; (e) the current use of each active console can involve display of different images than those being generated for all other active consoles at the terminal; (f) the terminal can concurrently obtain support for the graphics system application and storage functions from more than one shared main computer system; and (g) the current use of each active console can involve a different application on a different shared main computer than is involved for all other active consoles at the terminal. The distribution of graphics system functions for the Design Terminal configuration are illuustrated in Figure 3.

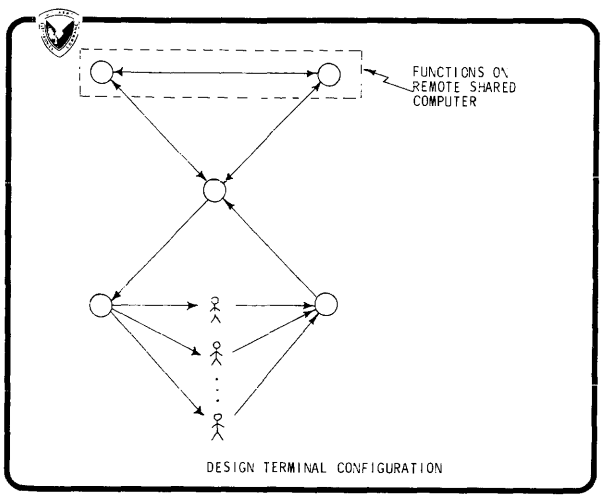


Figure 3

Experience

Although the development of the Design Terminal is still incomplete, our experience so far has provided insight into the difficulties of problem solving on network-based, virtual systems through interactive graphics.

The first point is not new. It is yet another confirmation of something well known to the computing profession and industry. Most users of computing, particularly in an interactive environment, cannot afford to be bogged down in the mechanics of the computer system. They certainly don't care about the subtle intricacies or enduring truths and beauties of the system that turn on its builders and masters. Therefore, the intricate knowledge about access to and use of distributed resources must somehow be built-in to the system.

The second point is also not completely unknown. The telecommunications people have been considering alternatives for a long time. The efforts of Hambrock, et al.[24] and Baron[25] are two of the most significant to our current situation. In a large, dynamic and expanding network, one cannot maintain deterministic directories of every possible combination of resources and associated interconnection schemes that are being used or can be expected to be used. The transmission facilities would be jammed with up-date traffic and the resource processors would be heavily burdened with directory maintenance.

For the user and the system builder, this point appears to raise a paradox. The user doesn't want to and can't manage to know everything about the network. And the maintenance of directories within the network would impose a severe utilization of scarce resources.

The approach of the ARPANET represents an interim compromise for this problem, based upon Baran's work on distributed communication systems. However, the user is still required to know a great deal about the use of each resource in the network even though the communications problem is taken care of for him. For each resource of interest; (a) he must know that the resource exists; (b) he must know where within the net it is located; and (c) he must know the usage procedure required by the processor of that resource. He may be required to know much more. For users of interactive mechanisms with extreme accessibility provided for by the Design Terminal type configuration, this approach to locating and specifying essential resources is especially troublesome.

The conclusion we draw toward the problem we pose is that the resource directory function cannot be built into either the resource processors or the interconnection facilities of the network. We also conclude that attempts to moderate search traffic loading in random techniques[24] and relieve switching bottlenecks can be successful[6] provided the search criteria and routing mechanism are carefully defined.

There is one more point to be considered. It is raised by the cost of computer software development and the growing diversity of available computing resources. We cannot afford and shall not be able to afford explicit re-design of resource linkages each time a new, useful combination is

devised that provides additional higher level capability. The interactive access mechanims to network-based virtual systems must provide or be able to call upon a generalized, dynamic linking function. This function will link together distributed resource modules it has determined to be available and to form the appropriate basis for satisfying an interactive service request.

HIERARCHICAL ACCESS TO VIRTUAL SYSTEMS VIA INTERACTIVE GRAPHICS

Cherry[27] has observed "that recognition, interpreted as response according to habits, depends upon the past experience from which an individual acquires his particular habits." Although his interest was human communication, one should recall the earlier discussion on basic concepts. In particular, consider Cherry's observation in the context of adaptive, use-directed specification and the extensions and amplification of man's basic abilities as a result of problem solving through interactive computing mechanisms. In this context, this observation provides a significant suggestion toward possible answers to many of the difficulties cited above.

Semiotic coupler function

In the linguistic environment, Pendergraft[11] characterized a self-regulating system with goal-directed behavior in terms useful to our purpose. A hierarchy of processes was described. The perception process tries to recognize input data in terms of familiar attributes. The symbolization process assigns identifiers to the recognized data. The valuation process associates the assigned symbols to processes that effect the response of the system to input data. It does so in terms of the system's current knowledge of pragmatic values that will satisfy its goal directed performance.

For problem solving through interactive graphics access to distributed resource networks, the goal would be for the system to correctly determine and cause the interconnection of a virtual system necessary to satisfy each interactive service request. Correctness would be a probablistic measure that would improve with experience for a given problem solving area.

The input data to the perception process would be the image data and/or symbol string that specifies the interactive service request. The output of the perception process would be the syntactic parsing of the service request over the language of service requests. The perception process also operates on a probablistic basis derived from experience.

The input data to the symbolization process would be the identification of processing functions that are required to satisfy the service request. The output of the symbolization process would be the identification of the network's known distributed resources that must be assembled into a virtual system to carry out the processing functions. Again, the performance of the symbolization process will improve as experience increases with both the problem solving topic and the network resources. In situations where processing functions are specified for which network resources are unknown or unavailable, two options exist. Either the symbolization process approximates the function in terms of known resources or the network is searched.

The input data to the valuation process would be the identification of resource modules that will be called upon to satisfy the service request. The output of the valuation process would be the identification of the processing sequence and data flow relationships that must exist amongst the activated resource modules. This valuation process is extremely dependent on experience for improved performance in a given problem solving area.

Adaptive classifier function

Each of the preceding processes depends upon feedback from experience to improve performance. Pendergraft suggests that the processes applied to stimulus information should also be applied to response information.[11]

For us, this suggests that the user should interactively "grade" the system's performance. The system would then apply the three preceding processes to the "grading" information in order to adjust the probability assignments to the estimates of relationship and to the classification structure for service requests vs. processing functions. When "misfits" were identified and/or when the probabilities computed for relationships dropped below some threshold, the classification structure would be recomputed. Pendergraft and Dale[28] originally demonstrated the feasibility of this approach in the linguistic environment using a technique based on Needham's[29] theory of clumps.

As new resources are added to the network, the processing functions that they provide are entered into the classification structure with some initial (perhaps standard) probability assignment for relations to all known types of service requests. These probabilities are then revised based upon the feedback from the user's grading of the system's performance.

Use-directed specification function

The user is of pivotal importance toward specifying service requests and generating performance grades for the system. Yet, as was indicated earlier, he must be capable of the actions without being required to have elaborate knowledge of the system's internal content, structure or mechanisms. It is in this area that interactive graphics plays a vital role in expediting the problem solving dialogue between man and machine.

Consider the simple concept of a "menu," that is, a set of alternatives displayed to the user for his selection. In a complex problem solving area, the result of a user selection from one menu can lead to the display of a subordi

nate menu for further specification of the task to be performed. In effect, this process leads to a concept of the dialogue as a selection process of the alternative paths in trees of menus.

We claim that generalized sub-trees can be devised for areas of problem solving methodology that can be parametrically instantiated to a given topic at run-time only guided by previous menu selections during the current session at the terminal. Furthermore, we claim that this sub-tree concept can be devised so as to allow a given sub-tree to be invoked from a variety of parent nodes in the specification process. Work on the Design Terminal includes an effort to implement this concept.

The specification process cannot be fully accommodated by the mechanism of the parametric dialogue tree. Procedures illustrated by the MATHLAB techniques, the methods of Coons,[30] the Space Form[31] system and more conventional interactive graphics layout, drafting, and curve plotting techniques will all be required in addition to alphanumeric data entry in order to complete the specification. The point is that the semantics of these specifications, in terms of the problem solving processing functions that are required, will have been directed by the current use of the dialogue mechanism.

One set of choices that is always displayed or selectable represents the user's evaluation alternatives of the system's performance. Another optional set is one that places the system in the role of a tutor, either for use of the system or for the use of a processing function to which the system provides access.

Another set of options should also be callable. In this case, the user may want to access a specific processing function. He may not know its name or the location of the resources in the distributed system that support it. If he does have specific identification, he may use it. If he lacks an identifier, the user will generally know of some attribute of the process. Therefore, he should be able to enter a search mode in which he can construct the search criteria for the known attribute in whatever terms that the system supports.

The set of use-directed specifications that are achieved in the above manner form the set of interactive service requests that are input to the semiotic coupler function. The selection of evaluation alternatives forms the feedback input to the adaptive classifier function.

A RECOGNITION AUTOMATON (RECOGNATON)

We suggest that distributed resource computing networks should contain nodes of at least two distinct types. The first type is a service node at which the computing resources of the network are connected. The second type is the access node. It is to the access node that the user is connected through his interactive graphics console.

We further suggest that the access node is the point in the network which implements the functions necessary to provide interactive hierarchical access to virtual systems in the network. We call the implementation vehicle at the access node a recognition automaton or Recognaton.

The Recognaton performs four principal functions. Three of them have already been described: semiotic coupling; adaptive classification; and use-directed specification. The fourth function generates messages into the distributed resource network. It uses the output data of the valuation process of the semiotic coupling function. These messages request assignment and activation of network resources according to the processing sequence and inter-process data flow requirements that were determined from the current status of the pragmatic value system. The messages are the immediate cause for the formation of the virtual system.

The functions of the Recognaton represent a significant computational load at a rather sophisticated level. It is unlikely that the cost to implement this computation could be afforded for a single interactive graphics console. Therefore, we conclude that multiple interactive graphics consoles must be serviced by a given Recognaton. Figure 4 illustrates an access node with the Recognaton functions based on the configuration of the Design Terminal that was discussed earlier.

SUMMARY

This discussion began with a concern for an interactive mechanism to facilitate man-machine dialogues oriented to man's creative thought processes. The intent was to consider problem solving as a test-case, creative process with practical implications. The activity of the mechanism was taken to be to serve as a means for specification of and hierarchical access to virtual systems formed in a distributed resource computing network. A recognition automaton, the Recognaton, has been proposed which appears to serve the purpose and does not impose system level conflicts. Implementation of the Recognaton appears feasible as an extension of the Design Terminal multi-console, interactive graphics configuration.

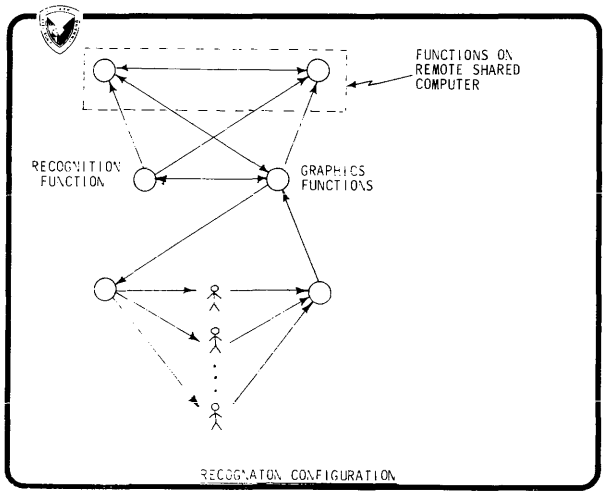


Figure 4

REFERENCES

1. Alexander, C., *Notes on the Synthesis of Form*, Harvard University Press, 1964.
2. Cave, W. C., Dunn, R. M., *Saturation Processing: An Optimized Approach to a Modular Computing System*, ECOM Technical Report - 2636, US Army Electronics Command, November 1965.
3. Dunn, R. M., Gallagher, J. C., Hadden, D. R., Jr. *Distributed Executive Control in a Class of Modular Multi-processor Computing Systems via a Priority Memory*, ECOM Technical Report - 2663, U.S. Army Electronics Command, January 1966.
4. Dunn, R. M., *Modular Organization for Nodes in an Adaptive, Homeostatic Process-Oriented Communications System*, ECOM Technical Report - 2722, US Army Electronics Command, June 1966.
5. Dunn, R. M., "Homeostatic Organizations for Adaptive Parallel Processing Systems," *Proceedings of the 1967 Army Numerical Analysis Conference*, ARO-D Report 67-3, pp. 99-110.
6. Dunn, R. M., *Threshold Functions as Decision Criteria in Saturation Signalling*, ECOM Technical Report - 2961, US Army Electronics Command, April, 1968.
7. Roberts, L. G., Wessler, B. D., "Computing Network Development to Achieve Resource Sharing," *Proceedings of the Spring Joint Computer Conference*, 1970, pp. 543.
8. Walden, D. C., "A System for Interprocess Communications in a Resource Sharing Computer Network," *Communications of the ACM*, Vol. 15, No. 4, April 1972, pp. 221-230.
9. Mink, A., "A Possible Distributed Computing Network," *Proceedings of the Computer Science Conference*, Columbus, Ohio, February 1973.
10. Lay, M., Mills, D., Zelkowitz, M., "A Distributed Operating System for A Virtual Segment-Network," *Proceedings of the AIAA Conference*, Huntsville, Alabama, April 1973.
11. Pendergraft, E. D., *Linguistic Information Processing and Dynamic Adaptive Data Base Management Studies*, Final Report, Linguistic Research Center/University of Texas, Austin, November 1968, chap. 3.
12. Slagle, J. R., *Artificial Intelligence: The Heuristic Programming Approach*, McGraw Hill Book Company, 1971.
13. Dunn, R. M., "Computer Graphics: Capabilities, Costs and Usefulness," *Proceedings of SHARE XL*, Denver, March 1973.
14. Boehm, B. W., Seven, M. J., Watson, R. A., "Interactive Problem Solving - An Experimental Study of 'lock-out' Effects," *Proceedings of SJCC*, 1971, pp. 205-216.
15. Dunn, R. M., "Interactive Graphics: Stand-Alone, Terminal, Or?," *Proceedings of IEEE INTERCON*, March 1973.
16. Raphael, B., *SIR: A Computer Program for Semantic Information Retrieval*, Doctoral Dissertation, M.I.T., June 1964.
17. *MATHLAB*, Project MAC, M.I.T., Cambridge, Mass.
18. Papert, S., "The Computer as Supertoy," *Spring Joint Computer Conference*, 1972.
19. Pendergraft, E. D., *Specialization of a Semiotic Theory*, ECOM Technical Report - 0559-1, Contract No. DAAB07-67-C-0559, TRACOR, INC., Austin, Texas, February 1968.
20. Cotton, I. *Level O Graphic Input Protocol*, ARPANET Network Graphics Working Group, NIC #9929, May 1972.
21. Morris, C., *Signs Language and Behavior*, Prentice-Hall, Inc., 1946.
22. Gray, B. H., "An Interactive Graphics Design Terminal Concept, Implementation, and Application to Structural Analysis," *Proceedings of Fourth Annual Pittsburgh Modeling and Simulation Conference*, April 1973.
23. Myer, T. H., Sutherland, I. E., "On the Design of Display Processors," *Communications of ACM*, Vol. 11, No. 6, June 1968, pp. 410-414.
24. Hambrock, H., Svala, G. G., Prince, L. J., "Saturation Signaling - Toward an Optimum Alternate Routing," *9th National Communication Symposium*, Utica, N.Y., October 1963.
25. Baran, Paul, *On Distributed Communications*, Vols. I-X, Memorandum RM-3420-PR, the Rand Corporation, Santa Monica, August, 1964.
26. Nehl, W., "Adaptive Routing in Military Communications Systems," *Proceeding of the Military Electronics Conference*, 1965.
27. Cherry, C., *On Human Communications*, (Sec. Ed), The M.I.T. Press, 1966, pp. 275.
28. Pendergraft, E., Dale, N., *Automatic Linguistic Classification*, Linguistics Research Center, University of Texas, Austin, 1965.
29. Needham, R., *The Theory of Clumps II*, Cambridge Language Research Unit, Cambridge, England, 1961.
30. Coons, S. A., *Surfaces for Computer-Aided Design of Spaceforms*, Project MAC Technical Report MAC-TR-41, M.I.T., Cambridge, June, 1967.
31. Carr, S. C., *Geometric Modeling*, RADC Technical Report - TR-69-248, University of Utah, Salt Lake City, June 1969.

Performance determination—The selection of tools, if any

by THOMAS E. BELL

The Rand Corporation
Santa Monica, California

INTRODUCTION

As interest in computer performance analysis has grown, the dedication of some analysts to a single tool and analysis approach appears to have become stronger. In most instances this affection probably comes from increased familiarity and success with an approach combined with a resulting lack of familiarity and success with other approaches.

Other equally experienced analysts use a variety of approaches and tools, and may give the appearance that any tool can be used in any situation. Only a little experience is necessary, however, to conclude that personal inspection, accounting data, hardware monitors, software monitors, benchmarks, simulation models, and analytical models are not equally cost effective for performing general operations control, generating hypotheses for performance improvement, testing performance improvement hypotheses, changing equipment, sizing future systems, and designing hardware and/or software systems. The analyst new to performance analysis may become confused, discouraged, and, eventually disinterested in the field as he attempts to start an effective effort. This paper attempts to aid him by presenting an overview. Tools are described; applications are listed; and important considerations are reviewed for selecting a tool for a specific application.

ALTERNATIVE TOOLS AND APPROACHES

An analyst's approach to analyzing a system's performance can, to a large extent, be described by the tools he uses. For example, an analyst using simulation as his tool performs an analysis based on abstracting important characteristics, representing them correctly in a simulation, checking the results of his abstractions, and performing simulation experiments.* If he were analyzing accounting data, his procedure would probably involve listing conditioned data, generating tentative reports, trying to employ them, and then revising the reports. The following summaries indicate some of the important analysis characteristics of simulation, accounting systems, and other available tools. Most of the technical details are omitted because they are only marginally relevant to this discussion.**

Personal inspection

Personal inspection can imply an uninspired glance at the machine room. This sort of activity often leads to beliefs about an installation based more on preconceived notions than on reality. This "tool" usually is employed in an "analysis" involving occasional glances at a machine room when the observer sees precisely what he expected to see (whether it's true or not, and often even in the face of significant, contrary evidence). Since the observer may only glance at the machine room for a few minutes two or three times per day, his sample of the day's operation is very incomplete. This type of performance analysis, although common, is without redeeming social value, and will not be considered further. Other types of personal inspection are more valuable for performance analysis.

Each time a piece of unit record equipment processes a record, it emits a sound. The performance analyst can use this sound to roughly estimate activity and judge the occurrence of certain system-wide problems. For example, a multiprogrammed system may be experiencing severe disk contention in attempting to print spooled records. Quite often, this problem manifests itself in strongly synchronized printing from the several printers on a large system. As the disk head moves from track to track, first one then another printer operates. When one printer completes output for its job, the other printer(s) begins operating at a sharply increased rate.

* More complete procedural suggestions for pursuing a simulation analysis can be found in Reference 1.

** See References 2 and 3 for some details. Further material can be found in References 4–7. A review of monitors available in 1970 appears in Reference 8.

Multiple, rapidly spinning tapes and extremely active disk heads can, in some environments, indicate severe trouble. In other environments (where loads should be causing this kind of behavior), they may indicate a smoothly running system. Unfortunately, most installations fall somewhere between these two extremes, leaving analysts—and managers—with an amorphous feeling of unease.

The clues from personal inspection can be valuable, but an experienced eye, accompanied with an equally experienced ear, is often necessary to make sense from the raw environment. Fortunately, alternatives are available.

Accounting systems

Accounting systems aggregate computer usage by task, job, or other unit of user-directed work. The primary objective of the accounting system's designer is cost allocation, which sometimes compromises the usefulness of accounting system data, particularly where overhead is involved.*

Although accounting data can be deceptive, analysts can determine the actual data collection methods used and perform analyses based on a good understanding of potential errors.** Accounting data also have some distinct advantages for analyses. They are usually quite complete because they are retained for historical purposes, and changes in collection methods are well documented so that users can examine them for correctness. The data are collected about the system's work and organized in precisely the correct way to facilitate workload control—by requests for computer work (by job). In addition to serving as input for reports about computer component usage, accounting data (sometimes combined with operations logs) can be used to determine the use to which this activity was devoted. For example, a user would seldom be using a simulation language if he were involved in writing and running payroll programs, and simulation execution could, prima facie, be considered of negligible value to the organization in this circumstance.

For most analysts, accounting data have the advantage of immediate availability, so analysis can begin without delays for acquisition of a tool. However, immediate data availability does not necessarily imply immediate usability. Accounting systems are commonly very extensive, so analysts are often overwhelmed with the quantity of items collected and the number of incidences of each item. All these data are usually placed in poorly formatted records on a file along with irrelevant or redundant data. The data conditioning problem may therefore be a major hurdle for successful analysis. Inadequate documentation of the details of data collection by manufacturers and inadequacies in the data collection (leading to variability in addition to significant bias) can confuse any analysis results unless the analyst is very careful.

Monitors

Performance monitors (whether implemented in hardware or software) are designed to produce data revealing the achieved performance of the system. These tools produce data, not understanding, so the analyst does not buy his way out of the need for thoughtful analysis when he purchases one.

A hardware monitor obtains signals from a computer system under study through high-impedance probes attached directly to the computer's circuitry. The signals can usually be passed through logic patchboards to do logical ANDs, ORs, and so on, enabling the analyst to obtain signals when certain arbitrary, complex relationships exist. The signals are then fed to counters or timers. For example, an analyst with a hardware monitor could determine (1) the portion of CPU time spent performing supervisory functions while only one channel is active, or (2) the number of times a channel becomes active during a certain period. Because hardware monitors can sense nearly any binary signal (within reason), they can be used with a variety of operating systems, and even with machines built by different manufacturers. This capability to monitor any type of computer is usually not a critically important characteristic, because the analyst is usually concerned with only one family of computers. Some hardware monitors are discussed in References 2-5 and 10-14.

The hardware monitor's primary disadvantage for analysis is its great flexibility. Analysts with extensive experience have learned the most important performance possibilities to investigate, but even the notes distributed by vendors of these monitors often prove inadequate for aiding the novice. Cases of wasted monitoring sessions and of monitors sitting in back rooms are seldom documented, but their validity is unquestionable. In some cases even hardware monitor vendors, while attempting to introduce clients to their monitors, have held a session on an unfamiliar machine and failed miserably. (In most cases, they have proved how valuable it is to have their expertise on hand to aid in performing a complex analysis with a minimum of fuss and bother.)

Software monitors consist of code residing in the memory of the computer being monitored. This means that they can have access to the tables that operating systems maintain, and thereby collect data that are more familiar to the typical performance analyst. Since he usually was a programmer before he became an analyst, descriptions of data collection are often more meaningful to him than the descriptions of hardware monitor data collection points. In addition, most software monitors are designed to produce specific reports that the designers found to be particularly meaningful for the hardware/software combination being monitored. This reduces the

* Determining system overhead is not trivial, and one of the least trivial problems is defining precisely what the term means. The appendix suggests some of its constituents.

** See Reference 9 for some useful techniques to employ accounting data.

difficulty of analysis, particularly where the design of application jobs is under consideration. Hardware monitors, in systems where a program may reside in a variety of places, do not typically produce reports on individual problem performance that can be easily interpreted, but software monitors typically can. For more material on some software monitors see References 2, 3, 7, and 16-20.

The answer to every analyst's problem is not a software monitor. Software monitors require a non-negligible amount of memory, often both central memory and rotating memory. In addition, some amount of I/O and CPU resources are necessary for operating the monitor. This all amounts to a degradation in system performance, and at the precise time when people are concerned with performance. As a result, the analyst needs to choose carefully how much data he will collect and over how long a period. This necessity adds to the analyst's problems, and is usually resolved in favor of short runs. This, in turn, leads to data of questionable representativeness. Since computer system loads usually change radically from hour to hour, the analyst may be led to conclude that one of his changes has resulted in significant changes in performance, when the change actually resulted from different loads over the short periods of monitoring.

The choice between hardware and software monitors (and between the subtypes of monitors in each group—sampling vs full time monitoring, separable vs integrated, recording vs concurrent data reduction, and so on) is largely dependent on situation-specific characteristics. Application in the specific situation usually involves monitoring the normal environment of the existing computer. An alternative exists: a controlled, or semi-controlled, environment can be created. This analysis approach is closely related to the use of batch benchmarks and artificial stimulation.

Benchmarks

A batch benchmark consists of a job, or series of jobs, that are run to establish a "benchmark" of the system performance. The benchmark run is usually assumed to be typical of the normal environment but to have the advantage of requiring a short time for execution. The most common use of benchmarks is for equipment selection, but analysts often use benchmarks for determining whether a change to their systems has improved the performance of the benchmark job stream. The conclusion about this performance (usually measured primarily by the elapsed time for execution) is then assumed to be directly related to the normal environment; an improvement of 20 percent in the benchmark's performance is assumed to presage an improvement of 20 percent in the real job stream's performance. Benchmark work is described in References 21 and 22.

For an on-line system this technique would not be applicable because on-line jobs exist as loads on terminals rather than as code submitted by programmers. The analog to the benchmark job in a batch system is artificial stimulation in the on-line environment. Through either hardware or software techniques, the computer system is made to respond to pseudo-inputs and the response is measured. A stimulator, implemented in software, is described in Reference 23.

The obvious difficulty in using batch or on-line benchmarking is relating the results to the real job stream. The temptation is to assume that jobs presented by users are "typical" and that the results will therefore be applicable to reality, or that the on-line work described by the users is actually what they do. Neither assumption is generally true.

Running benchmarks or artificially stimulating a system implies some kind of measurement during a period of disrupting the system's operation; then the results must be related to reality. Performance modeling has the same difficulty in relating its results to reality, but it does not disrupt the system's operation.

Performance modeling

Simulation modeling of computer system performance has seemed an attractive technique to analysts for years, and it has been used in response to this feeling. An analyst may design his own simulation using one of the general or special purpose languages, or employ one of the packaged simulators on the market.* In either case, he can investigate a variety of alternative system configurations without disrupting the real system, and then examine the results of the simulated operation in great detail. Virtually all such simulations model the operation of the system through time, so time-related interactions can be thoroughly investigated. Some simulation experiences are described in References 29-35. Problems and objectives in simulating computers are described in Reference 36.

Analytical models are usually steady-state oriented, and therefore preclude time-related analysis. However, they usually do provide mean and variance statistics for analyses, so those analyses requiring steady-state solutions (e.g., most equipment selections) could employ the results of analytical modeling. Simulations, on the other hand, must be run for extensive periods to determine the same statistics, and analysts need to worry about problems like the degree to which an answer depends on a stream of random numbers. Examples of analytical modeling are given in References 37-42.

The problem that often proves overwhelming in using either type of modeling is ensuring that the model (the abstraction from reality) includes the most important performance-determining characteristics and interactions of the real system. Without this assurance, the model is usually without value. Unfortunately for the analyst, indication that a particular model was correct for another installation is no real assurance that it is correct for his installation. Unique performance determinants are

* For information about such languages and packages see References 24-28.

usually found in operating system options, configuration details, and workload characteristics. Therefore, a validation exercise of a simulative or analytical model is usually a necessity if specific values of output parameters are to be used in an analysis.

APPLICATIONS

The applications of computer performance analysis can be categorized in a variety of ways, depending on the objective of the categorization. In this paper the objective is to aid in selecting an appropriate tool and analysis approach. The following categorization will therefore be adopted:

General control: Many installations are run by intuition. Freely translated, this means that they are not managed, but instead allowed to run without control. All attempts at other applications of performance analysis will be of marginal utility without control based on adequate operating information.

Hypothesis generation: Computer system performance improvement involves generating hypotheses, testing hypotheses, implementing appropriate changes, and testing the changes.* Useful information for hypothesis generation often appears so difficult to specify and obtain that random changes are attempted to improve the system. The failure rate for performance improvement efforts without explicit hypothesis generation is extremely high.

Hypothesis testing: Given an interesting hypothesis, an analyst's first impulse is to assume its correctness and begin changing the system. This usually results in lots of changes and little improvement. Hypothesis testing is imperative for consistently successful computer system performance improvement.

Equipment change: The friendly vendor salesman says his new super-belchfire system will solve all your problems. The change is too large to be classified as a performance improvement change. Should you take his word for it and make him rich, or do your own analysis? If you choose to do your own analysis, you're in this category when you're upgrading or downgrading your system.

Sizing: Sizing a new system is a step more difficult than equipment change because it often involves estimating workload and capacity in areas where extrapolation of existing characteristics is impossible or unreliable. This situation does not occur so often as equipment change, but usually involves much higher costs of incorrect decisions. Typical situations are bringing in a new computer for a conceptualized (but unrealized) workload, centralization of diverse workloads previously run on special purpose hardware/software systems, and decentralization of workload from a previously large system to a series of smaller ones. Vendor selection is included in this category since the performance-related part of this problem can be described as sizing and verifying (or merely verifying, in the case of some procurements) the performance of a certain size system.

System design: Whether dealing with hardware or software, designers today usually are concerned with performance. If the designers are in the application area, the concern for performance often comes too late for doing much about the mess. Early consideration, however, can be expensive and unfruitful if carried on without the proper approach.

The easiest situation would be for each of these categories to have exactly one tool appropriate for application in analyses, but performance analysis has more dimensions than the single one of analysis objective. Two of the most important ones are analyst experience and type of system under consideration.

ANALYST EXPERIENCE

Some groups of analysts have considered single systems (or a single model of system processing essentially the same load at several sites) over a period of years. These groups have often developed simulation and analytical tools for one type of analysis, and then, with the tool developed and preliminary analysis already performed, apply them in other situations. Similarly, they may have used accounting data for a situation where it is particularly applicable, and then have applied it in an analysis in which accounting data's applicability is not obvious. The ability of group members to apply a variety of familiar tools freely in diverse situations is one of the reasons for maintaining such groups.

Some other groups have developed analysis techniques using a single tool to the extent that their members can apply it to a much wider variety of situations than expected because they have become particularly familiar with its characteristics and the behavior of the systems they are analyzing. As a result, such groups have proved able to enter strange installations and produce valuable results by immediately executing rather stylized analyses to check for the presence of certain hypothesized problems.

The analyst with less than one or two years of performance analysis experience, however, cannot expect to achieve the same results with these approaches. The remainder of this paper will consider the situation of the more typical analyst who is not yet extensively experienced.

* This is a summary of the steps suggested in Reference 43.

TYPE OF SYSTEM

Software monitors are obviously commercially available for IBM System 360 and 370 computers, but their existence for other systems is often unrecognized. This sometimes leads analysts to believe that personal inspection is the only alternative for any other system. In fact, virtually every major computer system on the market currently possesses an accounting system; hardware monitors will work on any system (with the exception of certain very high speed circuits); batch benchmarks can be run on any system; and models can be constructed for any system (and have been for most). In addition, software monitors have been implemented for most computer systems in the course of government-sponsored research. The analyst's problem is to discover any required, obscure tools and to be able to use them without undue emphasis on learning the tools' characteristics.

The world of performance analysis tools is not so smooth as may be implied above. First, benchmarks for on-line systems are nearly impossible to obtain for any system without assistance of the computer system vendor. Second, many tools are simply no good; their implementers did a poor job, or they are poorly documented, or they don't do the thing needed for the problem at hand. Third, searching out the appropriate tool may require more time than the analyst can spend on the entire performance analysis. Fourth, the analyst seldom has prior knowledge of whether one of the first three problems will arise, so he doesn't know where to concentrate his search. Fortunately, any of several different types of tools can be used in most analyses, so the analyst can pick from several possibilities rather than search for some single possibility. The choice is largely dependent on the category of analysis being performed.

CATEGORY OF ANALYSIS

Having presented some important analysis characteristics of various tools, limited the discussion to apply only to analysts without extensive experience, and begged the question of tool availability, the important step remains of matching analysis objective with type of tool and analysis. Suggestions about tool selection for each analysis objective are given below; they should be interpreted in the context of the discussion above.

General control

Accounting data generally have proven most appropriate for general control. They are organized correctly for generating exception reports of system misuse by programmers (incorrect specification of job options, violating resource limitations, and running jobs inappropriate for their assigned tasks). They also usually provide valuable information about operations (number of reloads of the operating system, number of reruns, incidence of the system waiting for tape mounts, etc.). Further, it provides data on the level of chargeable resource utilization so that financial management can be performed.

Accounting data's primary disadvantage is the difficulty of generating meaningful reports from it. It also requires operators' adherence to appropriate standards of operation for maintaining reliable data. Further, it usually can provide reports no sooner than the following day. One alternative is to use a very inexpensive hardware monitor with dynamic output for on-line operational control and to use accounting data for normal reporting. (Regular use of monitors, perhaps one use per month, can also be adopted to supplement the accounting data.)

The most commonly used general control technique is one of the least useful. Personal inspection is inadequate for anything except the case of a manager continually on the floor—and he needs an adequate system of reporting to detect trends that are obscured by day-to-day problems. The techniques in this section may appear too obvious to be important, but we find that ignoring them is one of the most common causes of poor computer system performance.

Hypothesis generation

Hypothesis generation for system performance improvement is based on the free run of imagination over partly structured data, combined with the application of preliminary data analysis techniques. In general, the data leading to the most obvious relationships prove best, so personal inspection and partly reduced accounting data often are most useful. Quick scans of system activity, organized by job, often lead to hypotheses about user activity. An analyst can often hypothesize operational problems by visiting several other installations and trying to explain the differences he observes.

Some installations have found that regular use of a hardware or software monitor can lead to generating hypotheses reliably. The technique is to plot data over time and then attempt to explain all deviations from historical trends. This approach may have the advantage of hypothesis formulation based on the same data collection device that is used for hypothesis testing.

Hypothesis testing

Nearly any tool can be used for testing performance improvement hypotheses. The particular one chosen is usually based on the type of hypothesis and the tool used in generating the hypothesis. For example, hypotheses about internal processing inefficiencies in jobs can usually be best tested with software monitors designed to collect data on application code. Hypotheses about the allocation of resource use among programmers can usually be tested most readily through the use of account-

ing data. Simulative and analytical models can often be used to perform tests about machine scheduling and the trade-off of cost and performance, particularly when hypothesis generation employed modeling.

After a hypothesis is tested, implementation of a change is usually the next step. Following implementation, the resulting performance change requires examination to ensure that the expected change, and only that change, has occurred. Although the same tool may be used for both parts of hypothesis testing, employing a different tool provides the advantage of forcing the analyst to view the system from a slightly different point of view, and therefore reduces the chance of ignoring important clues in seemingly familiar data. This advantage must be traded-off against the advantage of easing detection of perturbations in the familiar data caused by implemented changes.

A special note is in order for the use of benchmarks in hypothesis testing. If a hypothesis involves a characteristic of the basic system, a completely controlled test often can test the hypothesis far more thoroughly than other types of tests. For example, an analyst might hypothesize that his operating system was unable to initiate a high-priority job when an intermediate-priority job had control of the CPU. While he could monitor the normal system until the condition naturally occurred, a simple test with the appropriate benchmark jobs could readily test the hypothesis. We have found that artificial stimulation of on-line systems can similarly test hypotheses rapidly in both controlled tests and monitoring normal operation. The temptation to examine only "the normal system" should be resisted unless it proves to be the most appropriate testing technique.

Equipment change

Equipment change might involve upgrading the system's CPU, changing from slow disks to faster ones, or adding a terminal system. All these changes might be considered merely major tuning changes, but they involve enough financial risk that more analysis is devoted to them than normal system performance improvement efforts. In addition, the analyst has a very stylized type of hypothesis: how much performance change results from the hardware change? These special characteristics of equipment change lead to increased use of benchmarks and simulation.

When the alternative configuration exists at another installation (usually a vendor's facility), analysts can generate a series of benchmarks to determine how well the alternative performs in comparison with the existing system. Recently, synthetic benchmarks have come to be used more extensively in this process, particularly in test designs which examine particular system characteristics, or which include carefully monitoring normal system utilization to improve the meaningfulness of the benchmarks.

In other cases there is no system available for running the benchmarks. Simulation is often employed in this environment. The most important problem in this type of analysis is ensuring the validity of the workload description on the alternative system and the validity of the alternative's processing characteristics. Unvalidated simulations may be the only reasonable alternative, but the risk of employing them is usually high.

Sizing

The technique used most commonly today in sizing computers is listening to vendor representatives and then deciding how much to discount their claims. This situation is partly the result of the difficulties involved in using the alternatives—benchmarking and modeling. Although analytical modeling is conceptually useful, its use in sizing operations has been minimal because its absolute accuracy is suspect. Simulative modeling appears less suspect because the models are closer to commonly-used descriptions of computer systems. The sensitivity of simulation models to changes in parameters can often be verified, at least qualitatively, so analysts can gain some degree of faith in their correctness.

All the problems of using benchmarking in equipment change analyses are present when benchmarking is used in sizing analyses. In addition, the relationship of benchmarks to workloads that will appear on a future system is especially difficult to determine. A synthetic benchmark job might be quite adequate for representing workload meaningfully on a modification of the existing system, but its characteristics might be very wrong on a completely different system. (This same problem may be true for simulations, but a validated simulation should facilitate correct workload descriptions.)

Design

Tool selection in design must be divided into two parts —selection in the early design phase and selection in the implementation phase. In the earlier phase, performance analysis must be based on modeling because, without any implemented system, real data cannot be collected. The later phase might, therefore, seem particularly suited to the data collection approaches. In fact, modeling appears to be a good technique to employ in concert with monitored data in order to compare projections with realized performance. Collecting data without using modeling may decrease management control over development and decrease the ease of data interpretation.

Design efforts can begin by using modeling exclusively, and then integrate monitoring into the collection of tools as their use becomes feasible.

FINAL COMMENT

Computer performance analysis tools and approaches are in a period of rapid development, so the appropriateness of their application in various situations can be expected to change. In addition, individual analysts often

find that an unusual application of tools proves the best match to their particular abilities and problems. The suggestions above should therefore not be interpreted as proclamations of the best way to do performance analysis, but as general indications of potentially useful directions.

Inadequate understanding of computer system performance currently precludes quantifying problems across large numbers of systems. Each analyst must feel his way to a solution for each problem with only helpful hints for guidance. If improved understanding is developed, the artistic procedure discussed in this paper may evolve into a discipline in which analysts have the assurance they are using the correct approach to arrive at the correct answer.

REFERENCES

1. Morris, M. F., "Simulation as a Process," *Simuletter*, Vol. 4, No. 1, October 1972, pp. 10-21.
2. Bell, T. E., *Computer Performance Analysis: Measurement Objectives and Tools*, The Rand Corporation, R-584-NASA/PR, February 1971.
3. Canning, R. G. (ed.), "Savings from Performance Monitoring", *EDP Analyzer*, Vol. 10, No. 9, September 1972.
4. Murphy, R. W., "The System Logic and Usage Recorder," *Proc. AFIPS 1969 Fall Joint Computer Conference*, pp. 219-229.
5. Rock, D. J., Emerson, W. C., "A Hardware Instrumentation Approach to Evaluation of a Large Scale System," *Proc. of The 24th National Conference (ACM)*, 1969, pp. 351-367.
6. Johnston, T. Y., "Hardware Versus Software Monitors," *Proc. of SHARE XXXIV*, Vol. I, March 1970, pp. 523-547.
7. Kolence, K. W., "A Software View of Measurement Tools," *Datamation*, Vol. 17, No. 1, January 1, 1971, pp. 32-38.
8. Hart, L. E., "The User's Guide to Evaluation Products," *Datamation*, Vol. 16, No. 17, December 15, 1970, pp. 32-35.
9. Watson, R. A., *Computer Performance Analysis: Applications of Accounting Data*, The Rand Corporation, R-573-NASA/PR, May 1971.
10. Bell, T. E., *Computer Performance Analysis: Minicomputer-Based Hardware Monitoring*, The Rand Corporation, R-696-PR, June 1972.
11. Estrin, G., Hopkins, D., Coggan, B., Crocker, S. D., "SNUPER Computer: A Computer in Instrumentation Automation," *Proc. AFIPS 1967 Spring Joint Computer Conference*, pp. 645-656. 645-656.
12. Carlson, G., "A User's View of Hardware Performance Monitors," *Proc. IFIP Congress 1971*, Ljubljana, Yugoslavia.
13. Cockrun, J. S., Crockett, E. D., "Interpreting the Results of a Hardware Systems Monitor," *Proc. 1971 Spring Joint Computer Conference*, pp. 23-38.
14. Miller, E. F. Jr., Hughes, D. E., Bardens, J. A., *An Experiment in Hardware Monitoring*, General Research Corporation, RM-1517, July 1971.
15. Stevens, D. F., "On Overcoming High-Priority Paralysis in Multiprogramming Systems: A Case History," *Comm. Of The ACM*, Vol. 11, No. 8, August 1968, pp. 539-541.
16. Bemer, R. W., Ellison, A. L., "Software Instrumentation Systems for Optimum Performance," *Proc. IFIP Congress 1968*, pp. 39-42.
17. Cantrell, H. N., Ellison, A. L., "Multiprogramming System Performance Measurement and Analysis," *Proc. 1968 Spring Joint Computer Conference*, pp. 213-221.
18. *A Guide to Program Improvement with LEAP*, Lambda LEAP Office, Arlington, Virginia (Brochure).
19. *Systems Measurement Software (SMS/360) Problem Program Efficiency (PPE) Product Description*, Boole and Babbage, Inc., Cupertino, California (Brochure).
20. Bookman, P. G., Brotman, G. A., Schmitt, K. L., "Use Measurement Engineering for Better System Performance," *Computer Decisions*, Vol. 4, No. 4, April 1972, pp. 28-32.
21. Hughes, J. H., *The Construction and Use of a Tuned Benchmark for UNIVAC 1108 Performance Evaluation—An interim report*, The Engineering Research Foundation at the Technical University of Norway, Trondheim, Norway, Project No. 140342, June 1971.
22. Ferrari, D., "Workload Characterization and Selection in Computer Performance Measurement," *Computer*, July/August 1972, pp. 18-24.
23. *Load Generator System General Information Manual*, Tesdata Systems Corporation, Chevy Chase, Maryland (Brochure).
24. Cohen, Leo J., "S3, The System and Software Simulator," *Digest of The Second Conference on Applications of Simulation*, New York, December 2-4, 1968, ACM et al., pp. 282-285.
25. Nielsen, Norman R., "ECSS: An Extendable Computer System Simulator," *Proceedings, Third Conference on Applications of Simulation*, Los Angeles, December 8-10, 1969, ACM et al., pp. 114-129.
26. Bairstow, J. N., "A Review of System Evaluation Packages," *Computer Decisions*, Vol. 2, No. 6, July 1970, Pg. 20.
27. Thompson, William C., "The Application of Simulation in Computer System Design and Optimization," *Digest of The Second Conference on Applications of Simulation*, New York, December 2-4, 1968, ACM et al., pp. 286-290.
28. Hutchinson, George K., Maguire, J. N., "Computer Systems Design and Analysis Through Simulation," *Proceedings AFIPS 1965 Fall Joint Computer Conference*, Part 1, pp. 161-167.
29. Bell, T. E., *Modeling the Video Graphics System: Procedure and Model Description*, The Rand Corporation, R-519-PR, December 1970.
30. Downs, H. R., Nielsen, N. R., and Watanabe, E. T., "Simulation of the ILLIAC IV-B6500 Real-Time Computing System," *Proceedings Fourth Conference on Applications of Simulation*, December 9-11, 1970, ACM et al., pp. 207-212.
31. McCredie, John W. and Schlesinger, Steven J., "A Modular Simulation of TSS/360," *Proceedings, Fourth Conference on Applications of Simulation*, New York, December 9-11, 1970, ACM et al., pp. 201-206.
32. Anderson, H. A., "Simulation of the Time-Varying Load on Future Remote-Access Immediate-Response Computer Systems," *Proceedings, Third Conference on Applications of Simulation*, Los Angeles, December 8-10, 1969, ACM et al., pp. 142-164.
33. Frank, A. L., "The Use of Simulation in the Design of Information Systems," *Digest of The Second Conference on Applications of Simulation*, New York, December 2-4, 1968, ACM et al., pp. 87-88.
34. Dumas, Richard K., "The Effects of Program Segmentation on Job Completion Times in a Multiprocessor Computing System," *Digest of The Second Conference on Applications of Simulation*, New York, December 2-4, 1968, ACM et al., pp. 77, 78.
35. Nielsen, Norman R., "An Analysis of Some Time-Sharing Techniques," *Comm. of The ACM*, Vol. 14, No. 2, February 1971, pp. 79-90.
36. Bell, T. E., *Computer Performance Analysis: Objectives and Problems in Simulating Computers*, The Rand Corporation, R-1051-PR, July 1972. (Also in *Proc. 1972 Fall Joint Computer Conference*, pp. 287-297.)
37. Chang, W., "Single-Server Queuing Processes in Computer Systems," *IBM Systems Journal*, Vol. 9, No. 1, 1970, pp. 37-71.
38. Coffman, E. G. Jr., Ryan Jr., Thomas A., "A Study of Storage Partitioning Using a Mathematical Model of Locality," *Comm. of The ACM*, Vol. 15, No. 3, March 1972, pp. 185-190.
39. Abate, J., Dubner, H., and Weinberg, S. B., "Queuing Analysis of the IBM 2314 Disk Storage Facility," *Journal of The ACM*, Vol. 15, No. 4, October 1968, pp. 577-589.
40. Ramamoorthy, C. V., Chandy, K. M., "Optimization of Memory Hierarchies in Multiprogrammed Systems," *Journal of The ACM*, Vol. 17, No. 3, July 1970, pp. 426-445.
41. Kimbleton, S. R., "Core Complement Policies for Memory Allocation and Analysis," *Proc. 1972 Fall Joint Computer Conference*, pp. 1155-1162.

42. DeCegama, A., "A Methodology for Computer Model Building," *Proc. 1972 Fall Joint Computer Conference*, pp. 299-310.
43. Bell, T. E., Boehm, B. W., Watson, R. A., *Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort*, The Rand Corporation, R-549-1-PR, November 1972. (Much of this document appeared as "Framework and Initial Phases for Computer Performance Improvement," *Proc. 1972 Fall Joint Computer Conference*, pp. 1141-1154.)

APPENDIX—COMPUTER OVERHEAD

Accountancy requires that computer overhead costs be borne by users who are charged directly for their demands on the system. Data collection systems tend to include this requirement as a basic assumption underlying their structures. The resulting aggregation obscures the type of overhead most prominent in a system, the resources heavily used by overhead activities, and the portion of total system capability devoted to overhead activities. System analysis requires these data; they need definition and should be available for performance analysis.

From the viewpoint of performance analysis, at least five components of overhead can be identified in most multiprogramming systems. These are:

1. I/O handling
2. User resource request handling
3. System handling of spooled I/O
4. Job or sub-job (e.g., job step or activity) initiation/termination
5. System operation (including task switching, swapping, maintaining system files, etc.)

I/O handling may require large amounts of time, but this is largely controllable by the individual user. Item one, therefore, may not be a candidate for inclusion in a definition of overhead in many analyses.

User resource request handling (at least at the time of job or sub-job initiation) is similarly controllable by the users except for required system-required resources (such as system files). Item two might be included in definitions more often than item one, particularly since item two is often influenced strongly by installation-specified practices (such as setting the number of required files).

System handling of spooled I/O is under the control of users to the extent that they do initial and final I/O, but the alternatives open to installation managements for influencing its efficiency are often very great. For example, changing blocking sizes or using an efficient spooling system (such as HASP) can have gross effects on the amount of resources consumed in the process. Installation management's control over this is so high that item three is often included in a definition of overhead.

Initiation and termination appear to consume far more resources than usually assumed. User-specified options influence the amount of resource usage, but installation-chosen options and installation-written code can impact usage to a large degree. The choice of specific operating system, order of searching files for stored programs, layout of system files, and options in the operating system can change the resources used to such an extent that item four should be included in overhead in nearly all cases.

System operation is always included as a part of overhead. The difficulty of separating this element of overhead from all the rest is very difficult, so analyses usually assume that it is included as part of one of the other elements. One technique for quantifying its magnitude is to decide on the parts of code whose execution represent it and then to measure the size of these elements. The same parts of code can be monitored with a hardware monitor to determine the amount of processor time and I/O requests that arise from execution of the code. The sizes of system files are usually not difficult to obtain for determining the amount of rotating memory used by this type of overhead. This technique, however, will nearly always underestimate that amount of overhead since pieces of overhead are so scattered through the system.

Ideally, each of the types of overhead would be identified and measured so that installations could control the amount of each resource that is lost to it. If the resource loss to overhead were known for typical systems, each of the applications of performance analysis would be eased.

Computing societies—Resource or hobby?

by ANTHONY RALSTON

State University of New York
Buffalo, New York

ABSTRACT

The fodder for a technical society is people but people can nevertheless use as well as be used by the society. Such use can be passive (e.g., publishing articles in the society's journals) or active through direct participation in the professional activities or administration of the society. As in the use of all computing resources, there is a potential for both profit and loss; these will be examined, in part at least, seriously.

The special libraries association today

by E. A. STRABLE

Special Libraries Association
New York, New York

ABSTRACT

Special librarians are part of the larger library community but can be differentiated from other groups of librarians (school, public, academic) by where they practice their profession, by the groups with whom they work, and most importantly, by their goals and objectives. The major objective, the utilization of knowledge for practical ends, brings special librarianship thoroughly into information processing in some unusual and unique ways. The Special Libraries Association is the largest society to which special librarians belong. The Association, like its members, is also involved in a number of activities which impinge directly upon, and affect, the role of information processing in the U.S.

Copyright problems in information processing

by B. H. WEIL

Esso Research and Engineering Company
Linden, New Jersey

ABSTRACT

Present copyright laws were developed largely to protect "authors" against large-scale piracy of books, articles, motion pictures, plays, music, and the like. These laws and related judicial decisions have in recent years raised serious questions as to the legality of such modern information processing as the photocopying, facsimile transmission, microfilming, and computer input and manipulation of copyrighted texts and data. Congress has so far failed to clarify these matters, except for sound recordings. It has proposed to have them studied by a National Commission, but it has repeatedly refused to establish this without also passing revisions chiefly dealing with cable-TV. Emphasis will be placed in this talk on consequences for libraries, library networks, and other information processors, and on recent legislative developments.

Standards for library information processing

by LOGAN C. COWGILL

Water Resources Scientific Information Center
Washington, D.C.

and

DAVID L. WEISBROD

Yale University Library
New Haven, Connecticut

ABSTRACT

Technical standards will be described in terms of their intent, their variety (national, international, etc.), their enumeration, and their development process. Their importance will be evaluated in terms of their present and future usefulness and impact.

A network for computer users

by BRUCE K. ALCORN

Western Institute for Science and Technology
Durham, North Carolina

ABSTRACT

Computer networks are an accepted fact in the world of computing, and have been for some time. Not so well accepted, however, is the definition of a computer network. Some claim that to be a network the communications system must connect a group of computers as opposed to a network of terminals communicating with one computer. Still others hold that both are examples of computer networks; the first being a ring network and the latter a star network.

Within education, computer networks of many descriptions exist. Most such activities have dealt with the institutions of higher education, but there are some notable exceptions. These networks are operated by universities, independent non-profit corporations, branches of state governments, and private industry. Some are time-sharing systems, some operate in the remote batch mode, and others offer both types of service. Most of the computing done through these networks has been for instructional purposes; however, a great many research problems are processed with administrative applications last in amount of activity, although increasing.

During 1968 the National Science Foundation initiated a number of projects which gave a great impetus to computer networks, mainly among colleges and universities. This effort continues today in a different form through the Expanded Research Program Relative to a National Science Computer Network of the NSF.

Currently the National Institute of Education is supporting the development of the Nationwide Educational Computer Service, a network designed to help colleges and school systems meet their computing needs at a minimum of cost. This network will consist of a large scale computer serving a series of intelligent terminals in institutions in various parts of the United States. The system is configured in such a way so as to assist the student, faculty, and administrator at a cost effective rate. The factors involved in producing this saving include the particular hardware and software at the central site and at the terminal location, the mode of operation and the effective use of existing tele-communication facilities.

Uses of the computer in large school districts

by THOMAS J. McCONNELL, JR.

Director, Atlanta Public Schools
Atlanta, Georgia

ABSTRACT

In this age of accountability in education it is apparent that the most economical and efficient systems conceivable must be made available to the administrator. This fact is true at all levels of management from the classroom to the superintendent.

Most large school districts could not perform all of the tasks required of them if they had to operate in a manual mode. This fact is certainly not unique to school districts but is a common problem of our dynamic society.

The administrative use of the computer in most school districts came about as a result of a need for more efficient and faster methods of performing accounting functions. After their first introduction they generally just "growed" as Topsy would say. Most large school districts today will have a rather sophisticated set of hardware and software supported by a very fine staff of professionals.

With the advent of tighter budget control and with most educators today clamoring for some form of "program budgeting" the computer is an even more vital ingredient that is required if we are to provide for quality education.

Additionally, it is no longer sufficient to provide automation to the administrative functions in a school district. The computer is fast becoming an essential part of our instructional program. This instructional role of the computer is coming into being in the form of Computer Managed Instruction (CMI) as well as Computer Assisted Instruction (CAI).

Although development of uses for the computer for instructional purposes has only been under way for a few short years, we have witnessed some very dramatic results. Most educators are in agreement as to the effectiveness of the computer for instructional purposes; the fact that it has not expanded as many had hoped and assumed is a function of finances rather than a shortcoming of the implementation.

Education can expect to have some very rewarding experiences in its relationship with the computer and the computer professional in the seventies. This fact will come about as a result of developments in computer technology both in hardware and in software. Also, the reduction in the cost factor should be of such magnitude that computer services will be available to more school districts and at a cost that they can afford.

With proper organization and cooperation the computer can begin to realize its full potential in bringing about efficient, effective education in its many aspects.

Training of teachers in computer usage

by DUANE E. RICHARDSON

Northwest Regional Educational Laboratory
Portland, Oregon

ABSTRACT

I plan to discuss the need in teacher education for training and experience in the selection of instructional materials for use on computers and the teacher's role in helping to identify criteria for developing additional instructional materials.

Specific discussion will be directed at describing a course which will guide teachers through the development of a set of criteria by which to judge the value of such instructional applications and will demonstrate how the criteria can be applied. The course will allow the teacher to practice application of the criteria to sample instructional uses from his particular interest.

How schools can use consultants

by DONALD R. THOMAS

ARIES Corporation
Minneapolis, Minnesota

ABSTRACT

Data processing consulting firms today offer a variety of professional services to schools. Users of these services, however, often differ in their opinions of the value of these services.

The point of this presentation is simply that unsatisfactory consultant relationships can have their source not only in the consultant himself, but also in the school's use of the consultant's services. In other words, use of consultive services implies a two-way relationship which is subject to misuse and abuse by either party.

The experience throughout the educational computer area demonstrates that time and effort devoted to sound use of consultants will pay substantial dividends. That factor should be a major one in the planned use of a consultant.

An experienced consultant will bring expertise to a study based upon his experiences with other clients. This should result in client confidence and in assuring that the unique needs of the clients will be identified and addressed.

NAPSS-like systems—Problems and prospects

by JOHN R. RICE

Purdue University
West Lafayette, Indiana,

NAPSS-NUMERICAL ANALYSIS PROBLEM SOLVING SYSTEM

This paper arises from the development of NAPSS and discusses the problems solved and still to be solved in this area. The original paper contains two phrases which define the objectives of NAPSS (and NAPSS-like systems) in general, yet reasonably precise, terms:

> *"Our aim is to make the computer behave as if it had some of the knowledge, ability and insight of a professional numerical analyst."*

> *"describe relatively complex problems in a simple mathematical language—including integration, differentiation, summation, matrix operations, algebraic and differential equations, polynomial and other approximations as part of the basic language."*

A pilot system has been completed at Purdue and runs on a CDC 6500 with an Imlac graphics console. It does not contain all the features implied by these objectives, but it has (*a*) shown that such a system is feasible and (*b*) identified the difficult problem areas and provided insight for the design of a successful production system. The purpose of this paper is to identify the eight principal problem areas, discuss four of them very briefly and the other four in somewhat more detail. Several of these problem areas are specific to NAPSS-like systems, but others (including the two most difficult) are shared with a wide variety of software systems of a similar level and nature.

The presentation here does not depend on a detailed knowledge of NAPSS, but specific details are given in the papers listed in the references.[1,2,3,4,5,6,7]

PROBLEM AREAS

The eight principal problem areas are listed below:

1. Language Design and Processing
2. Simulating Human Analysis (Artificial Intelligence)
3. Internal System Organization
4. User Interface
5. Numerical Analysis Polyalgorithms
6. Symbolic Problem Analysis
7. Operating System Interface
8. Portability

The first of these is the best understood and least difficult at the present time. The next four are very substantial problem areas, but the pilot NAPSS system shows that one can obtain acceptable performance and results. Symbolic problem analysis (except for things like symbolic differentation, is not made in the pilot NAPSS system and, in a numerical analysis context, this is an undeveloped area. The interface with the operating system is very complex in the pilot system and is an area of unsolved problems. Basically the pilot NAPSS system needs more resources than the operating system (which is indifferent to NAPSS) provides for one user. The final problem area, portability is as difficult for NAPSS as for other complex software systems.

All of these problem areas except 5 and 6 are present in any problem solving system with a level of performance and scope similar to NAPSS. Examples include statistical systems (BMD,SPSS,OSIRIS); linear programming and optimization packages (LP90,OPTIMA); engineering problem solving systems (COGO,NASTRAN,ICES) and so forth. There is considerable variation in the present characteristics of these systems, but they have as ultimate goal to provide a very high level system involving many built-in problem solving procedures of a substantial nature.

Only a brief discussion of the first four problem areas is presented here because they are either less difficult or already widely discussed in the literature. The next two problem areas are specific to NAPSS-like systems and several pertinent points are discussed which have arisen from an analysis of the pilot NAPSS system. The final two are widely discussed in the literature but still very difficult for a NAPSS-like system. Some alternatives are presented, but the best (or even a good) one is still to be determined.

LANGUAGE DESIGN AND PROCESSING

Ordinary mathematics is the language of NAPSS modulo the requirement of a linear notation. While this linear notation does lead to some unfamiliar expressions, it is not an important constraint. NAPSS has also included a number of conventions from mathematics that are not normally found in programming languages (e.g., $I=2, 4, \cdots, N$). Incremental compilation is used to

obtain an internal text which is then executed incrementally. This approach is, of course, due to the interactive nature of NAPSS.

The creation of this language processor was a very substantial task. The primary difficulties are associated with the fact that all variables are dynamic in type and structure, function variables are data structures (not program structures) and that many operators are quite large and complex due in part to the wide variety of operand types. For example, several integrals may appear in one assignment statement and each of these may lead to interaction with the user. The current NAPSS language processor is relatively slow, partly because of the nature of the language, partly because of the incremental and interactive approach and partly because it is the first one. However, it performs well enough to show that it is not a major barrier to obtaining an acceptable production system.

SIMULATING HUMAN ANALYSIS (ARTIFICIAL INTELLIGENCE)

The original objectives include a large component of automatic problem solving in the NAPSS system. This component lies primarily in the polyalgorithms and manifests itself in two ways. First, there are facilities to analyze the problem at hand and to select an appropriate numerical analysis technique. This analysis continues during the computation and specific techniques may be changed several times during the execution of a polyalgorithm. The second manifestation is in incorporating common sense into the polyalgorithms. This is both difficult and time consuming as it requires a large number of logical decisions and the collection and retention of a large amount of information about the history of the polyalgorithm's execution. The automation of problem solving in NAPSS-like systems leads inevitably to large codes for the numerical analysis procedures. A routine using the secant method may take a few dozen Fortran statements, but a robust, flexible and convenient nonlinear equation polyalgorithm requires many hundreds of statements

INTERNAL SYSTEM ORGANIZATION

NAPSS and NAPSS-like systems are inherently large. The compiler, interpreter, command processor and supervisor are all substantial programs. A polyalgorithm for one operator like integration or solution of nonlinear equations can easily run to 1000 lines of Fortran code. Data structures created during executions may also be quite large (e.g., matrices and arrays of functions). The organization of this system naturally depends on the hardware and operating system environment. The current pilot system is organized with three levels of overlays with a paging system and runs in a memory of about 16,000 words (CDC 6500 words have ten characters or 60 bits or multiple instructions). Many other configurations are feasible and this area does not, in itself, pose a major barrier to an acceptable production system. Currently NAPSS performs quite well provided one ignores operating system influences, i.e., when NAPSS is the only program running. However, it is unrealistic to assume that NAPSS-like systems have large computers dedicated to them or that the operating system gives them preferential treatment (compared to other interactive systems in a multiprogramming environment). Thus the internal system organization is determined by outside factors, primarily by the requirements of the interface with the operating system.

USER INTERFACE

A key objective of NAPSS-like systems is to provide natural and convenient operation. This means that substantial investment must be made in good diagnostic messages, editing facilities, console, and library and file storage facilities. These requirements for a NAPSS-like system are similar to those of a variety of systems. The pilot NAPSS system does not have all these facilities adequately developed and the primary effort was on editing, an operating system might provide many of these facilities in some cases.

A more novel requirement here is the need for access to a lower level language like Fortran. Note that applications of NAPSS-like systems can easily lead to very substantial computations. The intent is for these computations to be done by the polyalgorithms where considerable attention is paid to achieving efficiency. Inevitably there will be problems where these polyalgorithms are either inapplicable or ineffective. Detailed numerical analysis procedures (e.g., Gauss elimination) are very inefficient if directly programmed in NAPSS and thus some outlet to a language with more efficient execution is needed. In such a situation, NAPSS is a problem definition and management system for a user provided numerical analysis procedure.

There are several limits on this possibility due to differences in data structures and other internal features. An analysis of the pilot NAPSS system indicates, however, that a useful form of this facility can be provided with a reasonable effort.

NUMERICAL ANALYSIS

A NAPSS-like system requires at least ten substantial numerical analysis procedures:

1. Integration
2. Differentiation
3. Summation of infinite series
4. Solution of linear systems (and related items like matrix inverses and determinants)

5. Matrix eigenvalues
6. Interpolation
7. Least squares approximation (of various types)
8. Solution of nonlinear equations
9. Polynomial zeros
10. Solution of ordinary differential equations

The objective is to automate these numerical analysis procedures so that a user can have statements like:

```
      ANS← ∫F(X), (X←A TO B)
EQ2:  X ↑ 2*COS(X)−F(X)/(1+X)=A*B/2
      G(X)=F(X−ANS)/(1+X)
      SOLVE EQ2 FOR X
EQ3:  Y''(T)−COS(T)Y'(T)+TY(T)=G(T−X)−ANS
      SOLVE EQ3 FOR Y(T) ON (0,2) WITH Y(0)←0,
        Y(2)←3
```

The user is to have confidence that either these procedures are carried out accurately or that an error message is produced.

These procedures grow rapidly in size as one perfects the polyalgorithms. One polyalgorithm developed for the current NAPSS system is about 2500 Fortran statements (including comments). This large size does not come from the numerical analysis which constitutes perhaps 20 percent of the program. It comes from simulation of common sense (which requires numerous logical and associated pieces of information), the extensive communication facilities for interaction with the user and various procedures for searching, checking accuracy and so forth, aimed at providing robustness and reliability. A further source of complexity is the fact that all of these polyalgorithms must automatically interface. Thus we must be able to interpolate or integrate a function created by the differential equation solver as a tabulated function (or table valued function), one of the function types provided in NAPSS. Since this output is not the most convenient (or even reasonable) input to an integration polyalgorithm, one must make a special provision for this interface. For example in this case NAPSS could have a completely seperate polyalgorithm for integrating table valued functions or it could use a local interpolation scheme to obtain values for the usual polyalgorithm. The latter approach is taken by the pilot NAPSS system.

In addition to numerical analysis procedures, NAPSS currently has a symbolic differentiation procedure and numerical differentiation is only used as a back-up for those functions which cannot be differentiated symbolically (e.g., the gamma function). One may use Leibniz rules for differentiating integrals and piecewise symbolically differentiable functions present may be handled symbolically, so the numerical back-up procedure is infrequently used. It is noted below that future versions of NAPSS should have more function types and that there should be considerably more symbolic analysis of the program. If these features are added, then a number of additional symbolic procedures must also be added including at least a reasonable symbolic integration procedure.

NAPSS currently has two basic types of functions, the symbolic function and the tabulated (or discrete) function. (There are also the standard built-in functions.) Both of these may internally generate substantial data structures. Consider, for example, the result of a very common process, Gram-Schmidt orthogonalization. The program in NAPSS may well appear as:

```
/* DEFINE QUADRATIC SPLINE */
Q(X)←X ↑ 2 FOR X>=0
  ←0
S(X)←.5(Q(X)−3Q(X−1)+3Q(X−2)−Q(X−3))
/* FIRST THREE LEGENDRE POLYNOMIALS */
B(X)[0]←1, B(X)[1]←X, B(X)[2]←1.5X ↑ 2−.5
/* GRAM-SCHMIDT FOR ORTHOGONAL BASIS */
FOR K←3,4,· · ·, 21 DO
  T←(K−2)/10−1, TEMP(X)←S((X−T)/10)
  TEMP(X)←TEMP(X)−SUM((∫TEMP(Y)B(Y)[J],
  (Y← −1 TO 1)), J←0,1,· · ·, K−1)
  B(X)[K]←TEMP(X)/(∫TEMP(Y) ↑ 2, (Y← −1 TO 1))
    ↑ .5;
```

The result is an array $B(X)[K]$ of 22 quadratic spline functions orthogonal on the interval $[-1,1]$. These statements currently cause NAPSS difficulty because all of the functions are maintained internally in a form of symbolic text. By the time the 22nd function is defined the total amount of the text is quite large (particularly since $S(X)$ is defined piecewise) and the evaluation time is also large because of the recursive nature of the definition. The integrations are, of course, carried out and constant values obtained.

This difficulty may be circumvented in the pilot NAPSS by changing the code and a non-recursive text representation scheme has been found (but not implemented) which materially reduces the evaluation time in such situations. These remedies, however, do not face up to the crux of the problem, namely many computations involve manipulations in finite dimensional function spaces. NAPSS should have a facility for such functions and incorporate appropriate procedures for these manipulations. This, of course, adds to the complexity of the language processor, but it allows significant (sometimes by orders of magnitude) reductions in the size of the data structures generated for new functions and in the amount of time required for execution in such problems. Once this type function is introduced, then it is natural to simultaneously identify the case of polynomials as a separate function type. Again NAPSS would need appropriate manipulation procedures, but then even a simple symbolic integration procedure would be valuable and allow the program presented above to be executed in a very small fraction of the time now required.

SYMBOLIC ANALYSIS

We have already pointed out the usefulness of symbolic algorithms (e.g., differentiation, integration), but there is also a significant payoff possible from a symbolic analysis of the program. This may be interpreted as source language optimization and, as usual, the goal is to save on execution time by increasing the language processing time. There are three factors that contribute to this situation. First, NAPSS is a significantly higher level (more powerful) language than, say, Fortran and it is much easier to inadvertently specify a very substantial computation. Second, NAPSS allows one to directly transcribe ordinary mathematical formulas into a program. Many mathematical conventions ignore computations and hence, if carried out exactly as specified, lead to gross inefficiency. Finally, the ultimate class of users are people who are even less aware of the mechanics and procedures of computation than the average Fortran programmer. Indeed one of the goals of a NAPSS-like system is to make computing power easily available to a wider class of users.

The second factor is most apparent in matrix expressions where everyone is taught to solve linear equations by (in NAPSS) $X \leftarrow A \uparrow (-1)B$ and matrix expressions like $(D+U^{-1}L)^{-1}L^{-1}U$ are routinely applied to vectors. The inefficiencies of computing inverse matrices are well known and algorithms have been developed for processing each expression without unnecessarily computing matrix inverses. Another simple example comes from integration where the statement

$$D \leftarrow \int(\int F(X)G(Y),\ (X \leftarrow 0 \text{ TO } 1)),\ (Y \leftarrow 0 \text{ TO } 1)$$

is the direct analog of the usual mathematical notation. These two examples may be handled by optimization of single NAPSS statements. This presents no extraordinary difficulty in the current system, but optimization involving several statements presents severe difficulties for the current system design because it is an incremental language processor and all variables are dynamic.

Symbolic analysis of groups of statements is worthwhile and many of these situations are fairly obvious or correspond to optimizations made in common compilers. The following group of statements illustrate a situation unique to NAPSS-like languages (or any language where functions are true variables, i.e., data structures).

$H(X) \leftarrow GA'(X-A)/GB'(A-X)$
$G(X) \leftarrow \int F(T),\ (T \leftarrow 0 \text{ TO } X)$
PLOT $G(X)$ FOR $X \leftarrow 0$ TO 10
SOLVE $Y'(T)+G(T)Y(T)=H(T/10)/(1+G(T))$
  FOR $Y(T)$ WITH $Y(0) \leftarrow 2$ ON $T \leftarrow 0$ TO 10
SOLVE $G(W)/W-H(W-A)=\text{TAN}(W/A)$ FOR W

The first two statements define functions in terms of operators implemented by polyalgorithms (assume that $GA(X)$ or $GB(X)$ cannot be differentiated symbolically) and the last three statements required numerous evaluations of these functions. The straightforward approach now used simply makes these evaluations as needed by the PLOT or SOLVE processors. However, it is obvious that very significant economies are made by realizing that these functions are to be evaluated many times and thus, introducing the following two statements,

APPROXIMATE $H(X)$ AS $HH(X)$ ON 0 TO 10
APPROXIMATE $G(X)$ AS $GG(X)$ ON 0 TO 10

and then by using $HH(X)$ and $GG(X)$ in the last three statements. A good symbolic analysis of the program would recognize this situation and automatically replace the symbolic definition of $H(X)$ and $G(X)$ by the approximations obtained from the approximation algorithm. It is clear that a symbolic analysis would not be infallible in these situations, but it appears that the savings made in the straightforward situations would be significant.

OPERATING SYSTEM INTERFACE

The most likely environment (at this time) for a NAPSS-like system is a medium or large scale computer with a fairly general purpose multiprogramming mode of operation. From the point of view of NAPSS the key characteristics of this environment are (a) The operating system is indifferent to NAPSS, i.e., NAPSS does not receive special priority or resources relative to jobs with similar characteristics. (b) Central memory is too small. (c) Heavy, or even moderate, use of NAPSS in an interactive mode makes a significant adverse impact on the overall operation of the computer. One may summarize the situation as follows: NAPSS is too big to fit comfortably in central memory for semi-continuous interactive use. Thus it must make extensive use of secondary memory. The result is that in saving on one scarce resource, central memory space, one expends large amounts of another equally scarce resource, access to secondary memory.

One may consider five general approaches to the organization of a NAPSS-like system in an effort to obtain acceptable performance at an acceptable cost and with an acceptably small impact on the operating system. The first is to operate in a small central memory area and to be as clever as possible in instruction of programs and the access to secondary storage. In particular, paging would be heavily if not entirely controlled by the NAPSS system in order to optimize transfers to secondary storage. This is the approach used in the current pilot NAPSS system. The second approach is to use the virtual memory facilities of the operating and hardware system and then treat NAPSS as though it were in central memory at all times. The third approach is obtain enough real memory to hold all, or nearly all, of NAPSS. This approach includes the case of running a NAPSS-like system on a dedicated computer. The fourth approach is to limit NAPSS to batch processing use.

The final approach is to use distributed computing involving two processors. One processor is for language processing. A substantial memory is required because quite large data structure may be generated by NAPSS. A minicomputer with a disk might be suitable to handle a number of consoles running NAPSS. The other processor is that of a medium or large scale computer and its function is to execute polyalgorithms. These programs would reside in this central computer's secondary storage rather than in the minicomputer's memory. The necessary language and data structures would be transferred to the main computer when a polyalgorithm is to be executed.

The batch processing approach fundamentally changes the nature of the system and is hard to compare with the others. The other approaches have one or more of the following disadvantages:

1. The performance (response time) may be slow, especially when the computer is heavily loaded.
2. A very substantial investment in hardware is required.
3. The system is difficult to move to a new environment.

The performance of the pilot NAPSS system suggests that each of these approaches can lead to a useful production system. Those that invest in special hardware would no doubt perform better, but it is still unclear which approach gives the best performance for a given total investment (in hardware, software development, execution time and user time).

PORTABILITY

The development of the pilot NAPSS system was a significant investment in software, perhaps 8 to 12 man years of effort. The numerical analysis polyalgorithms are reasonably portable as they are Fortran programs with only a few special characteristics. Indeed one can locate some suitable, if not ideal, already existing programs for some of the numerical analysis. The language processor is very specific to the operating system interface and the hardware configuration. It is about 90 percent in Fortran, but even so changing environments requires perhaps 8 to 12 man months of effort by very knowledgeable people.

NAPSS-like systems must be portable in order to get a reasonable return from the development effort as few organizations can justify such a system on the basis of internal usage. A number of approaches to (nearly) machine independent software do exist (e.g., boot strapping, macros, higher level languages) which are very useful. However, I believe that a survey of widely distributed systems similar to NAPSS in complexity would show that the key is an organization which is responsible for the portability. This organization does whatever is necessary to make the system run on an IBM 360/75 or 370/155, a UNIVAC 1108, and CDC 6600 and so forth. No one has yet been able to move such a system from say an IBM 370 to a CDC 6600 with a week or two of effort.

Another approach is to make the system run on medium and large IBM 360's and 370's (within standard configurations) and ignore the rest of the computers.

The emergence of computer networks opens up yet another possibility for portability, but it is too early to make a definite assessment of the performance and cost of using a NAPSS-like system through a large computer network. Networks also open up the possibility of a really large NAPSS machine being made available to a wide community of users.

REFERENCES

1. Rice, J. R., Rosen S., "NAPSS—A Numerical Analysis Problem Solving System," *Proc. ACM National Conference*, 1966, pp. 51-56.
2. Rice, J. R., "On the Construction of Polyalgorithms for Automatic Numerical Analysis" in *Interactive Systems for Experimental Applied Mathematics*, (M. Klerer and J. Reinfeld, eds.), Academic Press, New York, 1968, pp. 301-313.
3. Rice, J. R., "A Polyalgorithm for the Automatic Solution of Nonlinear Equations," *Proc. ACM National Conference*, 1969, pp. 179-183.
4. Roman, R. V., Symes, L. R., "Implementation Considerations in a Numerical Analysis Problem Solving System" in *Interactive Systems for Experimental Applied Mathematics*, (M. Klerer and J. Reinfeld, eds.), Academic Press, New York, 1968, pp. 400-410.
5. Symes, L. R., Roman, R. V., "Structure of a Language for a Numerical Analysis Problem Solving System" in *Interactive Systems for Experimental Applied Mathematics*, (M. Klerer and J. Reinfelds, eds.), Academic Press, New York, 1968, pp. 67-78.
6. Symes, L. R., "Manipulation of Data Structures in a Numerical Analysis Problem Solving System," *Proc. Spring Joint computer Conference*, AFIPS, Vol. 36, 1970, p. 157.
7. Symes, L. R., "Evaluation of NAPSS Expressions Involving Polyalgorithms, Functions, Recursion and Untyped Variables," in *Mathematical Software* (J. R. Rice, ed.), Academic Press, new York, 1971, pp. 261-274.

The correctness of programs for numerical computation

by T. E. HULL

University of Toronto
Toronto, Canada

ABSTRACT

Increased attention is being paid to techniques for proving the correctness of computer programs, and the problem is being approached from several different points of view. For example, those interested in systems programming have placed particular emphasis on the importance of language design and the creation of well-structured programs. Others have been interested in more formal approaches, including the use of assertions and automatic theorem proving techniques. Numerical analysts must cope with special difficulties caused by round off and truncation error, and it is the purpose of this talk to show how various techniques can be brought together to help prove the correctness of programs for numerical computation.

The changing role of simulation and the simulation councils

by JOHN MCLEOD

Simulation Councils, Inc.
La Jolla, California

ABSTRACT

Simulation in the broadest sense is as old as man. Everyone has a mental model of his world. Furthermore he will use it to investigate—mentally—the possible results of alternative courses of action.

Simulation as we know it, the use of electronic circuits to model real or imaginary things, began about 35 years ago. Since that time we have seen such vast changes in both the tools and the techniques of simulation that only the underlying philosophy remains unchanged.

And the uses and abuses of simulation have changed radically, too. Seldom has a technology, developed primarily to serve one industry—in the case of simulation the aerospace industry--so permeated seemingly unrelated fields as has simulation. Today simulation is used as an investigative tool in every branch of science, and in many ways that by no stretch of the term can be called science.

These changes have had their impact on our society, too. The first Simulation Council was founded in 1952 after we had tried in vain to find a forum for discussion of simulation among the established technical societies. As interest grew other Simulation Councils were organized, and in 1957 they were incorporated and became known as Simulation Councils, Inc. Because the nine regional Simulation Councils now comprise the only technical society devoted exclusively to advancing the state-of-the-art of simulation and serving those people concerned with simulation, we are now known as SCS, the Society for Computer Simulation.

In 1952 the analog computer was the best tool for simulation, and not one of the technical societies concerned with the up-and-coming digital computers was interested in the analog variety. So circumstances, not purpose, decreed that the Simulation Councils should become thought of as the analog computer society. We are not, and never have been; the Society for Computer Simulation is concerned with the development and application of the technology, not the tool!

That being the case, and realizing the applicability of the simulation technology to the study of complex systems in other fields, the society fostered the necessary technology transfer by soliciting and publishing articles describing applications first in medicine and biology, and for the last several years, in the social sciences.

To foster the change in role of simulation from that of a tool for the aerospace industry to that of a means for studying and gaining and understanding of the problems of our society required that the society also change. This change was first reflected in the technical content of our journal *Simulation*. It has always been our policy to publish articles describing unusual applications of simulation, but until a few years ago that was the only reason material describing a socially relevant use of simulation appeared in *Simulation*. Now it is our policy to solicit such articles, and publish as many as are approved by our editorial review board. Therefore much of the material in our journal is now concerned with socially relevant issues.

The Society for Computer Simulation also publishes a *Proceedings* series. Of the three released to date, all are relevant to societal problems.

The changing role of the society is also evidenced by changes in official policy and in the organization itself. The change in policy was elucidated by our President in an article published in the April 1970 issue of *Simulation*, which stated in part"...the Executive Committee feels that [our society's] primary mission today should be to assist people who want to use simulation in their own fields and particularly to assist people who are dealing with the world's most urgent and difficult [societal] problems..."

The principal organizational change is the establishment of the World Simulation Organization to stimulate work towards the development of simulation technology applicable to the study of problems of our society from a global point of view.

Concomitant with the spread of simulation to all disciplines has been the increase in interest within technical societies which are only peripherally concerned with simulation. Although these societies are primarily dedicated to other fields, several have formed committees or special interest groups with aims and objectives similar to those of the Society for Computer Simulation.

However, the Society for Computer Simulation remains the only technical society dedicated solely to the service of those concerned with the art and science of simulation, and to the improvement of the technology on which they must rely. That others follow is a tribute to our leadership.

Up, up and away

by THOMAS NAYLOR

Duke University
Durham, North Carolina

ABSTRACT

In 1961, Jay Forrester introduced economists, management scientists and other social scientists to a new methodology for studying the behavior of dynamic systems, a methodology which he called *Industrial Dynamics*. Following closely on the heels of *Industrial Dynamics* was *Urban Dynamics*, which purported to analyze the nature of urban problems, their cases, and possible solution to these problems in terms of interactions among components of urban systems. More recently, Forrester has come forth with *World Dynamics*. We and the inhabitants of the other planets in our universe are now anxiously awaiting the publication of *Universe Dynamics*, a volume which is to be sponsored by the Club of Olympus, God, the Pope, Buddha, Mohammed, and the spiritual leaders of several other major religions of this world and the universe. Not unlike *World Dynamics* and other books by Jay Forrester, *Universe Dynamics* will be characterized by a number of distinct features. These features will be summarized in this paper.

In this presentation we shall comment on the methodology used by Forrester in *World Dynamics* as well as the methodology which is being set forth by his disciples who publish *The Limits of Growth* and the other people involved in the Club of Rome project. We shall address ourselves to the whole question of the feasibility of constructing models of the entire world and to model structures alternative to the one set forth by Forrester, et al.

It is first necessary to consider what possible objectives one might have in trying to prove programs correct, since different correctness criteria can be relevant to any particular program, especially when the program is to be used for numerical computation. Then it will be shown that careful structuring, along with the judicious use of assertions, can help one to organize proofs of correctness. Good language facilities are needed for the structuring, while assertions help make specific the details of the proof.

Examples from linear algebra, differential equations and other areas will be used to illustrate these ideas. The importance of language facilities will be emphasized, and implications for Computer Science curricula will be pointed out. A useful analogy with proofs of theorems in mathematics and the relevance of this analogy to certification procedures for computer programs will be discussed.

Policy models—Concepts and rules-of-thumb

by PETER W. HOUSE

Environmental Protection Agency
Washington, D.C.

ABSTRACT

The desire to build policy models or models for policy makers is based on two foundations. First, the need to solicit funds to pay for the construction of models means that those who want to construct models have to promise a "useful" product. Since a large portion of the models built are to support some level of policy, public or private, there is a deliberate attempt to promise output which will be useful to the decision process. Secondly, it is clear from history that the advisor to the throne is a coveted position and one dreamed of by many scientists. It is also clear that the day is coming when models will play a large role in making such policy. The advisory role then shifts to the model builder.

Unfortunately, the reality of model development for the policy level does not appear to agree with the rhetoric. This presentation will review the concept of policy models and suggest some rules-of-thumb for building them.

On validation of simulation models

by GEORGE S. FISHMAN

Yale University
New Haven, Connecticut

ABSTRACT

Before an investigator can claim that his simulation model is a useful tool for studying behavior under new hypothetical conditions, he is well advised to check its consistency with the true system, as it exists before any change is made. The success of this validation establishes a basis for confidence in results that the model generates under new conditions. After all, if a model cannot reproduce system behavior without change, then we hardly expect it to produce truly representative results with change.

The problem of how to validate a simulation model arises in every simulation study in which some semblance of a system exists. The space devoted to validation in Naylor's book *Computer Simulation Experiments with Models of Economic Systems* indicates both the relative importance of the topic and the difficulty of establishing universally applicable criteria for accepting a simulation model as a valid representation.

One way to approach the validation of a simulation model is through its three essential components; input, structural representation and output. For example, the input consist of exogenous stimuli that drive the model during a run. Consequently one would like to assure himself that the probability distributions and time series representations used to characterize input variables are consistent with available data. With regard to structural representation one would like to test whether or not the mathematical and logical representations do not conflict with the true system's behavior. With regard to output one could feel comfortable with a simulation model if it behaved similarly to the true system when exposed to the same input.

Interestingly enough, the greatest effort in model validation of large econometric models has concentrated on structural representation. No doubt this is due to the fact that regression methods, whether it be the simple least-squares method or a more comprehensive simultaneous equations techniques, in addition to providing procedures for parameter estimation, facilitate hypothesis testing regarding structural representation. Because of the availability of these regression methods, it seems hard to believe that at least some part of a model's structural representation cannot be validated. Lamentably, some researchers choose to discount and avoid the use of available test procedures.

With regard to input analysis, techniques exist for determining the temporal and probabilistic characteristics of exogeneous variables. For example the autoregressive—moving average schemes described in Box and Jenkins' book, *Time Series Analysis: Forecasting and Control*, are available today in canned statistical computer programs. Maximum likelihood estimation procedures are available for most common probability distribution and tables based on sufficient statistics have begun to appear in the literature. Regardless of how little data is available, a model's use would benefit from a conscientious effort to characterize the mechanism that produced those data.

As mentioned earlier a check of consistency between model and system output in response to the same input would be an appropriate step in validation. A natural question that arises is: What form should the consistency check take? One approach might go as follows: Let $X_1, \cdots, X_n$ be the model's output in n consecutive time intervals and let $Y_1, \cdots, Y_n$ be the system's output for n consecutive time intervals in response to the same stimuli. Test the hypothesis that the joint probability distribution of $X_1, \cdots, X_n$ is identical with that of $Y_1, \cdots, Y_n$.

My own feeling is that the above test is too stringent and creates a misplaced emphasis on statistical exactness. I would prefer to frame output validation in more of a decision making context. In particular, one question that seems useful to answer is: In response to the same input, does the model's output lead decision makers to take the same action that they would take in response to the true system's output? While less stringent than the test first described, its implementation requires access to decision makers. This seems to me to be a desirable requirement for only through continual interaction with decision makers can an investigator hope to gauge the sensitive issues to which his model should be responsive and the degree of accuracy that these sensitivities require.

In the beginning

by HOWARD CAMPAIGNE

Slippery Rock State Teachers College
Slippery Rock, Pennsylvania

ABSTRACT

The history of computers has been the history of two components; memories and software. These two depend heavily on each other, and all else depends on them.

The early computers had none of either, it almost seems in retrospect. The Harvard Mark I had 132 words of 23 decimal digits, usable only for data. ENIAC had ten registers of ten decimals, each capable of doing arithmetic.

It was von Neuman who pointed out that putting the program into the memory of ENIAC (instead of reading it from cards) would increase the throughput. Thereafter computers were designed to have data and instructions share the memory.

The need for larger storage was apparent to all, but especially to programmers. EDVAC, the successor to ENIAC, had recirculating sounds in mercury filled pipes to get a thousand words of storage. The Manchester machine had a TV tube to store a thousand bits.

Then the reliable magnetic core displaced these expedients, and stayed a whole generation. It was only in recent times when larger memories became available that the programmer had a chance. And of course it is his sophisticated software which makes the modern computer system responsive and effective.

Factors affecting commercial computers system design in the seventies

by WILLIAM F. SIMON

Sperry UNIVAC
Blue Bell, Pennsylvania

ABSTRACT

The design of a digital computer for the commercial market today must, of course, face up to the pervasive influence of IBM. But technological maturity in some areas is slowing the rate of change so that designs seem to converge on certain features. Microprogramming of the native instruction set (or sets?) with emulation of a range of older systems is such a feature. Virtual memory addressing may be another. Characteristics of main storage, random access mass storage devices, data exchange media seem to converge while terminals and communications conventions proliferate and diverge. Some reasons for these phenomena are evident; others will be suggested.

Whatever happened to hybrid packaging, thin films, large scale integration, and tunnel diodes? The more general question is: why do some technologies flourish only in restricted environments, or never quite fulfill the promise of their "youth?" Or is their development just slower than we expected? While these answers cannot be absolute, some factors affecting the acceptance of new technologies can be identified.

Factors impacting on the evolution of military computers

by GEORGE M. SOKOL

US Army Computer Systems Command
Fort Belvoir, Virginia

ABSTRACT

This paper will trace Army experience in ADP for the combat environment, with emphasis on the role of software as a factor in influencing computer organization and design. Early Army activity on militarized computers resulted in the Fieldata family of computers, a modular hierarchy of ADP equipment. Subsequently, software considerations and the evolution of functional requirements resulted in extended use of commercially available computers mounted in vehicles. The balance between central electronic logic and peripheral capability is central to the design of militarized computers, but constraints of size, weight and ruggedness have greatly limited the processing capability of fieldable peripheral equipment. The systems acquisition process also impacts on the available characteristics of militarized computers.

Modeling and simulation in the process industries

by CECIL L. SMITH and ARMANDO B. CORRIPIO

Louisiana State University
Baton Rouge, Louisiana

and

RAYMOND GOLDSTEIN

Picatinny Arsenal
Dover, New Jersey

INTRODUCTION

The objective of this paper is to present what is at least the authors' general assessment of the state-of-the-art of modeling and simulation in the process industries, which in this context is taken to include the chemical, petrochemical, pulp and paper, metals, waste and water treatment industries but excluding the manufacturing industries such as the automobile industry. Since a number of texts[1,2,3] are available on this topic for those readers interested in a more technical treatment, this discussion will tend to be more general, emphasizing such aspects as economic justification, importance of experimental and/or plant data, etc.

EXAMPLES

Paper machine

In the process customarily used for the manufacture of paper, an aqueous stream consisting of about 0.25 percent by weight of suspended fiber is jetted by a headbox onto a moving wire. As the water drains through the wire, a certain fraction of the fiber is retained, forming a mat that subsequently becomes a sheet of paper. The wire with the mat on top passes over suction boxes to remove additional water, thereby giving the mat sufficient strength so that it can be lifted and passed between press rolls. It then enters the dryer section, which consists of several steam-heated, rotating cylinders that provide a source of heat to vaporize the water in the sheet. The final sheet generally contains from 5 to 10 percent water by weight.

The paper machine is a good example of a model consisting almost entirely of relationships to describe physical processes. The formation of the mat over the wire is a very complex physical process.[4,5] Initially, the wire has no mat on top, and the drainage rate is high but the retention (fraction of fiber retained on wire) is low. As the mat builds up, the drainage rate decreases and the retention increases. This process continues until all of the free liquid has drained through the wire. In general, these processes are not well-understood (especially from a quantitative standpoint), and as a result, the equations used to describe them have been primarily empirical.

The action of the suction boxes and press rolls is also a physical process, and again are not well-understood. Similarly, the drying of sheets is also a complex physical process. Initially, the sheet contains a high percentage of water, and it is easily driven off. But as the sheet becomes drier, the remaining water molecules are more tightly bound (both chemically and physically) to the fiber, and the drying rate decreases. Quantitatively, the relationships are not well-developed, and again empiricism is relied upon quite heavily.

A model of the paper machine should be capable of relating the final sheet density (lbs/ft^2), sheet moisture, and other similar properties to inputs such as stock flow, machine speed, dryer steam pressure, etc.

TNT process

Whereas the model for the paper machine consists almost entirely of relationships describing physical processes, the description of chemical processes forms the heart of many models. For example, the manufacture of trinitrotoluene (TNT) entails the successive nitration of toluene in the presence of strong concentrations of nitric and sulphuric acids. In current processes, this reaction is carried out in a two phase medium, one phase being largely organic and the other phase being largely acid.[6] According to the currently accepted theory, the organic species diffuse from the organic phase to the acid phase, where all reactions occur. The products of the reaction then diffuse back into the organic phase.

In this process, the primary reactions leading to the production of TNT are well-known at least from a stoichi-

ometric standpoint. However, many side reactions occur, including oxidation of the benzene ring to produce gaseous products (oxides of carbon and nitrogen). These reactions are not well-understood, but nevertheless must be included in a process model. Similarly, the relationships describing the diffusion mechanism are complex and include constants whose quantitative values are not available. In this particular process, the solubility of the organic species in the acid phase is not quantitatively known.

From a model describing the TNT process, one should be able to compute the amount of product and its composition from such inputs as feed flows and compositions, nitrator temperatures, etc.

STEADY-STATE VS. DYNAMIC MODELS

A steady-state model is capable of yielding only the equilibrium values of the process variables, whereas a dynamic process model will give the time dependence of the process variables.

Using the paper machine as an example, the design of the plant would require a model that gives the final sheet moisture, density, and other properties obtained when the inputs are held at constant values for long periods of time. This would be a steady-state model. On the other hand, one of the most difficult control problems in the paper industry occurs at grade change.[7] For example, suppose the machine has been producing paper with a sheet density of 60 lbs/1000ft^2. Now the necessary changes must be implemented so that the machine produces paper with a sheet density of 42 lbs/1000ft^2. Since virtually all of the paper produced in the interim must be recycled, the time required to implement the grade change should be minimized. Analysis of this problem requires a model that gives the variation of the paper characteristics with time. This would be a dynamic model.

ECONOMIC JUSTIFICATION

Due to the complexity of most industrial processes, development of an adequate process model frequently requires several man-years to develop, significant outlays for gathering data, and several hours of computer time. Therefore, some thought must be given to the anticipated returns prior to the start of the project. In essence, there is frequently no return from just developing a model; the return comes from model exploitation.

Virtually every project begins with a feasibility study, which should identify the possible ways via which a model can be used to improve process performance, estimate the returns from each of these, develop specific goals for the modeling effort (specify the sections of the process to be modeled; specify if the model is to be steady-state or dynamic, etc.), and estimate the cost of the modeling efforts. Unfortunately, estimating returns from model exploitation is very difficult. Furthermore, returns can be divided into tangible returns for which dollar values are assigned and intangible returns for which dollar values cannot readily be assigned. For example, just the additional insight into the process gained as a result of the modeling effort is valuable, but its dollar value is not easily assigned. Perhaps the day will come when the value of process modeling has been established to the point where models are developed for all processes; however, we are not there yet.

For many processes, the decision as to whether or not to undertake a modeling project is coupled with the decision as to whether or not to install a control computer, either supervisory or DDC. In this context, perhaps the most likely subjects are plants with large throughputs, where even a small improvement in process operation yields a large return due to the large production over which it is spread.[8] Many highly complex processes offer the opportunity to make great improvements in process operation, but these frequently necessitate the greatest effort in model development.

Typical projects for which a modeling effort can be justified include the following:

1. Determination of the process operating conditions that produce the maximum economic return.
2. Development of an improved control system so that the process does not produce as much off-specification product or does not produce a product far above specifications, thereby entailing a form of "product give-away." For example, running a paper machine to produce a sheet with 5 percent moisture when the specification is 8 percent or less leads to product give-away in that the machine must be run slower in order to produce the lower moisture. Also, paper is in effect sold by the pound, and water is far cheaper than wood pulp.
3. Design of a new process or modifications to the current process.

Although many modeling efforts have been in support of computer control installations, this is certainly not the only justification. In fact, in many of these, hindsight has shown that the greatest return was from improvements in process operation gained through exploitation of the model. In many, the computer was not necessary in order to realize these improvements.

MODEL DEVELOPMENT

In the development of a model for a process, two distinct approaches can be identified:

1. Use of purely empirical relationships obtained by correlating the values of the dependent process variables with values of the independent process variables.
2. Development of detailed heat balances, material balances, and rate expressions, which are then combined to form the overall model of the process.

The first method is purely empirical, whereas the second relies more on the theories regarding the basic mechanisms that proceed within the process.

While it may not be obvious at first, both of these approaches are ultimately based on experimental data. Since regression is used outright to obtain the empirical model, it is clearly based on experimental data. For any realistic process, the detailed model encompassing the basic mechanisms will contain parameters for which no values are available in the literature. In these cases, one approach is to take several "snapshots" of the plant, where the value of as many process variables as possible are obtained. In general, the normal process instrumentation is not sufficient to obtain all of the needed data. Additional recording points are often temporarily added, and samples are frequently taken for subsequent laboratory analysis. With this data available, a multivariable search technique such as Pattern[13,14] can be used to determine the model parameters that produce the best fit of the experimental data.

In efforts of this type, the availability of a digital computer for data logging can be valuable. The proper approach is to determine what data is needed in the modeling effort, and then program the computer to obtain this data from carefully controlled tests on the process. The use of a digital computer to record all possible values during the normal operation of the process simply does not yield satisfactory data from which a model can be developed.

Another point of contrast between the empirical model and the basic model involves the amount of developmental effort necessary. The empirical model can be developed with much less effort, but on the other hand, it cannot be reliably used to predict performance outside the range within which the data was obtained. Since the detailed model incorporates relationships describing the basic mechanisms, it should hold over a wider range than the empirical model, especially if the data upon which it is based was taken over a wide range of process operating conditions.

NUMERICAL METHODS

In the development and exploitation of process models, numerical techniques are needed for the following operations:

1. Solution of large sets of nonlinear algebraic equations (frequently encountered in the solution of steady-state models).
2. Solution of large sets of nonlinear, first-order differential equations (frequently encountered in the solution of unsteady state models).
3. Solution of partial differential equations (usually encountered in the solution of an unsteady-state model for a distributed-parameter system).
4. Determination of the maximum or minimum of a high-order, nonlinear function (usually encountered in either the determination of the model parameters that best fit the experimental data or in the determination of the process operating conditions that produce the greatest economic return).

Only digital techniques are discussed in this section; analog and hybrid techniques will be described subsequently.

In general, the numerical techniques utilized for process models tend to be the simpler ones. The characteristic that generally presents the most difficulties is the size of the problems. For example, the model of the TNT plant described in Reference 9 contains 322 nonlinear equations plus supporting relationships such as mole fraction calculations, solubility relationships, density equations, etc.

In the solution of sets of nonlinear algebraic equations, the tendency is to use direct substitution methods in an iterative approach to solving the equations. In general, a process may contain several recycle loops, each of which requires an iterative approach to solve the equations involved. The existence of nested recycle loops causes the number of iterations to increase significantly. For example, the steady-state model for the TNT process involves seven nested recycle loops. Although the number of iterations required to obtain a solution is staggering, the problem is solved in less than a minute on a CDC 6500.

A few types of equations occur so frequently in process systems that special methods have been developed for them. An example of such a system is a countercurrent, stagewise contact system, which is epitomized by a distillation column. For this particular system, the Theta-method has been developed and used extensively.[10]

In regard to solving the ordinary differential equations usually encountered in dynamic models, the simple Euler method has enjoyed far more use than any other method. The advantages stemming from the simplicity of the method far outweigh any increase in computational efficiency gained by using higher-order methods. Furthermore, extreme accuracy is not required in many process simulations. In effect, the model is only approximate, so why demand extreme accuracies in the solution?

Although once avoided in process models, partial differential equations are appearing more regularly. Again, simple finite difference methods are used most frequently in solving problems of this type.

Maximization and minimization problems are encountered very frequently in the development and exploitation of process models. One very necessary criterion of any technique used is that it must be able to handle constraints both on the search variable and on the dependent variables computed during each functional evaluation. Although linear programming handles such constraints very well, process problems are invariably nonlinear. Sectional linear programming is quite popular, although the conventional multivariable search techniques coupled with a penalty function are also used.

Over the years, a number of simulation languages such as CSMP and MIMIC have been used in simulation.[11] On

the steady-state side, a number of process simulation packages such as PACER, FLOTRAN, and others have appeared.[12] An alternative to these is to write the program directly in a language such as Fortran.

One of the problems in steady-state simulation is the need for extensive physical property data. Many of the steady-state simulation packages have a built-in or readily available physical properties package that is a big plus in their favor. However, many prefer to use subroutines for physical properties, subroutines for the common unit operations, and subroutines to control the iteration procedures, but nevertheless write in Fortran their own master or calling program and any special subroutine for operations unique to their process.

For dynamic process models with any complexity, Fortran is almost universally preferred over one of the simulation languages.

COMPUTATIONAL REQUIREMENTS

With the introduction of computing machines of the capacity of the CDC 6500, Univac 1108, IBM 360/65, and similar machines produced by other manufacturers, the computational capacity is available to solve all but the largest process simulations. Similarly, currently available numerical techniques seem to be adequate for all but the very exotic processes. This is not to imply that improved price/performance ratios for computing machines would be of no benefit. Since the modeling effort is subject to economic justification, a significant reduction in computational costs would lead to the undertaking of some modeling projects currently considered unattractive.

As for the role of analog and hybrid computers in process simulation, no significant change in the current situation is forecast. Only for those models whose solution must be obtained a large number of times can the added expense of analog programming be justified. However, for such undertakings as operator training, the analog computer is still quite attractive.

SUMMARY

At this stage of process modeling and simulation, the generally poor understanding of the basic mechanisms occurring in industrial processes is probably the major obstacle in a modeling effort. Quantitative values for diffusions, reaction rate constants, solubilities, and similar coefficients occurring in the relationships comprising a process model are simply not available for most processes of interest.

This paper has attempted to present the state-of-the-art of process modeling as seen by the authors. This discussion has necessarily been of a general nature, and exceptions to general statements are to be expected. In any case, these should always be taken as one man's opinion for whatever it is worth.

ACKNOWLEDGMENT

Portions of the work described in this paper were supported by the Air Force Office of Scientific Research under Contract F-44620-68-C-0021.

REFERENCES

1. Smith, C. L., Pike, P. W., Murrill, P. W., *Formulation and Optimization of Mathematical Models*, International Textbook Company, Scranton, Pennsylvania, 1970.
2. Himmelblau, D. M., Bischoff, K. B., *Process Analysis and Simulation—Deterministic Systems*, Wiley, New York, 1968.
3. Franks, R. G. E., *Modeling and Simulation in Chemical Engineering*, Wiley-Interscience, New York, 1972.
4. Schoeffler, J. D., Sullivan, P. R., "A Model of Sheet Formation and Drainage on a Fourdrinier," *Tappi*, Vol. 49, No. 6, June 1966, pp. 248-254.
5. Sullivan, P. R., Schoeffler, J. D., "Dynamic Simulation and Control of the Fourdrinier Paper-Making Process," *IFAC Congress*, London, June 20-26, 1966.
6. Slemrod, S., "Producing TNT by Continuous Nitration," *Ordnance*, March-April 1970, pp. 525-7.
7. Brewster, D. B., *The Importance of Paper Machine Dynamics in Grade Change Control*, Preprint No. 17.2-3-64, 19th Annual ISA Conference and Exhibit, New York, October 12-15, 1964.
8. Stout, T. M., "Computer Control Economics," *Control Engineering*, Vol. 13, No. 9, September 1966, pp. 87-90.
9. Williams, J. E., Michlink, F. P., Goldstein, R., Smith, C. L., Corripio, A. B., "Control Problem in the Continuous TNT Process," *1972 International Conference on Cybernetics and Society*, Washington, D.C., October 9-12, 1972.
10. Holland, C. D., *Multicomponent Distillation*, Prentice Hall, Englewood Cliffs, New Jersey, 1963.
11. Smith, C. L., "All-Digital Simulation for the Process Industries," *ISA Journal*, Vol. 13, No. 7, July 1966, pp. 53-60.
12. Evans, L. B., Steward, D. G., Sprague, C. R., "Computer-Aided Chemical Process Design," *Chemical Engineering Progress*, Vol. 64, No. 4, April 1968, pp. 39-46.
13. Hooke, R., Jeeves, T. A., "Direct Search Solution of Numerical and Statistical Problems," *The Journal of the Association for Computing Machinery*, Vol. 8, No. 2, April 1961, pp. 212-229.
14. Moore, C. F., Smith, C. L., Murrell, P. W., IBM SHARE Library, LSU, PATE, SDA No. 3552, 1969.

Needs for industrial computer standards—As satisfied by ISA's programs in this area

by THEODORE J. WILLIAMS

Purdue University
West Lafayette, Indiana

and

KIRWIN A. WHITMAN

Allied Chemical Corporation
Morristown, New Jersey

INTRODUCTION

Never before has the relevancy of institutions been questioned as critically as today. Many of us have now learned what should have always been evident; that the key to relevance is the satisfaction of needs. The computer industry, and those technical societies that support it, should view this new emphasis on service as an opportunity to stimulate its creative talents. The implication of the "future shock" concept requires that we must anticipate problems if we are ever to have enough time to solve them.

But in what way can a technical society serve; should it be a responder or a leader? Ironically, to be effective, it must be both. It must respond to the requests of individuals in the technical community to use the society's apparatus for the development, review and promulgation of needed standards. The development and review stages can be done by groups of individuals, but it remains for the technical society to exert a leadership role to make these standards known and available to all who might benefit from them.

Thus, our purpose here is to bring to your attention two new, and we feel exciting, industrial computer standards developments that have been undertaken by the Instrument Society of America, as well as a discussion of further actions contemplated in this field. The first is RP55, "Hardware Testing of Digital Process Computers." The second is the work cosponsored with Purdue University on software and hardware standards for industrial computer languages and interfaces. This latter work is exemplified by the ISA series of standards entitled S61, "Industrial Computer FORTRAN Procedures," among others. Both standards development projects have succeeded in furthering ISA's commitment "to provide standards that are competent, timely, unbiased, widely applicable, and authoritative."

ACCEPTANCE TESTING OF DIGITAL PROCESS COMPUTERS

Needs

A Hardware Testing Committee was formed in October 1968 because a group of professionals recognized certain specific needs of the process computer industry. The user needed a standard in order to accurately evaluate the performance of a digital computer and also to avoid the costly duplication of effort when each user individually writes his own test procedures. Conversely, the vendor needed a standard to avoid the costly setting up of different tests for different users and also to better understand what tests are vital to the user.

Purpose

The purpose of the committee has been to create a document that can serve as a guide for technical personnel whose duties include specifying, checking, testing, or demonstrating hardware performance of digital process computers at either vendor or user facilities. By basing engineering and hardware specifications, technical advertising, and reference literature on this recommended practice, there will be provided a clearer understanding of the digital process computer's performance capabilities and of the methods used for evaluating and documenting proof of performance. Adhering to the terminology, definitions, and test recommendations should result in clearer specifications which should further the understanding between vendor and user.

Scope

The committee made policy decisions which defined the scope of this recommended practice to:

(1) Concern digital process computer hardware testing rather than software testing. However, certain software will be necessary to perform the hardware tests.
(2) Concern hardware test performance at either the vendor's factory or at the user's site. This takes into account that it would be costly for a vendor to change his normal test location.
(3) Concern hardware performance testing rather than reliability or availability testing. These other characteristics could be the subject for a different series of long term tests at the user's site.
(4) Concern hardware testing of vendor supplied equipment rather than also including user supplied devices. Generally, the vendor supplied systems includes only that equipment from the input terminations to the output terminations of the computer system.
(5) Consider that specific limits for the hardware tests will not exceed the vendor's stated subsystem specifications.
(6) Consider that before the system contract is signed, the vendor and user will agree upon which hardware testing specifications are applicable. It was not the intent of the standard to finalize rigid specifications or set specific rather than general acceptance criteria. This recognizes that there are many differences both in vendor product design and in user requirements.
(7) Consider that the document is a basic nucleus of tests, but other tests may be substituted based on cost, established vendor procedures, and changing state of the art. Although requirements to deviate from a vendor's normal pattern of test sequence, duration or location could alter the effectiveness of the testing, it could also create extra costs.
(8) Consider that the document addresses a set of tests which apply to basic or typical digital process computers in today's marketplace. Where equipment configurations and features differ from those outlined in this standard, the test procedures must be modified to account for the individual equipment's specifications.
(9) Consider that the document does not necessarily assume witness tests (i.e., the collecting of tests for a user to witness). This collection may or may not conform to the vendor's normal manufacturing approach. There are three cost factors which should be considered if a witness test is negotiated:
    a. Added vendor and user manhours and expenses.
    b. Impact on vendor's production cycle and normal test sequence.
    c. Impact on user if tests are not performed correctly in his absence.

Recommended test procedures

It will not be attempted in this paper to detail reasons for particular procedures in the areas of peripherals, environmental, subsystem and interacting system tests. Time will only permit naming the test procedure sections and what they cover. In this way you may judge the magnitude of this undertaking and the probable significance of this recommended practice.

(1) Central Processing Unit—including instruction complement, arithmetic and control logic, input/output adapters, I/O direct memory access channel, interrupts, timers and core storage.
(2) Data Processing Input/Output Subsystems including the attachment circuitry which furnishes logic controls along with data links to the input/output bus; the controller which provides the buffer between the computer and the input/output device itself; and finally the input/output devices themselves.
(3) Digital Input/Output—including operation, signal level, delay, noise rejection, counting accuracy, timing accuracy and interrupt operation.
(4) Analog Inputs—including address, speed, accuracy/linearity, noise, common mode and normal mode rejection, input resistance, input over-voltage recover, DC crosstalk, common mode crosstalk and gain changing crosstalk.
(5) Analog Outputs—including addressing, accuracy, output capability, capacitive loading, noise, settling time, crosstalk and droop rate for sample and hold outputs.
(6) Interacting Systems—including operation in a simulated real time environment in order to check the level of interaction or crosstalk resulting from simultaneous demands on the several subsystems which make up the system.
(7) Environmental—including temperature and humidity, AC power and vibration.

Costs

The committee constantly had to evaluate the costs of recommended tests versus their value. Typical factors affecting the costs of testing are:

(1) Number of separate test configurations required
(2) Methods of compliance
(3) Sequence, duration, and location of tests
(4) Quantity of hardware tested
(5) Special programming requirements
(6) Special testing equipment
(7) Effort required to prepare and perform tests
(8) Documentation requirements

The additional testing costs may be justified through factors such as reduced installation costs, more timely installation, and early identification of application problems.

Documentation

Another unique feature of this recommended practice is that it has given special attention to documentation of evidence of tests performed on the hardware. Three types of documentation are proposed in order that the user may choose what is most appropriate cost-wise for his situation.

Type 1 would include any statement or evidence provided by the manufacturer that the hardware has successfully passed the agreed-upon tests.

Type 2 would be an itemized check list indicating contractually agreed-upon tests with a certification for each test that had been successfully performed.

Type 3 would be an individual numerical data printout; histograms, etc., compiled during the performance of the tests.

It is, therefore, the aim to provide maximum flexibility in documentation related to the testing.

Board of review

The committee was composed of eight vendors, eight users, and two consultants. In addition to the considerable experience and varied backgrounds of the committee, an extensive evaluation by a Board of Review was also required.

Serious effort was given to insuring that a wide cross-section of the industry was represented on the Review Board. Invitations were sent to the various ISA committees, to attendees at the Computer Users Conference, and various computer workshops. Announcements also appeared in *Instrumentation Technology* and *Control Engineering*. Interest was expressed by approximately 250 people, and these received the document drafts. A very comprehensive questionnaire was also sent to each reviewer in order that a more meaningful interpretation of the review could be made. Subsequently, 117 responses were received. In addition to the questionnaire response, other comments from the Review Board were also considered by the appropriate subcommittee and then each comment and its disposition were reviewed by the SP55 Committee. The magnitude of this effort can be judged from the fact that the comments and their disposition were finally resolved on 44 typewritten pages.

The returned questionnaires indicated an overwhelming acceptance and approval of the proposed documents. The respondents, who came from a wide variety of industrial and scientific backgrounds, felt that it would be useful for both vendors and users alike. They gave the document generally high ratings on technical grounds, and also as to editorial layout. Some reservation was expressed about economic aspects of the proposed testing techniques, which is natural considering that more testing is called for than was previously done. However, ninety-one percent of the respondents recommended that RP55.1 be published as an ISA Recommended Practice.[5] Only three percent questioned the need for the document. The responses were also analyzed for any generalized vendor-user polarity. Fortunately, the percentage of those recommending acceptance of the document were in approximate proportion to their percentages as vendors, users, or consultants. In other words, there was no polarization into vendor, user or consultant classes.

The recommended practice was subsequently submitted to the American National Standards Institute and is presently being evaluated for acceptability as an ANSI standard. ISA's Standards and Practices Board has met with ANSI in order to adopt procedures permitting concurrent review by both ISA and ANSI for all future standards.

PRESENT EFFORTS AT PROCESS CONTROL SYSTEM LANGUAGE STANDARDIZATION

As mentioned earlier, standardization has long been recognized as one means by which the planning, development, programming, installation, and operation of our plant control computer installations as well as the training of the personnel involved in all these phases can be organized and simplified. The development of APT and its variant languages by the machine tool industry is a very important example of this. The Instrument Society of America has been engaged in such activities for the past ten years, most recently in conjunction with the Purdue Laboratory for Applied Industrial Control of Purdue University, West Lafayette, Indiana.[4,6]

Through nine semiannual meetings the Purdue Workshop on Standardization of Industrial Computer Languages has proposed the following possible solutions to the programming problems raised above, and it has achieved the results listed below:

(1) The popularity of FORTRAN indicates its use as at least one of the procedural languages to be used as the basis for a standardized set of process control languages. It has been the decision of the Workshop to extend the language to supply the missing functions necessary for process control use by a set of CALL statements. These proposed CALLS, after approval by the Workshop, are being formally standardized through the mechanisms of the Instrument Society of America. One Standard has already been issued by ISA,[7] another is being reviewed at this writing,[8] and a third and last one is under final development.[9]

(2) A so-called Long Term Procedural Language or LTPL is also being pursued. A set of Functional Requirements for this Language has been approved. Since the PL/1 language is in process of standardization by ANSI (the American National Standards Institute), an extended subset of it (in the manner of the extended FORTRAN) will be tested against these requirements.[12] Should it fail, other languages will be tried or a completely new one will be developed.

(3) The recognized need for a set of problem-oriented languages is being handled by the proposed development of a set of macro-compiler routines which will, when completed, allow the user to develop his own special language while still preserving the transportability capability which is so important for the ultimate success of the standardization effort. This latter will be accomplished by translating the former language into one or the other of the standardized procedural languages before compilation.
(4) To establish the tasks to be satisfied by the above languages, an overall set of Functional Requirements has been developed.[10]
(5) In order that all Committees of the Workshop should have a common usage of the special terms of computer programming, the Glossary Committee of the Workshop has developed a *Dictionary for Industrial Computer Programming* which has been published by the Instrument Society of America[11] in book form.

The Workshop on Standardization of Industrial Computer Languages is composed entirely of representatives of user and vendor companies active in the on-line industrial digital computer applications field. Delegates act on their own in all Workshop technical discussions, but vote in the name of their companies on all substantive matters brought up for approval. It enjoys active representation from Japan and from seven European countries in addition to Canada and the United States itself. Procedures used in meetings and standards development are the same as those previously outlined for the Hardware Testing Committee.

As mentioned several times before, it is the aim and desire of those involved in this effort that the Standards developed will have as universal an application as possible. Every possible precaution is being taken to assure this.

The nearly total attention in these and similar efforts toward the use of higher level languages means that the vendor must be responsible for producing a combination of computer hardware and of operating system programs which will accept the user's programs written in the higher level languages in the most efficient manner. A relatively simple computer requiring a much higher use of software accomplished functions would thus be equivalent, except for speed of operation, with a much more sophisticated and efficient computer with a correspondingly smaller operating system.

The present desire on the part of both users and vendors for a simplification and clarification of the present morass of programming problems indicates that some standardization effort, the Purdue cosponsored program, or another, must succeed in the relatively near future.

Future possibilities and associated time scales

The standardized FORTRAN extensions as described can be available in final form within the next one to two years. Some of those previously made have been implemented already in nearly a dozen different types of computers. The actual standardization process requires a relatively long period of time because of the formality involved. Thus, the 1974-75 period appears to be the key time for this effort.

The work of the other language committees of the Workshop are less formally developed than that of the FORTRAN Committee as mentioned just above. Successful completion of their plans could result, however, in significant developments in the Long Term Procedural Language and in the Problem Oriented Languages areas within the same time period as above.

In addition to its Instrument Society of America sponsorship, this effort recently received recognition from the International Federation for Information Processing (IFIP) when the Workshop was designated as a Working Group of its Committee on Computer Applications in Technology. The Workshop is also being considered for similar recognition by the International Federation of Automatic Control (IFAC).

As mentioned, this effort is achieving a very wide acceptance to date. Unfortunately, partly because of its Instrument Society of America origins and the personnel involved in its Committees, the effort is largely based on the needs of the continuous process industries. The input of interested personnel from many other areas of activity is very badly needed to assure its applicability across all industries. To provide the necessary input from other industries, it is hoped that one or more of the technical societies (United States or international) active in the discrete manufacturing field will pick up cosponsorship of the standardization effort presently spearheaded by the Instrument Society of America and, in cooperation with it, make certain that a truly general set of languages is developed for the industrial data collection and automatic control field.

RECOMMENDED PRACTICES AND STANDARDIZATION IN SENSOR-BASED COMPUTER SYSTEM HARDWARE

In addition to the work just described in programming language standardization, there is an equally vital need for the development of standards or recommended practices in the design of the equipment used for the sensor-based tasks of plant data collection, process monitoring, and automatic control. Fortunately, there is major work under way throughout the world to help correct these deficiencies as well.

As early as 1963 the Chemical and Petroleum Industries Division of ISA set up an annual workshop entitled The User's Workshop on Direct Digital Control which developed an extensive set of "Guidelines on Users' Requirements for Direct Digital Control Systems." This was supplemented by an equally extensive set of "Questions and Answers on Direct Digital Control" to define and explain what was then a new concept for the application of digital computers to industrial control tasks. A re-

cently revised version of these original documents is available.[6] The Workshop has continued through the years, picking up cosponsorship by the Data Handling and Computation Division and by the Automatic Control Division in 1968 when it renamed the ISA Computer Control Workshop. The last two meetings have been held at Purdue University, West Lafayette, Indiana, as has the Workshop on Standardization of Industrial Computer Languages described above.

The ESONE Committee (European Standards of Nuclear Electronics) was formed by the EURATOM in the early 1960's to encourage compatibility and interchangeability of electronic equipment in all the nuclear laboratories of the member countries of EURATOM. In cooperation with the NIM Committee (Nuclear Instrumentation Modules) of the United States Atomic Energy Commission, they have recently developed a completely compatible set of interface equipment for sensor-based computer systems known by the title of CAMAC.[1-3,13] These proposals merit serious consideration by groups in other industries and are under active study by the ISA Computer Control Workshop.

Japanese groups have also been quite active in the study of potential areas of standardization. They have recently developed a standard for a process control operator's console (non CRT based)[14] which appears to have considerable merit. It will also be given careful consideration by the Instrument Society of America group.

It is important that the development of these standards and recommended practices be a worldwide cooperative endeavor of engineers and scientists from many countries. Only in this way can all of us feel that we have had a part in the development of the final system and thus assure its overall acceptance by industry in all countries. Thus, both the ISA Computer Control Workshop and the Language Standardization Workshop are taking advantage of the work of their compatriots throughout the world in developing a set of standards and recommended practices to guide our young but possible overly-vigorous field.

While we must be careful not to develop proposals which will have the effect of stifling a young and vigorously developing industry, there seems to be no doubt that enough is now known of our data and control system requirements to specify compatible data transmission facilities, code and signal standards, interconnection compatibility, and other items to assure a continued strong growth without a self-imposed obsolescence of otherwise perfectly functioning equipment.

SUMMARY

This short description has attempted to show some of the extensive standards work now being carried out by the Instrument Society of America in the field of the applications of digital computers to plant data collection, monitoring, and other automatic control tasks. The continued success of this work will depend upon the cooperation with and acceptance of the overall results of these developments by the vendor and user company managements and the help of their personnel on the various committees involved.

REFERENCES

1. *CAMAC—A Modular Instrumentation System for Data Handling*, AEC Committee on Nuclear Instrument Modules, Report TID-25875, United States Atomic Energy Commission, Washington, D.C., July 1972.
2. *CAMAC Organization of Multi-Crate System*, AEC Committee on Nuclear Instrument Modules, Report TID 25876, United States Atomic Energy Commission, Washington, D.C., March 1972.
3. *Supplementary Information on CAMAC Instrumentation System*, AEC Committee on Nuclear Instrument Modules, Report TID 25877, United States Atomic Energy Commission, Washington, D.C., December 1972.
4. Anon., *Minutes, Workshop on Standardization of Industrial Computer Languages*, Purdue Laboratory for Applied Industrial Control, Purdue University, West Lafayette, Indiana; February 17-21; September 29-October 2, 1969; March 2-6; November 9-12, 1970; May 3-6; October 26-29, 1971; April 24-27; October 2-5, 1972; May 7-10, 1973.
5. Anon., "Hardware Testing of Digital Process Computers," ISA RP55.1, Instrument Society of America, Pittsburgh, Pennsylvania, October 1971.
6. Anon., *Minutes, ISA Computer Control Workshop*, Purdue Laboratory for Applied Industrial Control, Purdue University, West Lafayette, Indiana; May 22-24; November 13-16, 1972.
7. Anon., "Industrial Computer System FORTRAN Procedures for Executive Functions and Process Input-Output," Standard ISA—S61.1, Instrument Society of America, Pittsburgh, Pennsylvania, 1972.
8. Anon., "Industrial Computer System FORTRAN Procedures for Handling Random Unformatted Files, Bit Manipulation, and Data and Time Information," Proposed Standard ISA—S61.2, Instrument Society of America, Pittsburgh, Pennsylvania, 1972.
9. Anon., "Working Paper, Industrial Computer FORTRAN Procedures for Task Management," Proposed Standard ISA—S61.3, Purdue Laboratory for Applied Industrial Control, Purdue University, West Lafayette, Indiana, 1972.
10. Curtis, R. L., "Functional Requirements for Industrial Computer Systems," *Instrumentation Technology, 18*, No. 11, pp. 47-50, November 1971.
11. Glossary Committee, Purdue Workshop on Standardization of Industrial Computer Languages, *Dictionary of Industrial Digital Computer Terminology*, Instrument Society of America, Pittsburgh, Pennsylvania, 1972.
12. Pike, H. E., "Procedural Language Development at the Purdue Workshop on Standardization of Industrial Computer Languages," Paper presented at the Fifth World Congress, International Federation of Automatic Control, Paris, France, June 1972.
13. Shea, R. F., Editor, *CAMAC Tutorial Issue, IEEE Transactions on Nuclear Science, Vol. NS-18*, Number 2, April 1971.
14. *Standard Operator's Console Guidebook, JEIDA-17-1972*, Technical Committee on Interface Standards, Japan Electronics Industry Development Association, Tokyo, Japan, July 1972.

Quantitative evaluation of file management performance improvements*

by T. F. McFADDEN

McDonnell Douglas Automation Company
Saint Louis, Missouri

and

J. C. STRAUSS

Washington University
Saint Louis, Missouri

INTRODUCTION

Operating systems generally provide file management service routines that are employed by user tasks to access secondary storage. This paper is concerned with quantitative evaluation of several suggested performance improvements to the file management system of the Xerox Data Systems (XDS) operating systems.

The file management system of the new XDS Universal Time-Sharing System (UTS) operating system includes the same service routines employed by the older operating system—the Batch Time-Sharing Monitor (BTM). Models for both UTS[1] and BTM[2] have been developed to facilitate performance investigation of CPU and core allocation strategies. These models do not, however, provide capability to investigate performance of the file management strategies.

A wealth of literature is available on file management systems. A report by Wilbur[3] details a new file management design for the Sigma Systems. Other articles have been published to define basic file management concepts,[4,5] to discuss various organization techniques[4,5,6] and to improve understanding of the current Sigma file management system.[6,7] However, there is little published work on the performance of file management systems.

The task undertaken here is to develop and test a simple quantitative method to evaluate the performance of proposed modifications to the file management system. Models are developed that reproduce current performance levels and these models are employed to predict the performance improvements that will result from the implementation of specific improvement proposals. The models are validated against measured performance of the McDonnell Douglas Automation Company XDS Sigma 7 running under the BTM operating system.

The models developed here are extremely simple, deterministic representations of important aspects of file management. This use of simple models to represent very complex systems is finding increasing application in computer system performance work. The justification for working with these simple models on this application are twofold:

1. File management system behavior is not well understood and simple models develop understanding of the important processes.
2. When applied properly, simple models can quantify difficult design decisions.

The underlying hypothesis to this and other work with simple models of computer systems is that system behavior must be understood at each successive level of difficulty before proceeding to the next. The success demonstrated here in developing simple models and applying them in the design process indicates that this present work is an appropriate first level in the complexity hierarchy of file management system models.

The work reported here has been abstracted from a recent thesis.[8] Additional models and a more detailed discussion of system measurement and model verification are presented in Reference 8.

The paper is organized as follows. The next section describes the XDS file management system; current capabilities and operation are discussed and models are developed and validated for opening and reading a file. Several improvement proposals are then modeled and evaluated in the third section.

CURRENT XDS FILE MANAGEMENT SYSTEM

This section is divided into three parts: description of the file management capabilities, description of the file manage-

* Abstracted from an M. S. Thesis of the same title submitted to the Sever Institute of Technology of Washington University by T. F. McFadden in partial fulfillment of the requirements for the degree of Master of Science, May 1972. This work was partially supported by National Science Foundation Grant GJ-33764X.

ment system structure, and development and validation of models for the open and read operations.

Description of capabilities

A file, as defined by XDS,[7] is an organized collection of space on the secondary storage devices that may be created, retrieved, modified, or deleted only through a call on a file management routine.

Each file is a collection of records. A record is a discrete subset of the information in a file that is accessed by the user independent of other records in the file. When the file is created the organization of the records must be specified. Records may be organized in a consecutive, keyed, or random format.

The space used by a consecutive or keyed file is dynamically controlled by the monitor; the space used by a random file must be requested by the user when the file is created and it never changes until the file is released.

Open—When a file is going to be used it must be made available to the user via a file management routine called OPEN. When the file is opened, the user may specify one of the following modes: IN, that is, read only; OUT—write only; INOUT—update; OUTIN—scratch. When a file is opened OUT or OUTIN it is being created; when it is opened IN or INOUT it must already exist. The open routine will set some in-core pointers to the first record of the file if it has been opened IN or INOUT and to some free space that has been allocated for the file if it has been opened OUT or OUTIN. These pointes are never used by the user directly. When the user reads or writes a record the in-core pointers are used by file management to trace the appropriate record.

Close—When the user has completed all operations on the file he must call another routine named CLOSE. A file can be closed with RELEASE or with SAVE. If RELEASE is specified then all space currently allocated to the file is placed back in the monitor's free space pool and all pointers are deleted. If SAVE is specified then the in-core pointers are written to file directories maintained by the monitor so that the file can be found when it is next opened.

Read and Write—There are a number of operations that may be performed on records. The two that are of primary interest are read and write. In a consecutive file the records must be accessed in the order in which they are written. Records in a keyed file can be accessed directly, with the associated key, or sequentially. Random files are accessed by specifying the record number relative to the beginning of the file. Random file records are fixed length (2048 characters).

When reading or writing the user specifies the number of characters he wants, a buffer to hold them, and a key or record number.

File management system structure

The structures used to keep track of files and records are described.

File Structure—The account number specified by a calling program is used as a key to search a table of accounts called an account directory (AD). There is an entry in the AD for each account on the system that has a file. The result of the AD search is a pointer to the file directory. There is only one AD on the system and it is maintained in the 'linked-sector' format (double linked list of sector size [256 word] blocks).

Each entry in the AD contains the account number and the disc address of the file directory (FD). An FD is a 'linked-sector' list of file names in the corresponding account. With each file name is a pointer to the File Information Table (FIT) for that file.

The FIT is a 1024 character block of information on this file. It contains security, file organization, and allocation information. The FIT points to the table of keys belonging to this file.

Figure 1 presents a schematic of the file structure.

Sequential Access and Keyed Structure—In describing record access, attention is restricted to sequential accesses. The structure of consecutive and keyed files is identical. Both file organizations allow sequential accesses of records. Because the structures are the same and both permit sequential accesses there is no true consecutive organization. All of the code for this organization is imbedded in the keyed file logic. The only difference between the two structures is that records in a consecutive file may not be accessed directly.

The result of this implementation is that most processors have been written using keyed files rather than consecutive files because there is an additional capability offered by keyed files and there is no difference in speed in sequentially accessing records on either structure. Measurements have established that only 16 percent of the reads on the system are done on consecutive files while 94 percent of the reads on the system are sequential accesses. For these reasons, emphasis is placed on improving sequential accesses.

Once a file is opened there is a pointer in the monitor Current File User (CFU) table to a list of keys called a Master Index (MIX). Each MIX entry points to one of the data granules associated with the file. Data granules are 2048 character blocks that contain no allocation or organization information. Figure 2 is a schematic of the record structure. Each entry in the MIX contains the following fields:

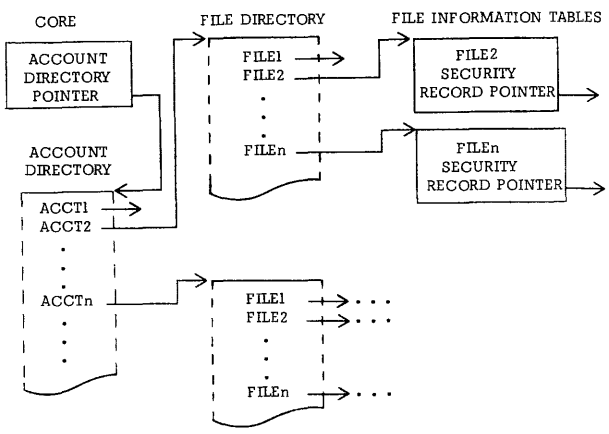


Figure 1—File structure

(a) KEY—In a consecutive file the keys are three bytes in length. The first key is always zero and all others follow and are incremented by one.
(b) DA—The Disc Address of the data buffer that contains the next segment of the record that is associated with this key. Data buffers are always 2048 characters long. The disc address field is 4 characters.
(c) DISP—The byte displacement into the granule of the first character of the record segment.
(d) SIZE—Number of characters in this record segment.
(e) C—A continuation bit to indicate whether there is another record segment in another data granule.
(f) FAK—First Appearance of Key—When FAK = 1 then this entry is the first with this key.
(g) EOF—When set, this field indicates that this is the last key in the MIX for this file. End Of File.

Space Allocation—The Sigma systems have two types of secondary storage devices—fixed head and moving head. The fixed head device is called a RAD (Rapid Access Device); the moving head device is a disc. The allocation of space on these devices is completely and dynamically controlled by the monitor subject to user demands.

The basic allocation unit is a granule (2048 characters) which corresponds to one page of memory. The basic access unit is a sector (1024 characters). A sector is the smallest unit that can be read or written on both devices.

The account directories, file directories, file information tables and master indices are always allocated in units of a sector and are always allocated on a RAD if there is room. Data granules are allocated in single granule units and are placed on a disc if there is room. The SWAP RAD is never used for file allocation.

Table I presents the characteristics for the secondary storage devices referenced here.

If s is the size in pages of an access from device d, the average time for that access ($TA_d(s)$) is:

$$TA_d(s) = L_d + S_d + 2*s*TM_d$$

where:

L_d is the average latency time of device d
S_d is the average seek time of device d
TM_d is the average multiple sector transfer time of device d.

When accessing AD, FD, MIX or FIT sectors, the average transfer time for a single sector from device d is:

$$TS_d = L_d + S_d + T_d$$

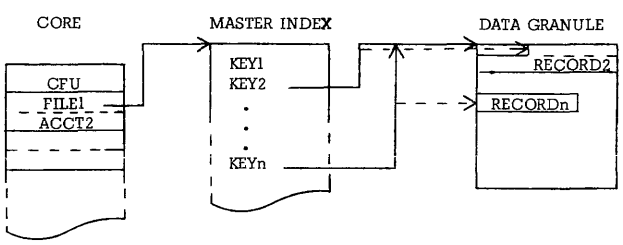


Figure 2—Record structure

TABLE I—Speeds of Sigma Secondary Storage Devices

DEVICE Variable Name	Symbol	7212 RAD SRAD	7232 RAD RAD	7242 DISC PACK DP
Capacity (granules)		2624	3072	12000
Latency (average ms)	L	17	17	12.5
Seek Time—average	S	0	0	75 ms
—range		0	0	25–135 ms
Transfer Time (one sector) (ms per sector)	T	.34	2.67	3.28
Transfer Time (multiple sectors) (ms per sector)	TM	.41	2.81	4.08

where:

T_d is the average single sector transfer time for device d.

Models of current operation

Models are developed and validated for the open file and read record operations. The model algorithms are closely patterned after a simple description of system operation such as found in References 3 or 8. Reference 8 also develops a model for the write operation and describes in detail the measurements for specifying and validating the models.

Open Model—A simple model for the time it takes to open a file is:

$$TO = TPC + TOV + TFA + TFF + TFFIT$$

where:

TPC = time to process the request for monitor services.
TOV = time to read in monitor overlay.
TFA = time to search AD and find the entry that matches the requested account.
TFF = time to search FD and find FIT pointer for the requested file.
TFFIT = time to read FIT and transfer the allocation and organization information to the CFU and user DCB.

The functions for TOV, TFA, TFF and TFFIT can be refined and expressed using the following parameters:

PADR = probability that an AD is on a RAD instead of a disc pack.
PFDR = probability that an FD is on a RAD instead of a disc pack.
PFITR = probability that an FIT is on a RAD.
NAD = average number of AD sectors.
NFD = average number of FD sectors per account.
TAD1 = time it takes to discover that the in-core AD sector does not contain a match.
TFD1 = same as TAD1 for an FD sector.
TAD2 = time it takes to find the correct entry in the in-core AD sector given that the entry is either in this sector or there is no such account.

TABLE II—Observable Open Parameters

PADR	1.00	TO	186.17 ms
PFDR	.872	TAD1	.1 ms
PFITR	.906	TAD2	1.5 ms
NAD	6 sectors	TFD1	.1 ms
NFD	2.6 sectors	TFD2	1.5 ms
SO	2.5 pages	PON	.333
		TPC	1.7 ms

TFD2 = same as TAD2 for an FD sector.
PON = the probability that, when a request is made for the open overlay, it is not in core and therefore the RAD must be accessed.
SO = number of granules occupied by the open overlay on the SWAP RAD.

The time for the open overlay can be expressed as:

$$TOV = PON * TA_{SRAD}(SO)$$

and the time to find the file directory pointer is:

$$TFA = PADR * \frac{NAD}{2} * (TS_{RAD} + TAD1) + (1 - PADR) * \frac{NAD}{2} * (TS_{DP} + TAD1) + TAD2 - TAD1$$

and the time to find the file information table pointer is:

$$TFF = PFDR * \frac{NFD}{2} * (TS_{RAD} + TFD1) + (1 - PFDR) * \frac{NFD}{2} * (TS_{DP} + TFD1) + TFD2 - TFD1$$

and the time to read the file information table is:

$$TFFIT = PFITR * TS_{RAD} + (1 - PFITR) * TS_{DP}$$

Table II contains the values of observable parameters measured in the period January through March, 1972.

Table III contains the values of computable parameters discussed in the open model.

The difference between the two figures, the observed and computed values of TO, is 33 percent. There are a number of ways this figure can be improved:

(a) When the TO of 186 ms was observed, it was not possible to avoid counting cycles that were not actually being spent on opening a file. The system has symbiont activity going on concurrently with all other operations. The symbionts buffer input and output between the card reader, on-line terminals, the RADs and the line printer. So the symbionts steal cycles which are being measured as part of open and they also produce channel conflicts. Neither of these is considered by the model.

TABLE III—Computed Open Parameters

$TA_{SRAD}(SO)$	19.0 ms
TOV	6.3 ms
TFA	60.7 ms
TFF	30.3 ms
TO	125.4 ms

(b) The figure NFD is supposed to reflect the number of file directories in an account. The measured value is 2.6. Unfortunately there is a large percentage (40 percent) of accounts that are very small, perhaps less than one file directory sector (30 files). These accounts are not being used. The accounts that are being used 90 percent of the time have more than three file directory sectors. Therefore if the average number of FD sectors searched to open a file had been observed rather than computed, the computed value for TO would have been closer to the observed TO.

Read Model—To simplify the read model these assumptions are made:

(a) This read is not the first read. The effect of this assumption is that all buffers can be assumed to be full. Since there are an average of 193 records per consecutive file, the assumption is reasonable.

TABLE IV—Observed Read Parameters

TPC	0.65 ms	TMS	2 ms
TTR	1.00 ms	PMIXR	0.585
TR	20.35 ms	NEM	47.7 entries

(b) The record being read exists. This assumption implies the file is not positioned at the end of file. Again, only 1 percent of the time will a read be on the first or last record of the file.

(c) The record size is less than 2048 characters. This assumption is made so that the monitor blocks the record. The average record size is 101.6 characters.

These assumptions not only simplify the model, but they reflect the vast majority of reads.

The time to read a record can therefore be written as:

$$TR = TPC + TGE + TTR + PEC * TGE$$

where:

TPC = time to process request to determine that it is a read request. This parameter also includes validity checks on the user's DCB and calling parameters.
TGE = time to *g*et the next key *e*ntry (even if the next entry is in the next MIX) and make sure that the corresponding data granule is in core.
TTR = time to *t*ransfer entire *r*ecord from monitor blocking buffer to user's buffer.
PEC = the probability that a record has two entries—entry continued.

The probability, PEM, that the next MIX entry is in the resident MIX can be expressed as a function of the average number of entries, NEM, in a MIX sector:

$$PEM = \frac{NEM-1}{NEM}$$

The probability, PEG, that the correct data granule is in core is a function of the number of times the data granule addresses change when reading through the MIX relative to the number of entries in the MIX.

Table IV presents the observed values of the parameters used in the sequential read model. The computed results for the read model are found in Table V.

The difference between the computed and observed values for TR is 42 percent. The error can be improved by refining the observed values to correct the following:

(a) The symbionts were stealing cycles from the read record routines and producing channel conflicts that the timing program did not detect.
(b) In addition, the observed value of TR includes time for reads that were direct accesses on a keyed file. These direct accesses violate some of the read model assumptions because they frequently cause large scale searches of all the Master Index sectors associated with a file.

MODELS OF THE PROPOSALS

In this section, two of the performance improvement proposals presented in Reference 8 are selected for quantitative evaluation. One proposal impacts the open model and the other proposal impacts the read model. The models developed previously are modified to predict the performance of the file management system after implementation of the proposals.

A proposal that impacts the open/close routines

Implementation Description—To preclude the necessity of searching the AD on every open, it is proposed that the first time a file is opened to an account the disc address of the FD will be kept in one word in the context area. In addition, as an installation option, the system will have a list of library accounts that receive heavy use. When the system is initialized for time-sharing it will search the AD looking for the disc address of the FD for each account in its heavy use list. The FD pointers for each heavy use account will be kept in a parallel table in the monitor's data area.

The result is that better than 95 percent of all opens and closes will be in accounts whose FD pointers are in core. For these opens and closes the AD search is unnecessary.

TABLE V—Computed Read Parameters

TR	11.8 ms	PEC	0.042
TGE	9.7 ms	PDG	0.93
PEM	0.979	TRMIX	49.18 ms

Effect on Open Model—For the majority of opens the above proposal gives the open model as:

$$\begin{aligned} TO' &= TPC + TOV + TFF + TFFIT \\ &= TO - TFA \\ &= 125.4 - 60.7 = 64.7 \end{aligned}$$

This represents a percentage improvement of 51.5 percent. (The figures used for TO and TFA are in Table III).

A proposal that impacts the read/write routines

Implementation Description—The significant parameter in the read model is the time to get the next entry, TGE. There are two expressions in TGE which must be considered: the first is the average time to get the next Master Index entry; the second, is the average time to make sure the correct data granule is in core. The levels of these expressions are 1.03 and 6.7 ms. It is apparent that the number of times that data granules are read is contributing a large number of cycles to both the read and write models.

One of the reasons for this, of course, is the disproportionately large access time on a disc pack compared to a RAD. Nevertheless it is the largest single parameter so it makes sense to attack it. A reasonable proposal to decrease the number of data granule accesses is to double the buffer size. The model is developed so that the size of the blocking buffer parameter can be varied to compare the effect of various sizes on the read and write model.

Effect of Proposal on the Read Model—The proposal outlined above will impact only one parameter in the read, PDG. PDG represents the probability that the correct data granule is in core. Its current value is 0.93.

Now, there are three reasons that a Master Index entry will point to a different data granule than the one pointed to by the entry that preceded it:

1. The record being written is greater than 2048 characters and therefore needs one entry, each pointing to a different data granule, for every 2048 characters.
2. The original record has already been overwritten by a larger record, requiring a second Master Index entry for the characters that would not fit in the space reserved for the original record. The second entry may point to the same data granule but the odds are ten to one against this because there are, on the average, 10.6 data granules per file.
3. There was not enough room in the data granule currently being buffered when the record was written. When this occurs the first Master Index entry points to those characters that would fit into the current data granule and the second entry points to the remaining characters that are positioned at the beginning of the next data granule allocated to this file.

The first two reasons violate the assumptions for the read model and are not considered further here.

The third reason is the only one that will be affected by changing the data granule allocation. It follows that if there

TABLE VI—Impact of Blocking Buffer of Size N Pages on Read Model

N (pages)	PDG (ms)	TGE (ms)	TR (ms)
1	.930	9.729	11.78
2	.955	7.337	9.20
5	.970	5.903	7.80
10	.975	5.424	7.30

are 10.6 data granules per file by doubling the allocation size there will be 5.3 data 'granules' per file. This effectively divides by two the probability that a record had to be continued because of the third item above. Tripling the size of the blocking buffer and data 'granule' would divide the probability by three.

The question to be resolved at this point is: What share of the 7 percent probability that the data granule address will change can be attributed to the third reason above?

A reasonable approximation can be developed by the following argument:

(a) There are 2048*n characters in a data 'granule'.
(b) There are 101.6 characters per record.
(c) Therefore there are 20.15*n records per data 'granule'.
(d) The 20*n record will have two Master Index entries.
(e) Then, on the average, one out of every 20*n entries will have a data granule address that is different from the address of the preceding entry. This probability is 1/(20.15*n) which is .0496/n.

Then for n = 1, as in the original read and write models, 5 of the 7 percent figure for 1-PDG is attributable to records overlapping data granule boundaries. The actual equation for PDG is:

$$PDG = 1 - \left(.02 + \frac{.05}{n}\right)$$

where n is the number of granules in the monitor's blocking buffer.

The impact of various values of n on the read model is listed in Table VI. It is obvious that by increasing the blocking buffer size, the performance of the read model can be improved. However the amount of improvement decreases as n increases.

If there are no other considerations, a blocking buffer size of two promises an improvement of 21 percent in the read routine. Perhaps a user should be allowed to set his own blocking buffer size. A heavy sort user that has very large files, for example, can be told that making his blocking buffers three or four pages long will improve the running time of his job. A final decision is not made here because the profile of jobs actually run at any installation must be considered. In addition, there are problems like: Is there enough core available? Is the swap channel likely to become a bottleneck due to swapping larger users? These problems are considered in Reference 8 and found not to cause difficulty for the operational range considered here.

CONCLUSIONS

This paper does not pretend to develop a complete file management system model. Such a model would necessarily contain model functions for each file management activity and some means of combining these functions with the relative frequency of each activity. The model result would then be related to system performance parameters such as the number of users, the expected interactive response time and the turn-around time for compute bound jobs.

The described research represents an effort to model a file management system. The model is detailed enough to allow certain parameters to be changed and thus show the impact of proposals to improve the system.

The development of the model is straightforward, based on a relatively detailed knowledge of the system. This type of model is sensitive to changes in the basic algorithms. The model is developed both to further understanding of the system and to accurately predict the impact on the file management system of performance improvements.

Work remains to be done to integrate the model into overall system performance measures. However comparisons can be made with this model of different file management strategies. Development of similar models for other systems will facilitate the search for good file management strategies.

REFERENCES

1. Bryan, G. E., Shemer, J. E., "The UTS Time-Sharing System—Performance Analysis and Instrumentation," *Second Symposium on Operating Systems Principles*, October 1969, pp. 147-158.
2. Shemer, J. E., Heying, D. W., "Performance Modeling and Empirical Measurements in a System Designed for Batch and Time-Sharing Users," *Fall Joint Computer Conference Proceedings*, 1969, pp. 17-26.
3. Wilbur, L. E., *File Management System, Technical Design Manual*, Carleton University, 1971.
4. Chapin, N., Common File Organization Techniques Compared," *Fall Joint Computer Conference Proceedings*, 1969, pp. 413-422.
5. Collmeyer, A. J., "File Organization Techniques," *IEEE Computer Group News*, March 1970, pp. 3-11.
6. Atkins, D. E., Mitchell, A. L., Nielson, F. J., "Towards Understanding Data Structures and Database Design," *Proceedings of the 17th International Meeting of the XDS Users Group*, Vol. 2, November 1971, pp. 128-192.
7. *Xerox Batch Processing Monitor (BPM)*, Xerox Data Systems, Publication No. 901528A, July 1971.
8. McFadden, T. F., *Quantitative Evaluation of File Management Performance Improvements*, M. S. Thesis, Washington University, St. Louis, Missouri, May 1972.

A method of evaluating mass storage effects on system performance

by M. A. DIETHELM

Honeywell Information Systems
Phoenix, Arizona

INTRODUCTION

A significant proportion of the cost and usefulness of a computing system lies in its configuration of direct access mass storage. A frequent problem for computing installation management is evaluating the desirability of a change in the mass storage configuration. This problem often manifests itself in the need for quantitative decision criteria for adding a fast(er) direct access device such as a drum, disk or bulk core to a configuration which already includes direct access disk devices. The decision criteria are hopefully some reasonably accurate cost versus performance functions. This paper discusses a technique for quantifying the system performance gains which could be reasonably expected due to the addition of a proposed fast access device to the system configuration. It should be noted that the measurement and analysis techniques are not restricted to the specific question of an addition to the configuration. That particular question has been chosen in the hope that it will serve as an understandable illustration for the reader and in the knowledge that it has been a useful application for the author.

The system performance is obviously dependent upon the usage of the mass storage configuration, not just on the physical parameters of the specific device types configured. Therefore, the usage characteristics must be measured and modelled before the system performance can be estimated. This characterization of mass storage usage can be accomplished by considering the mass storage space as a collection of files of which some are permanent and some are temporary or dynamic (or scratch). A measurement on the operational system will then provide data on the amount of activity of each of the defined files. The mass storage space is thereby modelled as a set of files, each with a known amount of I/O activity. The measurement technique of file definition and quantification of I/O activity for each is described first in this paper along with the results of an illustrative application of the technique.

The next step in predicting the system performance with an improved (hopefully) mass storage configuration is to decide which files will be allocated where in the revised mass storage configuration. The objective is to allocate the files in such a manner as to maximize system performance. In the case of adding a fast device to the configuration, this objective is strongly correlated, within reasonable limits, with an allocation policy which maximizes the resulting I/O activity to the fastest device in the mass storage configuration. This allocation policy can be mathematically modelled as an integer linear programming problem which includes the constraint of a specified amount of fast device storage capacity.

Having the file allocations and resulting I/O activity profiles for a range of fast access device capacities, the expected system performance change can be estimated by use of an analytical or simulation model which includes the parameters of proportionate distribution of I/O activity to device types and device physical parameters as well as CPU and main memory requirements of the job stream. The results of application of an analytic model are described and discussed in the latter paragraphs as a prelude to inferring any conclusions. The analytic model is briefly described in the Appendix.

MASS STORAGE FILE ACTIVITY MEASUREMENT

The first requirement in determining the files to be allocated to the faster device is to collect data on the frequency of access to files during normal system operation. Thus the requirement is for measurements of the activity on the existing configuration's disk subsystem. Such measurements can be obtained using either hardware monitoring facilities or software monitoring techniques. Hardware monitoring has the advantage of being non-interfering; that is, it adds no perturbation to normal system operation during the measurement period. A severe disadvantage to the application of hardware monitoring is the elaborate, and expensive, equipment required to obtain the required information on the frequency of reference to addressable, specified portions of the mass storage. The preferred form of the file activity data is a histogram which depicts frequency of reference as a function of mass storage address. Such histograms can be garnered by use of more recent hardware monitors

which include an address distribution capability, subject to, of course, the disadvantages of cost, set up complexity, and monitor hardware constrained histogram granularity. A more flexible method of gathering the required information is a software monitor. This method does impose a perturbation to system operation but this can be made small by design and code of the monitor program. It has the strong advantage of capturing data which can be analyzed after the fact to produce any desired reports with any desired granularity. A software monitor designed to work efficiently within GCOS* was utilized to obtain file activity data for the analysis described for illustration.

This 'privileged' software obtains control at the time of initiation of any I/O command by the processor and gathers into a buffer information describing the I/O about to be started and the current system state. This information, as gathered by the measurement program used for this study includes the following:

Job Characteristics
- Job and activity identification
- File identification of the file being referenced
- Central Processing Unit time used by the job

Physical I/O Characteristics
- Subsystem, channel and device identification
- I/O command(s) being issued
- Seek address
- Data transfer size

The information gathered into the data buffer is written to tape and subsequently analyzed by a free standing data reduction program which produced histograms of device and file space accesses, seek movement distances, device utilization and a cross reference listing of files accessed by job activities. Of primary concern to the task of selecting files to be allocated to a proposed new device are the histograms of device and file space accesses.

A histogram showing the accesses to a device is shown in Figure 1. This histogram is one of 18 device histograms resulting from application of the previously described measurement techniques for a period of 2 1/4 hours of operation of an H6070 system which included an 18 device DSS181 disk storage subsystem. The method of deriving file access profiles will be illustrated using Figure 1. The physical definition of permanently allocated files is known to the file system and to the analyst. Therefore, each area of activity on the histograms can be related to a permanently allocated file if it is one. If it is not a permanently allocated area, then it represents the collective activity over the measurement period of a group of temporary files which were allocated dynamically to that physical device area. Figure 1 depicts the activity of some permanent files (GCOS Catalogs, GCOS-LO-USE, SW-LO-USE, LUMP) and an area in which some temporary files were allocated and used by jobs run during the

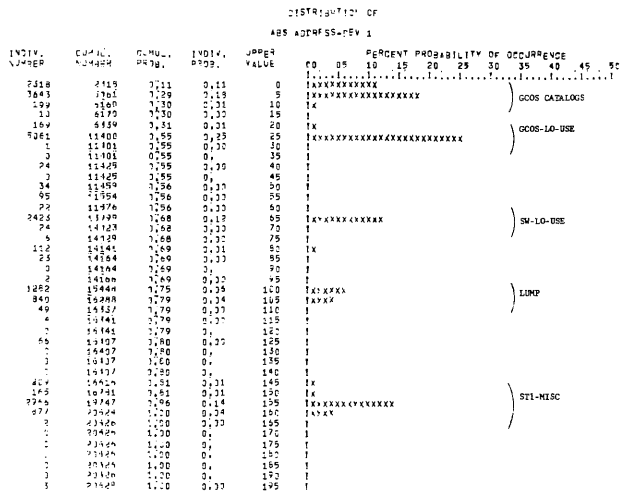


Figure 1—Histogram of accesses to device 1 space

measurement period (ST1-MISC). The leftmost column of the histogram gives the number of accesses within each 5 cylinder area of the disk pack and consequently is used to calculate the number of accesses to each file, permanent or temporary, defined from the activity histograms for each device. Often the accessing pattern on a device is not concentrated in readily discernible files as shown in Figure 1 but is rather randomly spread over a whole device. This is the case with large, randomly accessed data base files as well as for large collections of small, user files such as the collection of programs saved by the systems' time sharing users. In these cases the activity histograms take the general form of the one shown in Figure 2. In this case no small file definition and related activity is feasible and the whole pack is defined as a file to the fast device allocation algorithm. The resulting files defined and access proportions for the mentioned monitoring period are summarized in Table I. The unit of file size used is the "link," a GCOS convenient unit defined as 3840 words, or 15360 bytes. Table I provides the inputs to the file allocation algorithm which is described in the following section.

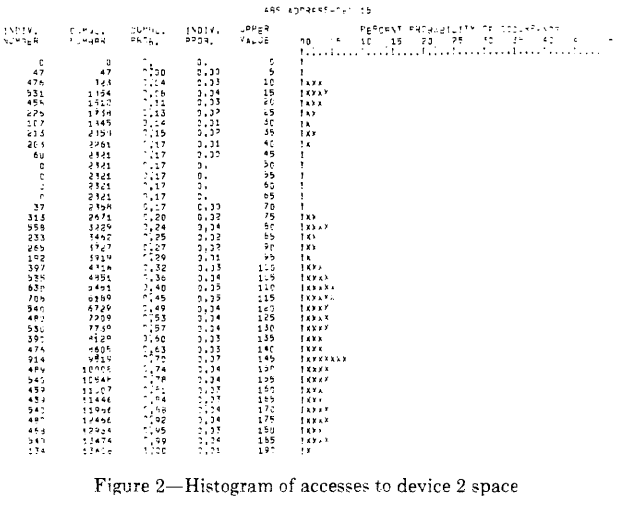


Figure 2—Histogram of accesses to device 2 space

* GCOS is the acronym for the General Comprehensive Operating Supervisor software for H6000 systems.

TABLE I—Measured File Activity

Dual 6070—Monitoring Period Approx. 2.25 Hours
DSS181 (16 used devices)—149180 Accesses
Exposure (3D) Workload (6/23/72)

File Name	Size (Links)	Activity (%)
GCOS CAT	80	4.1
GLOS-HI-USE	18	12.7
GLOS-LO-USE	140	3.5
SU-HI-USE	40	0.9
SW-LO-USE	150	1.7
SW-SYSLIB	40	1.3
LUMP	20	0.8
# P	55	0.9
# S	200	4.5
PACK16	750	3.9
STI-MISC	150	2.8
GLOAD FILES	150	4.7
SYOU 2	1200	9.1
DEV. CATS	360	2.6
PACK 11	1200	3.1
PACK10	1200	4.1
D6 SCRATCH	100	3.1
D7 SCRATCH	180	2.0
D13 C SCRATCH	90	1.4
D3 SCRATCH 1	150	2.7
D3 SCRATCH 2	90	1.0
D4 MISC	600	3.3
SYOU 1	1200	9.0
D7 MISC	600	1.4
PACK8	1200	2.7
PACK9	1200	4.0
OTHER	7940	8.6

Note: 1 Link = 3840 Words = 15360 bytes.

OPTIMAL FILE ALLOCATION TO FAST DEVICE WITH LIMITED CAPACITY

Having a set of mass storage files defined as well as a measured profile of the frequency of access to each, the next step is to postulate an allocation of these files to the mass storage subsystems. For purposes of illustration it will be assumed that this allocation problem may be characterized as the selection of that subset of files which will fit on a constrained capacity device. The selected files will then result in the maximum proportion of I/O activity for the new device being added to the mass storage configuration. This problem of selecting a subset of files may be formulated as an integer linear programming application as follows.

GIVEN

A set of n pairs (s_i, f_i), defined as:

s_i = Size (links) of the ith file,
f_i = Fractional frequency of reference to the ith file.

A maximum size constraint, S, of the proposed fast access device.

THE PROBLEM

Select from the given set of files that subset which maximizes the sum of the reference frequencies to the selected subset while keeping the sum of the selected subset file sizes less than or equal to the given fast device capacity limitation.

MATHEMATICAL FORMULATION

Define

$$\delta_i = \begin{cases} 0 - i^{\text{th}} \text{ file is not selected} \\ 1 - i^{\text{th}} \text{ file is selected for allocation to fast device.} \end{cases}$$

Then the problem is to find,

$$\text{MAX } z = [\sum_{i=1,n} f_i \cdot \delta_i]$$

Subject to,

$$\sum_{i=1,n} s_i \cdot \delta_i \leq S$$

This is an integer linear programming problem for determination of the δ_i. The Integer Programming applications program available with H6000 systems was used to find solutions for various sizes of proposed fast access devices. For the same data discussed previously the integer programming solution yielded the results shown in Table II and Figure 3.

Table II shows the same list of files, sizes and access frequencies given in Table I but a column has been included for each postulated size of fast access device to be added to the system configuration. An X in a row indicates that that file should be allocated to a fast access device of the capacity given by the column heading. Thus the X's in each column delineate which files to be allocated to the fast access device to provide the maximum number of fast device accesses for a specified capacity.

Figure 3 shows the activity profile for "optimum" file allocation as a function of the capacity on a $\log_2$ scale. The fairly constant slope of approximately 10% activity increase perdoubling of the capacity is an interesting result. In the range of 2-32 million bytes, the same slope characteristic has been observed in several applications of this technique.

Application of the preceding technique provides information on the I/O traffic split between the proposed fast access device and the existing mass storage and it also dictates which files to allocate to the fast access device to provide the maximum activity for a specified capacity. The effects of implementing the derived allocations on system performance must still be quantified. One method of estimating the resulting system performance is described in the following section.

SYSTEM PERFORMANCE IMPLICATIONS

The preceding section has discussed a technique for selecting a set of files to be allocated to the proposed fast access mass storage device. It produced not only the allo-

TABLE II—"Optimum" File Allocation to Fast Access Device

File Name	Size (Links)	Activity (%)	Fast Access Device Capacity (M Bytes) 1	2	4	8	16	32	64
GCOS CAT	80	4.1		X	X	X	X	X	X
GLOS-HI-USE	18	12.7		X	X	X	X	X	X
GLOS-LO-USE	140	3.5				X	X	X	X
SU-HI-USE	40	0.9					X	X	X
SW-LO-USE	150	1.7						X	X
SW-SYSLIB	40	1.3			X		X	X	X
LUMP	20	0.8		X	X	X	X	X	X
# P	55	0.9						X	X
# S	200	4.5					X	X	X
PACK16	750	3.9							
STI-MISC	150	2.8					X	X	X
GLOAD FILES	150	4.7				X	X	X	X
SYOU 2	1200	9.1							X
DEV. CATS	360	2.6						X	
PACK 11	1200	3.1							
PACK10	1200	4.1							
D6 SCRATCH	100	3.1			X	X	X	X	X
D7 SCRATCH	180	2.0						X	X
D13C SCRATCH	90	1.4					X	X	X
D3 SCRATCH 1	150	2.7						X	X
D3 SCRATCH 2	90	1.0						X	X
D4 MISC	600	3.3							
SYOU 1	1200	9.0							X
D7 MISC	600	1.4							
PACK8	1200	2.7							
PACK9	1200	4.0							
OTHER	7940	8.6							
	19200	100.0		118L 17.6%	258L 22.0%	508L 28.9%	1028L 39.8%	2013L 50.7%	4153 Size 66.2% Act.
	149180 Accesses Approx. 2.25 Hours								

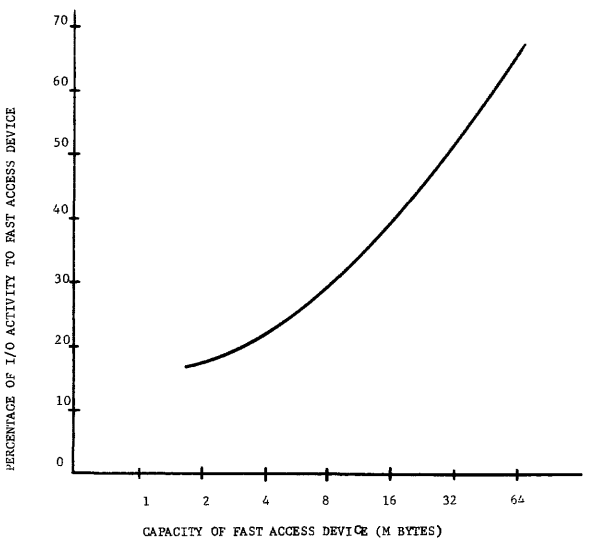


Figure 3—I/O activity resulting from "optimum" file allocation to proposed device

cation set but also the resulting proportionate I/O activity as a function of the capacity of the proposed device. The choice of capacity of the proposed device, or even to procure it, depends not directly on the proportionate activity but rather on the system performance that results from each possible activity level. In this section the system performance is discussed as estimated by an analytical model.

The analytical model used to determine the system throughput is described in the Appendix. It specifically includes the depth of multiprogramming, the job processor requirement, and the number, type and speed of one or two IO subsystems in the calculation of system throughput. The mathematical assumptions made in the analytical model are included in the Appendix information. An additional assumption, operational in nature, is that the system always has jobs to work on—it will never have to decrease its effective depth of multiprogramming due to an insufficient number of jobs being fed into it. It is also assumed that the processing and high speed channel load on the system due to unit record devices is negligible. Similarly, only batch type job streams are

being modelled, hence no communications subsystem overhead is included. As a result of these pragmatic assumptions, the determined throughput will be optimistic. However, the calculated throughputs for different configurations should have the same proportional relationships as the real systems.

The inputs required by the analytical model and the values used for the illustrative analysis are summarized in Table III below.

The output of the model consists of system throughput in terms of job completion rate as well as percentage utilization factors for the major system resources, e.g., CPU, memory and I/O channels. The object of this analysis is to determine the effectiveness of alternative mass storage configurations. Therefore system performance, in the throughput ("work done") sense, will be measured as the proportionate utilization of a system resource which is the same in each proposed configuration—in this case the processors. The best mass storage configuration, assuming a given processor, is that which keeps the given processor saturated. This definition of system performance goodness is only valid, of course, in the throughput sense, certainly it is not necessarily valid in the turn-around or response time sense of performance. Regardless of the relative merits of this definition of system performance, it will suffice for purposes of illustration. As mentioned before, any analytical or simulation model of performance commensurate with the analyst's requirements may be employed at this step of the analysis procedure.

Figure 4 shows the normalized system performance predicted by the analytical model for a group of system configurations which exemplify the possible mass storage reconfiguration options. The curves are plotted as a function of the capacity of a proposed, fast, direct access device such as a drum or bulk store. The higher curve shows system performance as a function of the capacity of a device with an average access time (including data transfer time) of 1 millisecond. The lower curve is for a device with an average access time of 10 milliseconds.

TABLE III—Model Inputs

Description	Value(s) used	
Central System	*H6070*	
CPU Time Per Physical I/O	12 milliseconds	
Average Multiprogramming Depth	4 and 7	
Disc Subsystems	*DSS181*	*DSS190*
Number of Drives	18	16
Number of Physical Channels	2	2
Average Seek Time (ms)	34	30
Average Latency Time (ms)	12.5	8.3
Average Data Transfer Time (ms)	7	3
Drum Subsystems	*No Specific Product*	
Average Access Time, including Data Transfer (ms)	1 and 10	
Distribution of I/O activity to Disc or Drum Subsystems	As specified by Figure 3 for Drum Capacity	

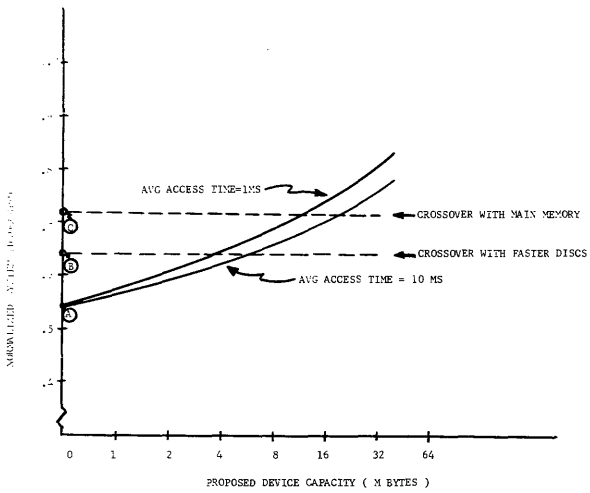


Figure 4—System performance vs. reconfiguration proposals

Points on the vertical axis of Figure 4 represent configurations which do not include a new, drum type device. Point A represents the measured system which had a mass storage configuration of DSS181 devices. Point B is the predicted system performance resulting from replacing the DSS181 subsystem by a DSS190 subsystem which also consists of movable arm discs but with faster seek, rotation and data transfer capabilities. Point C represents the predicted performance for a system whose mass storage configuration is unchanged (the original DSS181 subsystem) but whose main memory has been increased to enable a multiprogramming depth of 7 simultaneous production batch jobs. Horizontal lines intersecting the performance curves for the drum-like configurations have been drawn from points B and C. These then show the performance cross over points which can be factored into a cost trade-off analysis. It can be noted from these that a 4 megabyte drum produces the same performance improvement as upgrading the movable arm subsystem, but less than increasing the main memory capacity. To equal the performance gain accrued from extending the main memory, a drum capacity of at least 16 megabytes is required for the example system.

Cost/performance information can be readily inferred by replacing capacities and configuration by relevant cost data. This will not be pursued here as it is necessarily dependent upon choices of equipment suppliers as well as the details of any particular facilities' operations.

SUMMARY

The preceding has described a method of using measurements and mathematical analysis to determine, quantitatively, the gains in system throughput to be expected from various alternative changes to the system configuration. The steps in this method proceed as follows.

1. Obtain accurate measurements of the access characteristics of file activity on the current system during real operations.

2. Postulate allocations of files to various feasible capacities of the proposed new device. This could be done by subjective inspection of measurement results or, as described in the preceding text, a mathematical technique of allocating for maximum use of a proposed faster device could be employed.
3. Using this allocation-activity information predict system performance for the various feasible capacities of the proposed device by use of analytic or simulation models. This information can then be translated into cost-performance relationships for use as one of the criteria in the decision process.

The above procedure is useful and informative. It is obviously neither foolproof nor all inclusive. The file activity information, as used, represents average activity over a monitoring period which may or may not be representative of "most" periods of system operation. Further no sequential relationships (the patterns of activity shifting from one file to another during the period) are utilized although the measurement technique could provide this information to a reasonable degree, e.g., a Markov transition matrix of access from each file to the next one accessed. The pros and cons of any and all analytic and simulation models are also always a subject of controversy. Nevertheless it can be concluded that a scientific procedure such as the one described, which uses currently available measurement and analysis techniques to provide reasonable estimates of system performance gives a significant assist to the decision process.

APPENDIX

The Queuing Model of System Performance

There are various techniques of evaluating the performance of computer systems. The two more direct approaches are measurement and analysis of models. In the case of systems not yet in a productive situation, measurement cannot be done. Analysis of models, however, can be used to evaluate performance of even quite abstract hypothetical systems. There are two popular techniques of computer system modelling. One is simulation and the other is analytic mathematical modelling. Simulation has proven a successful technique but is typically a major project, costly in both men and computer time. Analytical modelling however, usually results in a small computer time requirement for system evaluation. This permits parametric or sensitivity analyses not always economically feasible with a simulation model. An analytic model, *which includes consideration of the basic computing system features* (e.g., main memory limitation, file subsystems, and processor competition), is a useful aid in evaluating system level performance of computer systems.

The particular model used in this analysis was developed by Don R. Rice.* Figure 5 is a sketch of the model system. When a job enters the system, it is placed in the system input queue. If there is room in memory (less than K jobs already resident) the job immediately is allocated to memory. The job then gets, in its turn, alternate processor quantums and I/O services from each device in a specified proportion. When its total processing and I/O requirements have been met, the job leaves the system. This departure leaves an unoccupied slot in memory which is immediately allocated to the earliest arriving job in the system input queue.

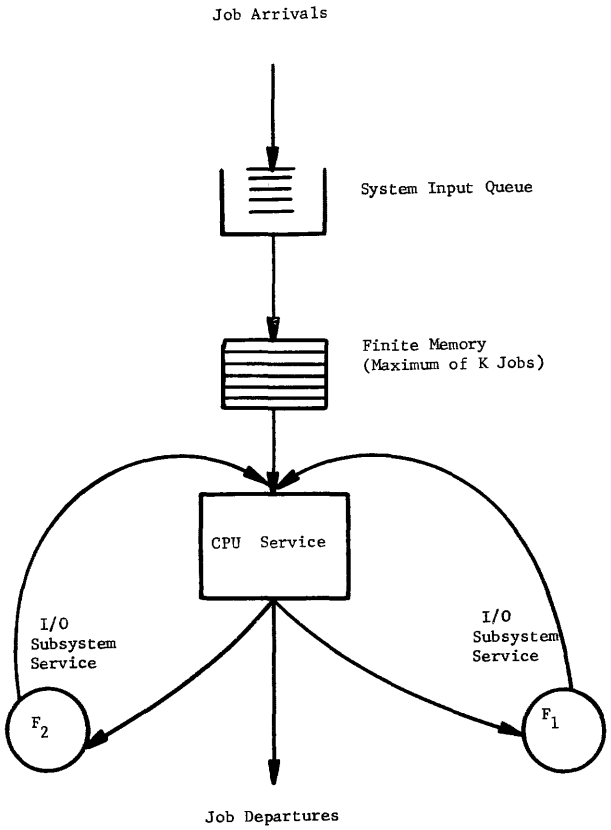


Figure 5—The system model.

The mathematical technique of evaluation such a system requires certain assumptions:
These are:

(1) Each user of the system behaves independently of all other users.
(2) All users are statistically identical.
(3) Arrivals into the system are Poisson distributed.
(4) Each increment of service by the processor(s) is exponentially distributed with mean value C/f where C is the mean processor requirement per job and f is the mean total number of I/O connects per job.
(5) The time required for each file service is exponentially distributed with mean value F_i where F_i is the mean effective access time for file device i.

The described system operation and assumptions permit formulation of the problem as a continuous parameter Markov chain as a basis for analysis.

* D. R. Rice, An Analytical Model for Computer System Performance Analysis, Ph D Dissertation, University of Florida, 1971.

The memory bus monitor—A new device for developing real-time systems

by RICHARD E. FRYER

Naval Weapons Center
China Lake, California

INTRODUCTION

The memory bus monitor was designed to assist in program development on dedicated computers. A dedicated computer is defined here to be one that is used for only one major application and is available to the programmer in blocks of time as needed to complete the development and checkout of his problem. Because of high cost, few large scale computers can be operated in this manner. However, many medium scale and virtually all minicomputers (including airborne and other process control computers) are operated this way, at least when assembly language code is being written. Most systems classified as real-time also fall in this category.

The cost of such systems hardware is decreasing at such a rate that their number is dramatically increasing. Progress in reducing software costs is virtually nonexistent, even though it is generally regarded as being the limiting factor in the development of these systems.[1] These reasons give economic justification for the development of tools to assist in the programming task.

Most instrumentation and monitoring techniques have been developed since 1965.[2] A survey of bibliographies on systems measurements[3,4] also reveals that software evaluation and measurement techniques are being explored at a much more intense level than hardware approaches—probably because computer hardware designers and users continue to be distinct varieties with relatively little cross-pollination. The predominance of measurement techniques has been developed to assist in improving time-sharing systems or to aid comparison of large systems. Very little emphasis has been placed on the needs of programmers, and even less on applications for medium and small scale systems.

Some instrumentation aids have been in use since the early 50's. The breakpoint register has been implemented on many systems at least as old as the IBM 650[5] and as large as the UNIVAC 1108. This register allows the programmer to execute a program at full speed until the instruction register (or perhaps a general register) agrees with the breakpoint value; then halt. The user can then step through the program one instruction at a time (a practice that is probably discouraged at 1108 installations). In spite of the simplicity of this device, it can be a great help to the programmer. It has been reinvented a number of times in the last few years by vendors of small machines.

REAL-TIME SYSTEMS DEVELOPMENT

Development of software systems for real-time applications typically proceeds through definition and specification, design, implementation and checkout stages. Due to difficulties in predicting subprogram size and timing for various functions, it isn't usually known if overall size and timing goals will be met until the complete system is benchmarked. When original design is incomplete or system specifications are revised, the goals my be exceeded by a substantial margin. Then program cycle time or memory requirements (sometimes both) must be reduced without adversely affecting the other parameter. And this optimization must take place while program debugging continues.

Machine software simulators and assembly language debugging programs (such as Digital Equipment Company's ODT) are among the tools available at this stage of system design. Many aspects of system design can be verified with these tools—logical flow, variable scaling and correctness of computation, for example—but they are rather useless for measuring cycle time, locating faults that occur during peak I/O bus loading, detecting race conditions in a multiple asynchronous interrupt environment and similar problems. A program trace recorded on tape can yield data on the actual program path taken (at least over short time periods), but is expensive to reduce, is not interactive, and can only approximate real execution times.

The memory bus monitor is intended to supplement these and other development tools. It is useful for initial testing, checkout, optimization, and system validation. Timing and instruction mix data obtained with the system are also expected to be useful in future system designs.

MEMORY BUS INFORMATION

The information carried on the memory bus of a typical system has a simple format. Address lines specify the

memory location to be accessed. Data lines carry the data read or to be written, and control information includes a Read/Write (R/W) line and sometimes a split cycle line for read-modify-write operations. This information is adequate for the operation of the memory bus monitor. An Instruction/Data (I/D) status line is also sometimes available. It specifies how the current bus data word will be interpreted; its use is described in the operation section. Other lines sometimes available include the Direct Memory Access control line. While this and other control lines can reveal still more information for the user, they are not discussed further in this article.

The stream of addresses and data traveling the memory bus, though simple in form, contains much information about the execution of a program; information not ordinarily available to a programmer. The rich content of bus information can be extracted using time and address correlation of bus traffic. Several specific operations on bus data and the program parameters that they measure are listed below.

1. The behavior of any selected variable is easy to follow by identifying when its address is upon the bus. Actual instead of intended behavior is observable.
2. The address(es) holding any specified data word that is accessed may also be obtained by monitoring data values and treating the address as an unknown.
3. One of the most valuable artifacts available from the bus is the history of addresses just prior to the time when a specific location is written. This instruction stream tells the programmer what code was executing when an instruction or program constant is accidentally destroyed.
4. Another artifact that is often referred to but rarely measured is the instruction mix for a particular section of code. If the code is short enough, a "static" mix may be determined by hand or via a program editor. However, large sections of code become unmanageable and generally unknown branching ratios at decision points make "dynamic" (actual) mixes difficult to predict.[6] Dynamic mixes are easily obtained with the bus monitor.
5. Branching ratios are also readily extracted from the bus.
6. Accurate and detailed timing information provides the most effective way to optimize program cycle time. Measuring the actual execution time of a section of code is easily accomplished using the bus monitor, in contrast with conventional techniques that, for example, require changing existing instructions (such s NOPs) to uncommitted output instructions to generate timing sentinels.

HARDWARE DESIGN

A basic system

The block diagram in Figure 1 shows the essential elements of a minimum bus monitor. It can be used to

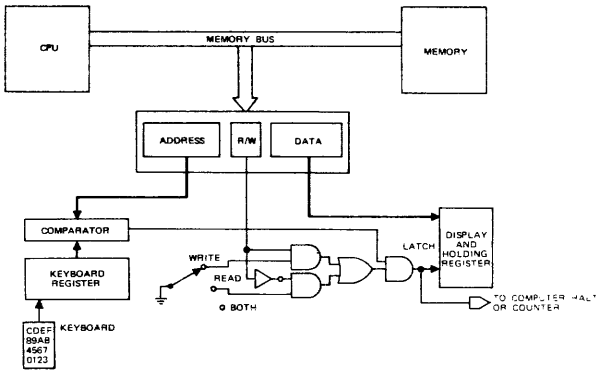


Figure 1—A basic bus monitor

display the contents of any memory location as it is read or written. The user enters the address of the location to be monitored and selects Read, Write, or both. The data values read or written are then displayed in the readout. A static/dynamic switch (not shown) allows the operator to stop the display if the update rate is too fast. A read status indicator can be set to blink or latch each time the selected location is read. The trigger to the readout register is also connected to a BNC connector on the front panel so that an oscilloscope or digital counter may be used for frequency, period, or counting operations. Interrupt and I/O rates can be monitored this way, for example. This signal may be ORed with the halt signal in some computers to effect the breakpoint function.

An intermediate system

A diagram of the first system constructed appears in Figure 2. It was designed for a Varian 622A computer in 1969, and has one primary feature missing from the previous system. The address stack is added consisting of an 8 word shift register. Each new address that appears on the bus is pushed onto this stack.

The control logic that is used to latch the display register can also be routed to freeze the stack. When the address compare signal halts the stack, the user can manually revolve the stack past the display to inspect memory addresses (and thus instruction activity) prior to

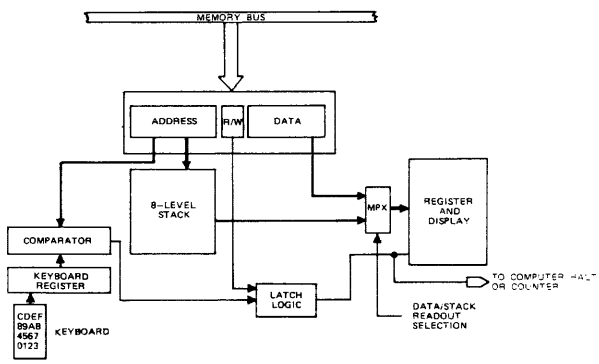


Figure 2—The monitor used on the Varian 622A

the selected read or write operation. This feature was commonly used to locate the section of code responsible for destroying a known location.

The current design of the memory bus monitor

A simplified block diagram of the Memory Bus Monitor appears in Figure 3. In addition to the functions that existed in the earlier design, the current design has the following features.

1. The shift register stack is replaced with a 16 word Content Addressable Memory (CAM) that is used as a stack, a learning memory, or as a 16 word comparator depending on mode.
2. A 16 word data stack is also added with 2 extra bits for the R/W status line and the I/D line when available. The data register (number 4 in Figure 3) and comparator can generate a trigger when a data word equal to the register contents appears on the bus. Bits set in the mask register (number 5) inhibit comparison of corresponding data bits. That is, they become "don't care" bits.
3. The address detector is expanded to 4 registers (numbers 0—3). Register 1 is used with register 0 to bracket sections of code and to straddle branches. An output is also generated that combines registers 0 and 1 to detect any address falling between (0) and (1), where (*n*) denotes the contents of register *n*.
4. The next two address registers, numbers 2 and 3, are enabling registers. They are used for arming conditions. Comparator pulses from these two are also used to set and reset an RS flip-flop. The output of this flip-flop can act as a gate to pass other events.
5. The base address register, number 6, and the associated adder are used to add a constant to keyboard address entries and to subtract that constant from address readouts. This feature is nearly essential for effective use with relocated code.
6. A watchdog timer function is implemented with register 7 and an up-counter. The internal 20MHz monitor clock is counted down to supply a 1 microsecond and a 1 millisecond trigger rate for the timer. The timer may be operated in either "gate mode" that compares the time interval of the RS gate with (7) or in a "cycle mode" that compares the time duration between pulses. In either case, failures (measured intervals greater than the register value) generate a trigger.
7. Finally, though not shown in Figure 3, an octal/hexadecimal switch controls the keyboard and readout mode to match the form used by the computer under test.

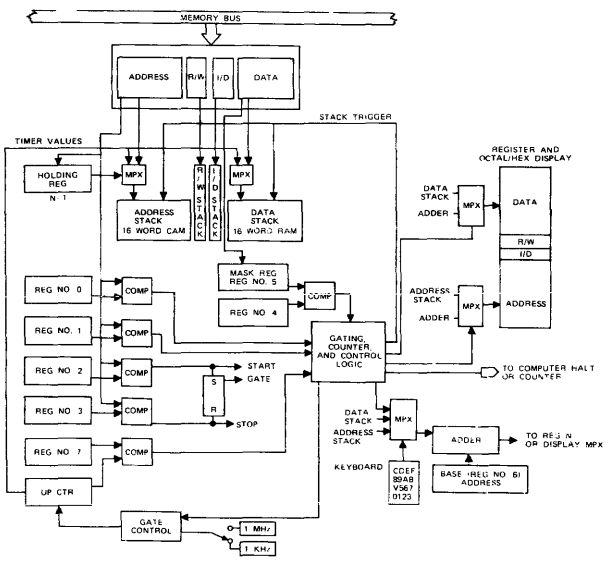


Figure 3—The current memory bus monitor

The current monitor is considerably more sophisticated than the earlier design, but is of no more than moderate complexity (simpler than a disc controller, for example). The unit is constructed using wire-wrap and TTL technology. Approximately 150 SSI and MSI packages are required for the design, excluding the interface to the memory bus. The bus interface is on a separate board that can be easily replaced for transportability between various 16 bit computers. For machines with a larger address or data word only the corresponding registers and display need to be enlarged. The device is mounted in a 19″×12″×3 1/2″case—one selected to fit along with a general purpose counter into a small suitcase.

The operators panel of the monitor has the following features. A hexadecimal keyboard, 2 digit thumbwheel switch, and an Address-Data display allow the user to load and verify the contents of the 8 registers. Also, the CAM storage appears at thumbwheel addresses 10_8 to 27_8; the data RAM at 30_8 to 47_8. Each comparator output and the two derived signals are routed to connectors on the panel. Toggle switches adjacent to the connectors allow the user to select a combination of these signals to form a "master" trigger that is routed to the display and stack. The selected signals may be ANDed or ORed to form a master trigger. This composite trigger is also routed to both a pulse and latching indicator. Control is also provided for the watchdog timer modes and for the condition to be used to halt the operation of the monitor. A 7 position selector switch gives access to the various modes of operation. Status lights reflect the condition of the timer and halt logic.

OPERATION MODES AND THEIR USES

The *Register R/W* mode is used to load or modify registers at the beginning of a run, and to review CAM and RAM contents after a run. Pushbuttons on the panel set the R/W and I/D bits in registers 0 through 4, and these bits are compared with the bus status lines along with the address bits.

The *Bus Monitor* mode routes the contents of the memory bus address and data registers to the display registers. This mode is useful when operating the computer in single step mode.

The *Load Stack* mode uses the trigger source(s) selected by the user to transfer current address and data registers to the stacks. These registers are also transferred to the display. Typical monitoring operations in this mode are described below.

1. The activity of a specific location is extracted from the bus by putting its address in one of the 4 address registers and enabling the comparator output to generate the master trigger with the corresponding toggle switch. More than one register can be used if desired (4 is the maximum). The trigger generated pushes the bus address and data registers onto their respective stacks and also transfers this data to the display. The R/W enabling bits for the 4 registers need not be the same. The stack continues to load until the halt flip-flop is set by the stack full trigger or any other trigger selected by the user.
2. Addresses containing a specific data word may be located as this word is read or written by loading and enabling register 4. These addresses are displayed and pushed onto the stack. The mask register is used to limit the comparison to, for example, the op code field.
3. Using the data and mask registers in conjunction with an address register allows a limited study of variable scaling.

The *Halt Stack* mode allows all address and data information to push the stacks, and halts this operation when the master trigger occurs. Recent history of bus traffic is the most important artifact in this mode.

1. The instruction stream that references a given word can be recovered—particularly useful if a data word is destroyed as mentioned for the earlier design.
2. Registers 2 and 3 are used to set and reset the RS flip-flop. The output then enables any other triggers, allowing the measurement mentioned above to take place only when an event arms the device, or while within (or outside) a defined section of code. The (0) $\leq$ ADDRESS $\leq$ (1) trigger can be similarly used for routines or data arrays instead of single words.
3. When a certain number of tasks must be completed in a specific time, the watchdog timer is used to halt the stack, allowing the task currently executing to be identified.

System Timing and *Counting* operations are done in both of the above two modes. The connectors on the front panel can be connected to a general purpose counter (a Hewlett Packard 5325 has been used). Common timing operations are described next.

1. Program loop cycle time is determined by monitoring the frequency or period of the trigger from an address comparator.
2. Subroutine (or any section of code) timing is done by measuring the period of the RS flip-flop with registers 2 and 3 set to the limits of the routine.
3. When a section of code calls another routine, is itself interrupted, or has multiple exit points the time measured using the RS flip-flop as described above is in error. The correct value can be determined by using the (0) $\leq$ ADDRESS $\leq$ (1) signal, since it falls to zero when the address falls outside the limits.
4. The effective cycle time is determined from the memory cycle trigger that is routed from the bus interface to the front panel. This measurement can aid the user in determining if faster memory would speed up his overall system.

Several specific counting operations that are useful are now described.

1. The branching ratios at a decision point are readily determined by setting one register to the branch instruction and another to an instruction in one of the branches. The counter is operated in the ratio mode to give a direct ratio reading. All branches can be quickly checked to insure that the sum of all ratios is 1.
2. The number of occurrences of an instruction with a given op code may be counted by using registers 5 and 6 to isolate the op code part of a word. A wait loop is often a part of real-time programs, and if the idle time is significant, the instruction mix becomes contaminated. The address range can then be limited to exclude the wait loop in a manner already described. This same technique is used to determine a mix for a specific task.

The monitor trigger that is derived for timing is also often useful for synchronizing an oscilloscope during hardware checkout. Events within the computer and on the I/O bus that result from instruction execution can be 'anticipated' by the monitor.

The *Address Sieve* technique operates the CAM as a learning memory. It is used to locate all instructions that refer to a specific location. The user derives a trigger in the usual way, and it causes the *previous* address (saved in the holding register shown in Figure 3) to be applied to the CAM. If that address is not already in the memory, it is added.

The *Selective Dump* mode uses the CAM as a 16 word comparator. The user loads the CAM with up to 16 locations that he wishes to capture; then sets up a halt trigger. When the CAM detects a match, the address of the match in the CAM (that is, 0 to 15) is supplied to the RAM and the data register is stored in the corresponding RAM location.

The *Watchdog Timer* runs in all modes and generates triggers that may be combined for the master or halt triggers. When the timer mode is selected, however, measured time intervals are pushed onto the stack. The RAM and CAM are both loaded to provide 32 values. The user may store all counter values or only those that exceed the value in register 7. The timer values are also routed to the display.

DIRECTIONS FOR DEVELOPMENT

When a user wants to know how long a given section of code takes to execute, he would really like to know the extremes and something about the distribution. The current monitor is unsuitable for collecting all measurements for statistical analysis since it is a manually operated device. Anticipating this problem, some effort was made to ease the transition to computer control of the monitor. Also, the counter mentioned allows for remote programming and data collection. There are many modes and instances, however, when the increased cost and complexity of a general purpose computer would not be justified. Developments in microcomputers make the inclusion of such a device in an advanced monitor an attractive option to consider, especially if compilers are written for them. We envision that both manual and automatic monitors will find their way into the programmers toolkit.

SUMMARY

This paper has briefly reviewed the software development problem for real-time and process control applications, and has shown how memory bus information may provide data to aid the development of these systems. A device is described that can extract many different artifacts from the memory bus with relative ease. The device is inexpensive, small, and can be switched from machine to machine with little effort.

REFERENCES

1. Kossiakoff, A., Sleight, T., "A Programming Language for Real-Time Systems," *AFIPS Fall Joint Computer Conference*, 1972.
2. Goodman, A., "Measurement of Computer Systems—An Introduction," *AFIPS Fall Joint Computer Conference*, 1972.
3. Menck, H., *Bibliography on Measurement of Computer Systems*, ACM Los Angeles Chapter, Special Interest Committee on Measurement and Evaluation, (unpublished), 1971.
4. Miller, E., "Bibliography on Techniques of Computer Performance Analysis," *Computer*, Vol. 5, No. 5, 1972.
5. Watson, R., *Time Sharing System Design Concepts*, McGraw-Hill, New York, p. 235, 1970.
6. Foster, C., Gonter, R., Riseman, E., "Measures of Op-Code Utilization," *IEEE Transactions on Computers*, C-20, No. 5, 1971.

Design and evaluation system for computer architecture

by KATUSYA HAKOZAKI, MASAHIRO YAMAMOTO, TAKAKI ONO, NAOYA OHNO, and MAMORU UMEMURA

Nippon Electric Company, Ltd.
Kawasaki, Japan

INTRODUCTION

Recent progress of IC and LSI technology and advances in microprogramming technique[4,10] have had a major impact on computer architecture. Rapidly decreasing logic and memory costs and ever increasing programming costs justify more trade-offs being carried out from software to firmware and/or hardware, and many experimental[1,3,9,12,16] or commercial[11,14] models, which stand for these trade-offs, have been announced.

However, it seems that only a little is known about performance gain obtained by these trade-offs. One of the main reasons for the lack of evaluation data is that there are only a few tools to quantitatively evaluate quality of architecture, firmware or hardware design. There are three major approaches to this purpose.

One approach is the use of software simulators[17] which, sometimes, are very useful, if appropriate simulation models can be established. However, in most cases, it is not an easy task to make a good simulation model for accurate design evaluation. When one wants to simulate a processor in detail, at the register transfer level, for example, to evaluate the processor performance using a software simulator, it will take a long time both to make a simulation model and to set up simulations of many parameters. Moreover, there is no way to automatically convert a simulation model to an actual model, after the desired model has been completely defined.

Another approach is to make an experimental model on a conventional microprogrammed processor[12] by replacing its microprogram memory with another one representing the model. In this approach, a higher simulation speed can be obtained and processor speed improvements and cost figures can be directly compared with the conventional processor on which the model is implemented. Difficulties, however, lie in that available hardware resources are limited to the processor used, and that evaluation data gathering is difficult unless some measurement means are provided.

The third approach to mathematical or theoretical treatment[2,4,6] of the model can be used to obtain roughly estimated values, and may be of supplementary use at this evaluation stage.

To cope with above mentioned problems of slow simulation speed, large gaps between evaluation models and actual models of software simulations, and insufficient evaluation data of hardware simulator approach, the Computer Design and Evaluation System (CODES), described in this paper, has been developed. It can be used to evaluate performance gain by a sort of hardware simulation approach with rapid simulation speed, a wide adaptability range, effective measurement capability and usable supporting softwares.

The system configuration will be described, and application examples of CODES to the architecture or computer design problems will be projected in the subsequent sections.

CODES SYSTEM

System configuration

The CODES system consists of cooperable GPMS, General Purpose Microprogram Simulator, and SYDAS, System Data Acquisition System, operable dependently or independently.

Therefore rapid simulation is attained by means of hardware simulator, GPMS, and an accurate and wide range of evaluation data can be obtained by means of hardware monitor, SYDAS, in addition to the GPMS built-in data acquisition capability.

Prior to simulation, the equivalent GPMS microinstructions, into which a model is translated by the aid of MPGS,[7] a software system prepared for description of a model, are loaded into the GPMS microprogram memory, and also sample program and data used for simulation are loaded into the main memory. In this case, data obtained by SYDAS through the current operating system may be used for modeling and as initial data.

While the GPMS has been working as the virtual image of a model, evaluation data, for example, the usage of microinstructions and hardware registers, are obtained by the built-in GPMS acquisition function, according to varying selection conditions.

Simultaneously, SYDAS can obtain more accurate and selective evaluation data without disturbing the GPMS simulation process, depending on GPMS control signals.

Moreover, it is possible for GPMS to control SYDAS positively in order that a wide range of more selective data can be obtained by SYDAS and on the contrary, it is possible for SYDAS to order GPMS so that GPMS ceases to simulate at a certain time and brings directly unaccessible information to the interface.

Furthermore, SYDAS can control GPMS simulation behavior, in detail for some part and in rough for the other.

GPMS

GPMS[15] is a general purpose microprogram controlled hardware simulator which rapidly simulates any kind of computer. It consists of three major modules; a processing module which efficiently simulates a data manipulation of a model, a simulation control module which supervises a simulation process and a data acquisition module which measures hardware resource usages. The block diagram is shown in Figure 1.

Processing module

This module is essentially an arithemetic unit capable of sixteen bit data manipulation for simulation. It contains a 16K-words core memory used for both a main memory and a microprogram memory, a 256 words IC memory, a sixteen bit arithmetic unit and a thirty-two bit shift unit, etc. A microprogram memory is writable so that the model and evaluation parameters are easily changed. A microinstruction is composed of forty bits and contains one bit timer control field, which is called an S field, in addition to various fields for data manipulation and control operation.

Simulation control module

This module supervises the exucution behavior of microsequences in a microprogram memory, into which the control section of a model is translated for simulation by MPGS, and consists of System Timer, Microsequence State Register, User's Timer, and Sequence Control Flip-Flop.

System Timer is the eight decimal digit clock in the model, which is advanced by one when some GPMS microinstructions, equivalent to a basic operation, for example a machine cycle, a microinstruction, or a machine instruction, have been executed. Microsequence State Register holds control words of up to eight microsequences, and is used to control the simulation process in GPMS. User's Timer is composed of eight 32-bit binary counters, which are incremented each time the microin-

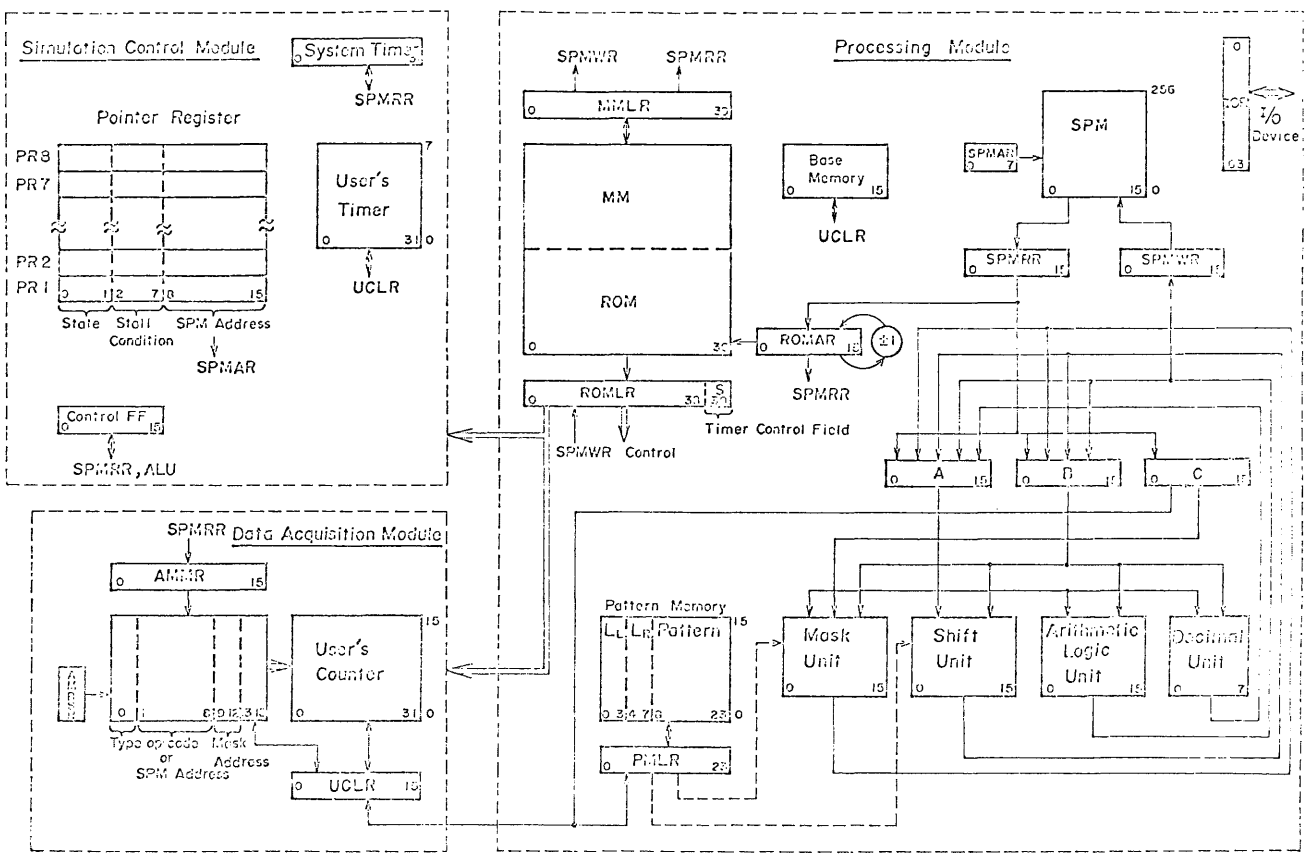


Figure 1—Block diagram of GPMS

struction with the S field set at "1" has been executed. User's Timer can interrupt simulation control after the preset time period, and so may be used to make a pseudo simulation of various interrupt mechanisms. Sequence Control Flip-Flop consists of the usual sixteen bit flip-flops, except that each of them can control the microsequence execution process.

Data acquisition module

A capability to acquire evaluation data is built in the GPMS, and it measures the usage of some hardware resources, such as register, microinstruction and data path. It consists of sixteen 16-bit words associative memory holding mask patterns and sixteen 32-bit binary counters holding the usage.

SYDAS

The SYDAS is a hardware monitor, which processes the system state signals detected by the high-impedance probes attached to the wiring of a computer or specially to the GPMS, measures the activities of the working system without affecting either the hardware or software. The use of the associative memory and associated counters makes it possible to acquire system data in more flexible ways, and to reduce the volume of the data by extracting the data concerned with a specific measurement.

The SYDAS contains the following major hardware components:

- Probes
- Level Converter
- Timer
- Counters (36-bit×2)
- Associative memory (24-bit×128 tag) and associated counters (32-bit×128)
- Buffer memory (8kW − 36-bit)
- Main control
- Magnetic tape units
- Minicomputer (NEAC-M4)

The system configuration of the SYDAS is shown in Figure 2.

System state signals are picked up by high impedance probes, which attach to the backboard pins of a computer or GPMS. Presently, a total of 60 system state signals are selected, as in the following groups.

- Memory address and its access factor signals
- Operation code signals
- I/O channel activity signals
- CPU status signals
- Control signals, etc.

These signals are converted and amplified by the converter and are then driven through a cable to the main control. Depending upon the data required, which are

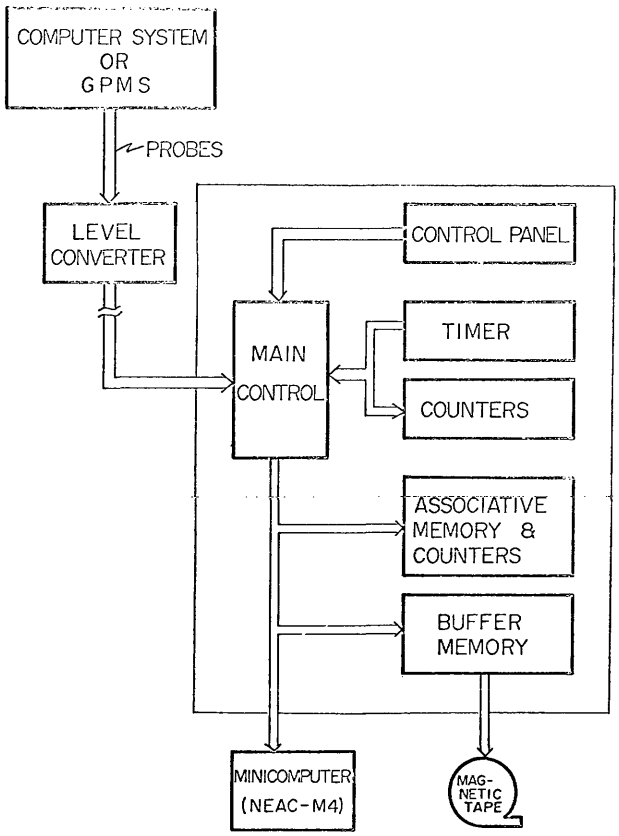


Figure 2—SYDAS system configuration

specified by the switches on the control panel, the main control determines how to manipulate the system state signals. The following four measurement routes are prepared for SYDAS.

Route 1: For measuring the occurrence frequency or time duration of the event, the system state signals which are selected by the control panel are sent to the counters, then, they are measured in the form of time or frequency count. The measurement data are displayed on the numerical display and also transferred to the minicomputer periodically—at intervals of 1 or 10 seconds, as specified by the operation panel.

Route 2: When system events occur, the system state signals are selected by the control panel, and the timer values at that event are recorded in the magnetic tape through the buffer memory.

Route 3: The system signals which correspond to the watched events are stored in the associative memory beforehand. Then the system state signals, sent from a computer system or GPMS, are used to interrogate the associative memory and an event is found from it. An event, extracted in such a way, is counted by each associated counter in the form of the event occurrence frequency. The measurement data are recorded in the magnetic tape periodically (100 ms, 1 second or 10 seconds) or when the measurement was finished.

Route 4: In the same way as for route 3, the system state and the timer value at that event are recorded in the magnetic tape for every extracted event.

It is possible that the various system data can be obtained to rearrange the foregoing four routes with the system state signals.

Data acquisition interface

Together with the GPMS built-in data acquisition capability, timely evaluation data in GPMS can be obtained by SYDAS through the data acquisition interface between GPMS and SYDAS.

There are two kinds of interface, one is a static interface through which SYDAS can acquire fixed, directly accessible data. For example, microsequence address patterns, microsequence state information, main memory address patterns, machine state information and simulation system time information can be obtained independently of GPMS.

The other is a dynamic and universal interface, which is installed for GPMS input/output operation. It is used to acquire data which SYDAS cannot directly access or cannot obtain alone because their meaning has been changing during simulation. In this case, required data with a control field are edited from that by a GPMS microsequence, especially prepared for special data acquisition. An example is the intermediate results obtained by the built-in GPMS data acquisition module.

PROBLEM AREA

Computer systems can be considered to be composed of three layers, as is shown in Figure 3. At the top, there are application programs with which users communicate, then the system software (OS) follows to the first layer, and the processors or input/output devices, which may be divided into the firmware and the hardware, are set at the bottom. The interface between the first and second layer is defined by the OS specification, typically higher-level languages, and the other interface is defined by a set of processor specifications, which is here defined as computer architecture, represented by the machine language.

Most of the architecture problems can be considered as trade-off problems between the second and the third layers, so as to obtain better performance-cost at the state-of-the-art technology. In determining the architecture, at least two designs, which must be carefully investigated, are required. One is the design of the system software, which stands on the architecture, and the other is that of the processors. In the scope of this paper, the latter part of the design area is to be mainly discussed. This does not mean that CODES cannot be applied to evaluating software, but that the current interest in rapid progress of hardware technology emphasizes that part of the design. The major processor design problems can also be recognized as trade-off problems of determining the firmware and the hardware interface. Types of these trade-offs and examples of trades are presented in R. L. Mandell's paper.[8]

The evaluation of these trade-offs with respect to performance-cost is far from trivial, since the performance criteria are not clearly established and the performance measures differ from system to system. The performance measure in this system is primarily defined to be the processing speed of the model in a particular working environment. The hardware cost is estimated by the resources used by the simulation model. Using these values, overall system performance-cost can be obtained, applying mathematical analysis or system level software simulation techniques.

DESIGN AND EVALUATION USING CODES

Firmware/hardware design

In designing the firmware and/or hardware, a designer frequently encounters a problem of trading functions performed in the firmware and hardware, or, more generally, trading performance and cost. He has to determine the following things so as to optimize the design.

- Functional capabilities of hardware blocks, such as adders, counters, special functional blocks, etc.
- Operating speed of those functional blocks.
- Amount of hardware resources
- Connections between hardware resources.

When CODES is used in this type of design, the designer makes a model, which is described in the MPGS language used in CODES. Control part of the model is expressed in a form of microprogram, which is translated into the equivalent GPMS microprogram executable on

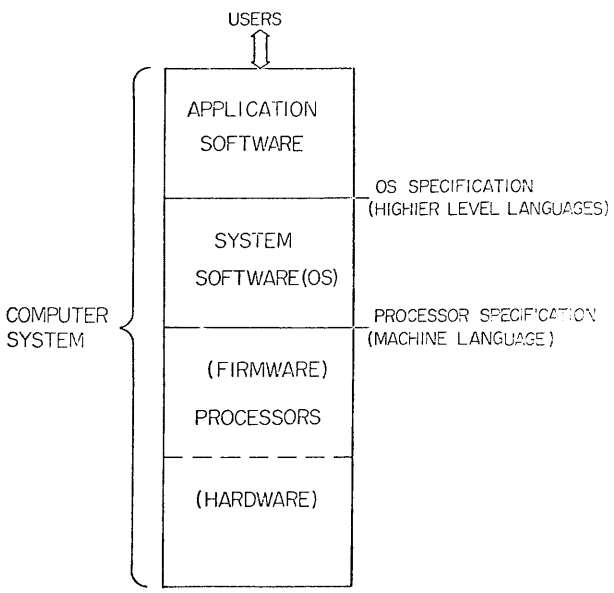


Figure 3—Computer system structure

the CODES. The model description can be either macroscopic or microscopic, depending upon evaluation purposes. At the early design stages, where roughly estimated values are sufficient, a coarse model can work well, and at later stages, the model should be described as precisely as the production model. CODES provides evaluation data by simulating sample programs and data fed into it. The following data can be obtained through this simulation:

- Output of the simulated programs
- Processing time of the simulated model in executing the simulated programs.
- Processing time in the processing modules or microprogram routines
- Usage statistics of the hardware resources

By making use of the above data, the designer can repeatedly change his model to improve the performance-cost figure, to obtain which these simulation results are interpreted according to his criterion. Instead of simulating the whole machine, it is possible to obtain the model performance estimating the module performance figures, that are measured by the CODES simulation and statistic data which are obtained by the currently operating systems. If the machines are of similar architecture and hardware, this approach has great advantages.

Trade from system software to firmware

Extensive use of writable microprogram memory features trading frequently used functions from system software to the firmware, keeping the rest of the instruction repertoire unchanged. The process of performing the trade should be started with observation of the current systems to investigate the effective trade-off point. The CODES measurement facility serves to determine potential trade-off functions by measuring the current systems with respect to the module usage data and the time spent in each module. Next step is to estimate the performance and cost increase or decrease ratio, obtained by the CODES simulation, to embed the selected functions on the firmware of the system. Finally, the trade is determined, taking adverse effects into account, such as reprogramming the system software to accomodate the trade. Should convenient linkage convention of calling microprogram routines from machine language programs be provided, this type of trade-off may be widely accepted.

When a large amount of reprogramming is acceptable another approach[13] can be taken, as is the case described in the next section.

Higher level language processor

Implementation of higher level language (HLL) processors on the firmware/hardware is a good example of major trade-off problems, and experiments of implementing many kinds of HLL processors, such as ALGOL,[1] SYMBOL,[3] FORTRAN,[9] EULER,[12] APL,[16], etc., have been reported. However, the cost-effectiveness of firmware HLL processors relative to conventional software implementation is not clearly established. The important thing is to define the evaluation measure. An approach, which is being carried out using CODES, is comparative evaluation.

At the first stage, relative performance improvement is to be measured, by implementing the interpretation phase of the HLL compiler on the firmware. The interpretation method and the functional definition of the firmware implementation should be as close as those used in the conventional compiler, in order to fairly compare the two processors. Comparison is made between the simulation result of the firmware HLL processor and the measured data of the conventional one.

After this comparison is made the further evaluation data of firmware-hardware trade-off, which can be evaluated keeping the architecture fixed, or the evaluation data of software-firmware/hardware trade-off can be translated into the relative performance-cost to the conventional HLL processor.

CONCLUSION

The CODES system is a powerful evaluation tool for architecture and firmware/hardware design, especially, it is useful to determine trade-offs, based on quantitative evaluation data obtained by CODES simulation. For example, as LSI technology advances, the trade-off between firmware and hardware becomes of importance. CODES simulation greatly helps to determine what kind of LSI should be made. Defining better architecture is also important, in relation to hardware and software progress. Since the architecture stands on the state-of-the-art balance of hardware and software, it is affected by the changes in either part. CODES can be of use to determine architecture.

Fast simulation speed have attained in CODES. The CODES simulation speed depends on the model complexity. Approximate speed ratio of models and the simulator is 20-30 to 1 in a small scale model simulation, and 50-100 to 1 in a medium scale model. This fast simulation speed makes it possible to evaluate on various parameters and, to some extent, to actually use CODES as a special computer. The measurement facility embedded in CODES provides plenty of meaningful data for analysis, and supporting software helps to make or modify simulation models.

However, CODES is not a completely satisfactory system for applying it to various design and evaluation problems. One of the deficiencies of CODES is that it takes a lot of time to make a large precise model. More design aid softwares should be developed to use CODES more effectively. It is being planned to connect CODES to a conventional computer system on line, facilitating to rapidly change model description or to retrieve models in the files. As the course of the CODES usage experiences many design problems, more CODES softwares will be developed to effectively use the system.

ACKNOWLEDGMENT

The authors would like to express their grateful thanks to Dr. Y. Kotaka, Mr. K. Kiji and Mr. K. Kagiyama for their encouragement, Messrs. K. Fujino, M. Hattori and Mrs. M. Yano, who have developed MPGS, and Mr. S. Fushimi who has designed a large part of CODES hardware.

REFERENCES

1. Anderson, J. P., "A Computer for Direct Execution of Algorithmic Languages," *AFIPS Eastern JCC*, Vol. 20, pp. 184-193, 1961.
2. Barsamian, H., DeCegama, A., "Evaluation of Hardware-Firmware-Software Trade-offs with Mathematical Modeling," *SJCC*, Vol. 38, pp 151-161, 1971.
3. Chesley, G. D., Smith, W. R., "The Hardware-Implemented High-Level Machine Language for SYMBOL", *SJCC*, Vol. 38, pp. 563-573, 1971.
4. Cook, R. W., Flynn, M. J., "System Design of a Dynamic Microprocessor," *IEEE*, Vol. C-19, No. 3, March 1970.
5. Flynn, M. J., "Some Organizations and Their Effectiveness" *IEEE* Vol. C-21, No. 9, pp. 948-960, September 1972.
6. Foley, J. D., "An Approach to the Optimum Design of Computer Graphics Systems," *CACM*, Vol. 14, No. 6, pp. 380-290, June 1971.
7. Hattori, M., et al., "A Highlevel Language for Microprogram Generating System," *ACM*, 1972, National Conference.
8. Mandell, R. L., "Hardware-Software Trade-Offs—Reasons and Directions" *FJCC*, Vol. 41, pp 453-460, 1972.
9. Melbourne, A. J., Pugmire, J. M., "A Small Computer for the Direct Processing of FORTRAN Statements, *Computer Journal*, Vol. 8, pp. 24-28, April 1965.
10. Opler, A., "Fourth-Generation Software," *Datamation*, January 1967, pp 22-24.
11. Reigel, E. W., Faber, U., Fisher, D. A., "The Interpreter—A Microprogrammable Building Block System," *SJCC*, Vol. 40, pp 705-723, 1972.
12. Weber, H., "A Microprogrammed Implementation of EULER on IBM System/360 Model 30" *CACM*, Vol. 10, No. 9, pp 549-558, September, 1967.
13. Werkheiser, A. H., *Microprogramming the Operating System*, 3rd Annual Workshop on Microprogramming, October 1970.
14. Wilner, W. T., "Design of the Burroughs B1700," *FJCC*, Vol. 41, pp 489-497, 1972.
16. Zaks, R., Steingart, D., Moore, J., "A Firmware APL Time-sharing System," *SJCC*, Vol. 38, pp 179-190, 1971.
17. Zucker, M. S., "LOCS: An EDP Machine Logic and Control Simulator," *IEEE*, Vol. EC-14, No. 6, pp. 403-416, June 1965.

An analysis of multiprogrammed time-sharing computer systems

by M. A. SENCER and C. L. SHENG

The University of Ottawa
Ottawa, Canada

INTRODUCTION

The size of the investment required and the complexity of the time-sharing[1–3] computer systems, TSCS, necessitate a good deal of effort to be spent on the analysis of the resource allocation problems which are obviously tied to the cost and the congestion properties of the system configuration. In this paper we study the congestion problems in the multiprogramming[1–3] TSCS's.

The activity in the past sixteen years in the analysis of the congestion properties of the TSCS's by purely analytical means has been concentrated on the study of the central processing unit (or units), CPU('s), and the related queues. A good survey of the subject has been provided by McKinney.[4] In the meantime more contributions to this area have appeared in the literature.[5–11]

There has been a separate but not so active interest in the study of the congestion problems in the multiprogramming systems. Heller[12] and Manacher[13] employed Gantt charts (cf. 12) in the scheduling of the sequential "job lists" over different processors. Ramamoorthy et al.[14] considered the same problem with a different approach. A single server priority queueing model was applied by Chang et al.[15] to analyze multiprogramming. Kleinrock[16] used a 2-stage (each stage with a single exponential server) "cyclic queueing"[17] model to study the multistage sequential servers. Later Tanaka[18] extended the 2-stage cyclic queueing model to include the Erlang distribution in one of the stages. Gaver[19] extended the 2-stage cyclic queueing model of multiprogramming computer systems further by including an arbitrary number of identical processors with exponential service times in the input-output, IO, stage in providing the busy period analysis for the CPU under various processing distributions.

Here we present a multiprogramming model for a general purpose TSCS, where the arrival and the departure processes of the programs along with the main system resources, i.e., the CPU's, the different kind of IOP's and the finite main memory size, are included and their relationships are examined.

MODEL

The configuration of the TSCS we want to study consists of $k_0 \geq 1$ identical CPU's with m groups of IOP's where the ith group has $k_i \geq 1$, $(i = 1, 2, \ldots, m)$ identical processors, as shown in Fig. 1.

The new programs are assumed to arrive to the ith IOP group in the form of a time-homogeneous Poisson process with an average arrival rate λ_i, $(i = 1, 2, \ldots, m)$. If there is less than N programs in the system a new arrival is accepted into the system and is immediately placed in the common queue before the particular IOP group. If the number of programs in the system is N, then any new arrival is assumed to leave the system and never to return. These assumptions are justified where there is practically a large number of program originating sources and these attempt to access the system, each time, independent of each other and the system status, otherwise they represent approximations.

Here we consider that each program currently in the system has one unit of space (or a "page") reserved for it in the main memory. This assumption increases the efficiency of the time-sharing policy by making more programs available to the CPU's at low system administration complexity, but at the same time it may increase the IO activity between the main and the auxiliary memories. In this paper we do not attempt to study this trade-off.

A program (or a piece of it) after being processed, in the ith IOP group according to some service discipline (say first-come-first-served), either proceeds to queue for CPU processing or departs from the system with the probabilities p_i and $1-p_i$, respectively.

The programs demanding CPU processing are served by k_0 CPU's in the order of their priorities, in time-sharing policy. Thus an idle CPU takes the program with the highest priority among the waiting ones in the common CPU-queue for processing. Each program is allocated a "time-scale" of processing, the length of which depends on the priority level of the program, each time it is scheduled to be processed by a CPU. A program which needs further CPU processing after one full time-slice is rescheduled in the CPU queue with an appropriate priority level. Some programs may not complete the full time-slice of processing due to various reasons, such as termination, need for data transfer to and from the auxiliary memory or the programmer, etc. In that case the program leaves the CPU to join the queue in the ith IOP group to be processed, with the probability q_i, where $\sum_{i=1}^{m} q_i = 1$. An interrupt from a higher priority program may also cause a currently running program to stop. Here we admit only

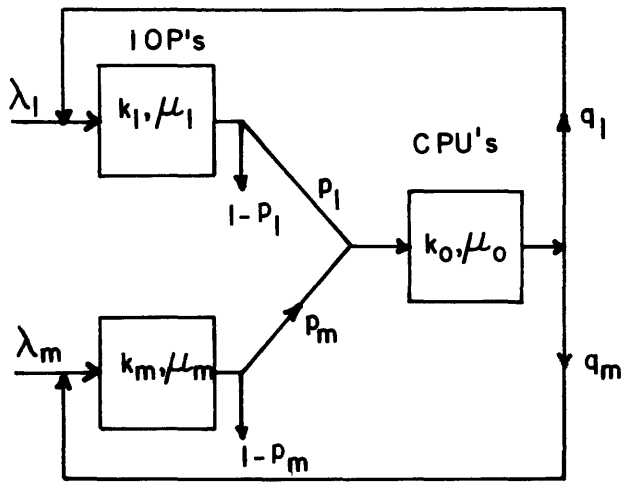


Figure 1—System configuration

preemptive-resume type of interrupt policy where the interrupted program sequence continues from the point where it was stopped when its turn comes. All the IO and the CPU queues are conceptual and assumed to be formed in the main memory.

In this model the processing times of all the CPU's and the IOP's are assumed to be independent and distributed negative exponentially. The processors in the ith group all have the same mean service time $1/\mu_i$, where $i=0, 1, \ldots, m$. The transition probabilities between the IOP groups and the CPU group are assumed to be stationary. These assumptions have already been used successfully by other authors investigating the TSCS in one form or other (cf. 4 and other references on TSCS).

MATHEMATICAL ANALYSIS

In this section we present a mathematical analysis of the model described in the previous section, by an extension of the method presented by Jackson.[20]

Let us denote by Λ_i, $i=0, 1, \ldots, m$, the average arrival rate (both internal and external) of programs to the ith group of processors where the oth group represents the CPU group and the groups 1 to m correspond to the IOP groups. Then

$$\Lambda_i=\lambda_i+\sum_{j=0}^{m} r_{ji}\Lambda_j, \qquad i=0, 1, \ldots, m \tag{1}$$

where $\lambda_0=0$

and $r_{0i}=q_i$ for $i=1, 2, \ldots, m$

$r_{j0}=p_j$ for $j=1, 2, \ldots, m$

$r_{ji}=0$ otherwise.

Let $P(n_0, n_1, \ldots, n_m; t)$ denote the probability of the state of the system with n_i programs in group i (being served or waiting) at time t, $(i=0, 1, \ldots, m)$. Then we can write the system equation, following Feller,[21] by relating the state of the system at time $t+h$ to that at time t, as follows:

$$\begin{aligned}P(n_0, n_1, \ldots, n_m; t+h) &= \{1-[C\sum_{i=1}^{m}\lambda_i+\sum_{j=0}^{m}\mu_j\alpha_j(n_j)]h\}\\ &\times P(n_0, \ldots, n_m; t)\\ &+\sum_{i=1}^{m}\lambda_i\epsilon_i h P(n_0, \ldots, n_i-1, \ldots, n_m; t)\\ &+\sum_{j=1}^{m}\alpha_0(n_0+1)\mu_0 r_{0j}\epsilon_j h P(n_0+1, \ldots, n_j-1, \ldots, n_m; t)\\ &+\sum_{i=1}^{m}\alpha_i(n_i+1)\mu_i r_{i0}\epsilon_0 h P(n_0-1, \ldots, n_i+1, \ldots, n_m; t)\\ &+C\sum_{i=1}^{m}\alpha_i(n_i+1)\mu_i(1-r_{i0})h\\ &\times P(n_0, \ldots, n_i+1, \ldots, n_m; t)+0(h).\end{aligned} \tag{2}$$

where $C=1$ if $\sum_{i=0}^{m} n_i<N$

$=0$ if $\sum_{i=0}^{m} n_i\geq N$,

$\alpha_i(n)=\min\{n, k_i\}, \quad i=0, 1, \ldots, m$

$\epsilon_i=\min\{n_i, 1\}, \quad i=0, 1, \ldots, m.$

By the usual process of forming the derivative on the left hand side and letting it equal to zero for steady-state and also replacing $P(n_0, n_1, \ldots, n_m; t)$ by $P(n_0, n_1, \ldots, n_m)$, we have

$$\begin{aligned}0 &= -[C\sum_{i=1}^{m}\lambda_i+\sum_{j=1}^{m}\mu_j\alpha_j(n_j)]P(n_0, \ldots, n_m)\\ &+\sum_{i=1}^{m}\lambda_i\epsilon_i P(n_0, \ldots, n_i-1, \ldots, n_m).\\ &+\sum_{i=1}^{m}\alpha_0(n_0+1)\mu_0 q_i\epsilon_i P(n_0+1, \ldots, n_i-1, \ldots, n_m)\\ &+\sum_{i=1}^{m}\alpha_i(n_i+1)\mu_i p_i\epsilon_i P(n_0-1, \ldots, n_i+1, \ldots, n_m)\\ &+C\sum_{i=1}^{m}\alpha_i(n_i+1)\mu_i(1-p_i)P(n_0, \ldots, n_i+1, \ldots, n_m).\end{aligned} \tag{3}$$

Here as we deal with Markov processes of finite states, mutually inter-communicating, the normalized solution of (3) is a proper and unique probability distribution.[22] Then it can be shown by substitution, as in Reference 20, that the solution to (3) is given by the following theorem which is different from the previously proved one[20] in that an arbitrary number, N, is permitted in the system.

Theorem: Define $P_i(n)$, for $(i=0, 1, \ldots, m)$ and $(n=0, 1, \ldots, N)$ as the probability of n programs in the processor group i in the above described system in steady state. Then

$$P_i(n)=\begin{cases}P_i(o)\,(\Lambda_i/\mu_i)^n/n! & \text{for } n=0, 1, \ldots, k_i \\ P_i(o)\,(\Lambda_i/\mu_i)^n/k_i!k_i^{n-k_i} & \text{for } n=k_i, k_i+1, \ldots, N.\end{cases}$$

Also

$$P(n_0, n_1, \ldots, n_m)=P_0(n_0)\cdot P_1(n_1)\ldots P_m(n_m)$$

where

$P_i(o)$ is determined from the normalizing condition

$$\sum_{n=0}^{N} P_i(n)=1.$$

In the Appendix the theorem given above is extended to cover general m-stage network as in Reference 20.

ARRIVAL RATES AND SOME SYSTEM MEASURES

The theorem stated in the previous section suggests that at the steady state our model can be decomposed into independent stages (each stage corresponding to a group of processors) of similar configuration (with different parameters) where each stage can be analyzed in the same way.

First we determine the average arrival rate, Λ_i, for each stage in terms of the given system parameters. Writing $R=[r_{ij}]$, $\Lambda=[\Lambda_i]$ and $\lambda=[\lambda_i]$ where R is $a(m+1)\times(m+1)$ square matrix and Λ and λ are column vectors, we can put (1) in matrix form, as

$$\begin{bmatrix} 1 & -p_1 & -p_2 & \ldots\ldots & -p_m \\ -q_1 & 1 & 0 & \ldots\ldots & 0 \\ -q_2 & 0 & 1 & 0\ldots\ldots & 0 \\ \vdots & & & & \\ -q_m & 0 & \ldots\ldots & \ldots\ldots & 1 \end{bmatrix}\begin{bmatrix}\Lambda_0\\ \Lambda_1\\ \vdots\\ \Lambda_m\end{bmatrix}=\begin{bmatrix}0\\ \lambda_1\\ \lambda_2\\ \vdots\\ \lambda_m\end{bmatrix} \tag{4}$$

or

$$R\Lambda=\lambda.$$

When $|R|$ is not identically zero (4) can be uniquely solved for Λ_i's and it can be shown that the solution is given by

$$\Lambda_0=\frac{\sum_{j=1}^{m} p_jq_j}{1-\sum_{j=1}^{m} p_jq_j}$$

$$\Lambda_i=\frac{\sum_{j=1,j\neq i}^{m}\lambda_jp_jq_j+(1-\sum_{j=1,j\neq i}^{m}p_jq_j)\lambda_i}{1-\sum_{j=1}^{m}p_jq_j} \tag{5}$$

for $i=1, 2, \ldots, m$.

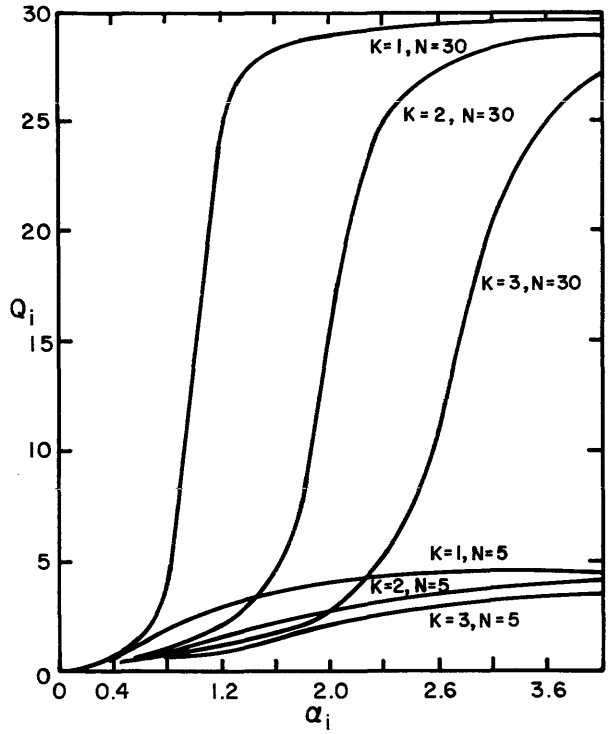


Figure 2—The average number of programs, Q_i, in stage i

The first system measure to be considered is the average number of programs, Q_i, at stage i, where $i=0, 1, \ldots, m$.

Let us define $$\rho_i=\frac{\Lambda_i}{k_i\mu_i}$$

and $$\alpha_i=k_i\rho_i$$

From the normalizing condition we obtain for the emptiness probability, $P_i(o)$, for stage i as,

$$P_i(o)=\left[\sum_{n=0}^{k_i-1}\frac{\alpha_i^n}{n!}+\frac{\alpha_i^{k_i}(1-\rho_i^{N-k_i+1})}{k_i!(1-\rho_i)}\right]^{-1}. \tag{6}$$

Then Q_i can be computed from

$$Q_i=\sum_{n=0}^{N} nP_i(n)$$

$$=P_i(o)\left[\sum_{n=1}^{k_i}\frac{n\alpha_i^n}{n!}+\frac{k_i^{k_i}}{k_i!}\sum_{n=k_i+1}^{N} n\rho_i^n\right]. \tag{7}$$

The average number of programs at the stage i, Q_i, has been plotted as a function of α_i with parameters k_i and N in Figure 2.

Next we consider the average number of busy processors at stage i, Q_{ib}, which is given by

$$Q_{ib}=\sum_{n=0}^{k_i-1} nP_i(n)+k_i\sum_{n=k_i}^{N}P_i(n)$$

$$=P_i(o)\left[\sum_{n=1}^{k_i-1}\frac{n\alpha_i^n}{n!}+\frac{\alpha^{k_i}(1-\rho^{N-k_i+1})}{(k_i-1)!(1-\rho)}\right] \tag{8}$$

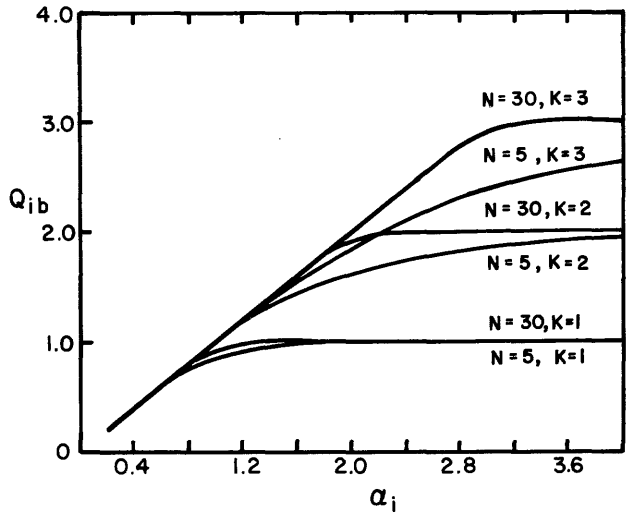


Figure 3--The average number of busy processors, Q_{ib} in stage i

The Q_{ib} has been plotted as a function of α_i with parameters k_i and N in Figure 3.

The average flow time of a program through the system can be obtained from Little's result,[23]

$$Q = \lambda_T W$$

where Q = the average number of programs in the system
λ_T = total mean arrival rate of programs to the system
W = average flow time of a program

and $$Q = \sum_{i=0}^{m} Q_i, \qquad \lambda_T = \sum_{i=1}^{m} \lambda_i.$$

CONCLUSION

The analysis presented in this paper provides us with a practical tool to examine the effects of main system resources in a TSCS quantitatively. It is possible to extend these results in the area of system balancing by taking the cost-revenue considerations into account.

In the systems where the assumptions made in this study are not justified, at least approximately, this type of analysis may serve as a preliminary to more refined techniques or simulation, if ever waranted.

The results are applicable to a wide range of problems where the total common storage space is limited and an output from a stage either departs from the system or forms an input to another stage.

ACKNOWLEDGMENTS

The research was supported in part by the National Research Council of Canada under Grant No. A-1690.

REFERENCES

1. Bull, G. M., Packham, S. F. G., *Time-Sharing Systems*, McGraw-Hill, London, 1971.
2. Pyke, N. T. Jr., "Time-Shared Computer Systems," in *Advances in Computers*, Vol. 8, pp. 1-45, Academic Press, 1968.
3. Watson, R. W., *Time-Sharing System Design Concepts*, McGraw-Hill, 1970.
4. McKinney, J. M., "A Survey of Analytical Time-Sharing Models," *Computing Surveys*, Vol. 1, June 1969, pp. 106-116.
5. Kleinrock, L., "Certain Analytical Results for Time Shared Processors," *Info. Processing 68*, North Holland Pub. Co. Amsterdam, 1969.
6. Hellerman, H., "Some Principles of Time-Sharing Schedular Strategies," *IBM Systems Journal*, Vol. 8, No. 2, 1969, pp. 94-117.
7. Rasch, P. J., "A Queueing Theory Study of RR Scheduling of Time-Shared Computer Systems," *J.ACM*, Vol. 17, Jan. 1970, pp. 131-145.
8. Coffman, E. G., Muntz, R. R., Trotter, H., "Waiting Time Distribution for Processor Sharing Systems," *J.ACM*, Vol. 17, Jan. 1970, pp. 123-130.
9. Sakata, M., Noguchi, S., Oizumi, J., "An Analysis of the M/G/1 Queue Under Round-Robin Scheduling," *Opns. Res.* Vol. 19, March-April 1971, pp. 371-385.
10. Bhat, N., Nance, R. E., "Busy Period Analysis of a Time Sharing System Modelled as a Semi-Markov Process," *J.ACM*, Vol. 18, April 1971, pp. 221-238.
11. Adiri, I., "A Dynamic Time Sharing Priority Queue," *J.ACM*, Vol. 18, Oct. 1971, pp. 603-610.
12. Heller, J., "Sequencing Aspects of Multiprogramming," *J.ACM*, Vol. 8, 1961, pp. 426-439.
13. Manacher, G. K., "Production Stabilization of Real Time Task Schedules," *J. ACM*, Vol. 14, July 1967, pp. 439-465.
14. Ramamoorthy, C. V., Cahndy, K. M., Gonzalez, M. J., "Optimal Scheduling Strategies in Multiprocessor Systems," *IEEE Trans. on Comp.*, Feb. 1972, pp. 137-146.
15. Chang, W., Wong, D. J., "Analysis of Real Time Multiprogramming," *J.ACM*, Vol. 12, Oct. 1965, pp. 581-588.
16. Kleinrock, L., "Sequential Processing Machines Analyzed with Queueing Theory Model," *J.ACM*, Vol. 13, April 1966, pp. 179-193.
17. Koenigsberg, E., "The Cyclic Queue," *Opns. Res. Quarterly*, Vol. 9, March 1958, pp. 22-35.
18. Tanaka, H., "An Analysis of Multiprogramming System Using a Cyclic Queueing Model," *Systems-Computers-Control*, Vol. 1, No. 1, 1970, pp. 37-45.
19. Gaver, D. P., "Probability Models for Multiprogramming Computer Systems," *J. ACM*, Vol. 14, July 1967, pp. 423-438.
20. Jackson, J. R., "Networks of Waiting Lines," *Opns. Res.*, Vol. 5, August 1957, pp. 518-521.
21. Feller, W., *Introduction to Probability Theory and Its Applications*, Vol. 1, John Wiley, 1968, Ch. XVII.
22. Cox, D. R., Miller, H. D., *Theory of Stochastic Processes*, Methuen and Co. Ltd., 1965, pp. 183-185.
23. Little, J. D. C., "A Proof of the Queueing Formula $L = \lambda W$," *Opns. Res.*, Vol. 9, May-June 1961, pp. 383-387.

APPENDIX

The general "network of waiting lines" (network of queues) was described in Reference 20. Here we add one more condition to that stated in Reference 20 by allowing no more than N customers in the system at any time. If a new customer arrives to find N customers already in the system, he departs and never returns.

Then following the previously defined notation in earlier sections, we can write the system equation for general network of waiting lines as

$$
\begin{aligned}
&P(n_1, n_2, \ldots, n_m; t+h) \\
&\quad = \{1-[C \sum_{i=1}^{m} \lambda_i + \sum_{j=1}^{m} \mu_j \alpha_j(n_j)]h\} P(n_1, \ldots, n_m; t) \\
&\quad + \sum_{i=1}^{m} \lambda_i \in {}_i h P(n_1, \ldots, n_i-1, \ldots, n_m; t) + \sum_{i=1}^{m} \sum_{j=1}^{m} \qquad (10) \\
&\quad \times \alpha_i(n_i+1)\mu_i r_{ij} \in {}_i h P(n_1, \ldots, n_i+1, \ldots, n_j-1, \ldots n_m; t) \\
&\quad + C \sum_{i=1}^{m} \alpha_i(n_i+1)\mu_i(1-r_i) h P(n_1, \ldots, n_i+1, \ldots, n_m; t) \\
&\quad + 0(h).
\end{aligned}
$$

where

$$r_i = \sum_{j=1}^{m} r_{ij}.$$

It can be shown that the solution to (10) in steady state is given by the theorem stated earlier.

In this case it is difficult to obtain a closed form solution for $\wedge_i$'s.

Use of the SPASM software monitor to evaluate the performance of the Burroughs B6700

by JACK M. SCHWARTZ and DONALD S. WYNER

Federal Reserve Bank of New York
New York, New York

INTRODUCTION

The need for system performance measurement and evaluation

The benefit to be derived from a large multi-purpose system, such as the B6700, is that many jobs of very diverse characteristics can (or should) be processed concurrently in a reasonable period of time. Recognizing that certain inefficiencies may result from improper or uncontrolled use, it is necessary to evaluate the computer system carefully to assure satisfactory performance. To this end, the objective of our work in the area of performance evaluation is to:

1. determine the location(s) and cause(s) of inefficiencies and bottlenecks which degrade system performance to recommend steps to minimize their effects,
2. establish a profile of the demand(s) placed upon system resources by programs at our facility to help predict the course of system expansion,
3. determine which user program routines are using inordinately large portions of system resources to recommend optimization of those routines,
4. establish control over the use of system resources.

Among the techniques which have been applied to date in meeting these objectives are in-house developed software monitors, benchmarking, and in-house developed simulations. This paper discusses the software monitor, SPASM (System Performance and Activity Software Monitor), developed at the Federal Reserve Bank of New York to evaluate the performance and utilization of its Burroughs B6700 system.

THE B6700 SYSTEM

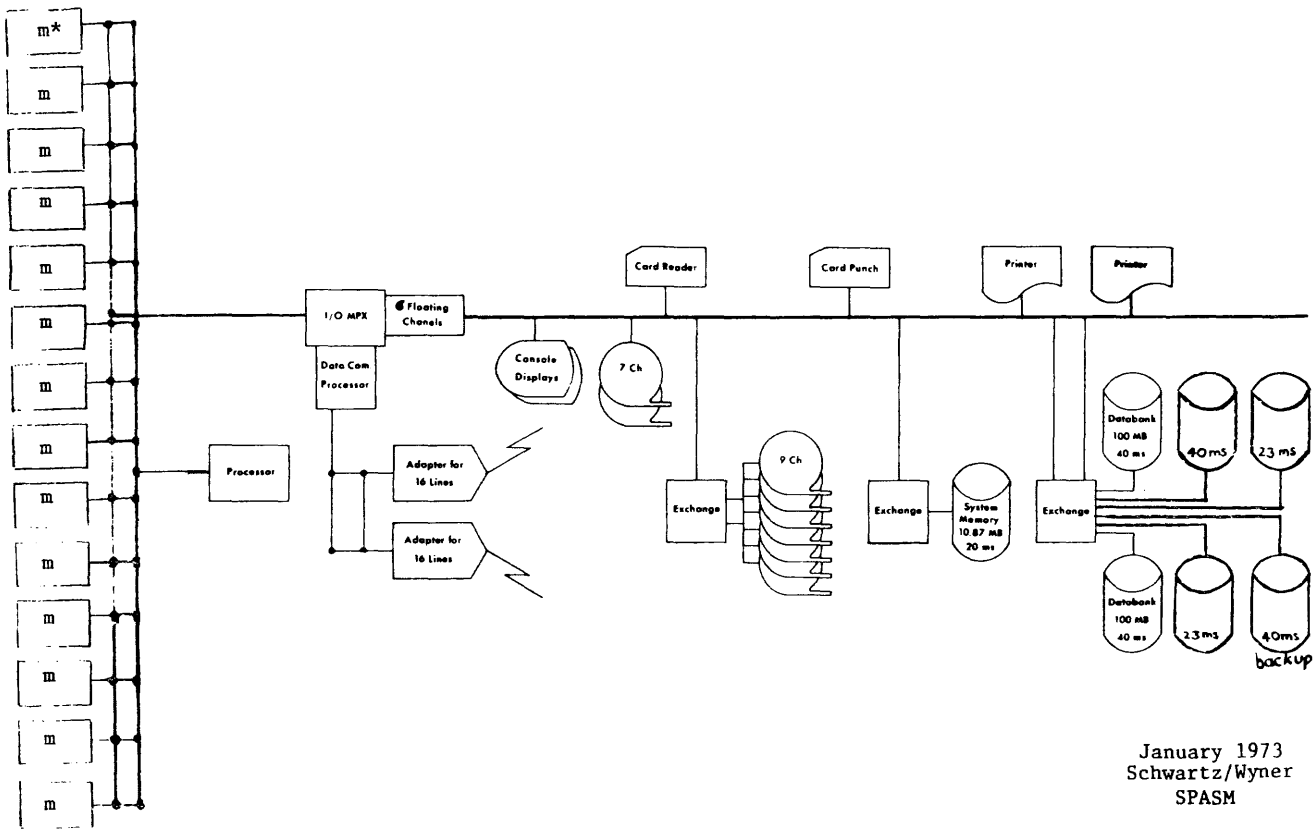
The B6700 is a large-scale multiprogramming computer system capable of operating in a multiprocessing mode which is supervised by a comprehensive software system called the Master Control Program (MCP).[1,2] Some of the features of this system which distinguish it from many other systems are:

- Each task has assigned to it a non-overlayable area of memory called a stack. This area provides storage for program code and data references* associated with the task as well as temporary storage for some data, history and accounting information.
- Multiple users can share common program code via a reentrant programming feature.
- The compilers automatically divide source language programs into variable sized program code and data segments rather than fixed sized pages.
- Core storage is a virtual resource which is allocated as needed during program execution. (This feature is discussed in more detail below.)
- Secondary storage including magnetic tape and head-per-track disk is also allocated dynamically by the MCP.
- Channel assignments are made dynamically; that is they are assigned when requested for each physical I/O operation.
- I/O units are also assigned dynamically.
- Extensive interrupt facilities initiate specific MCP routines to handle the cause of the interrupt.
- The maximum possible B6700 configuration includes 3 processors, 3 multiplexors, 256 peripheral devices, 1 million words of memory (six 8-bit characters per word or 48 information bits per word), and 12 data communications processors.

The current B6700 system at the Federal Reserve Bank of New York shown in Figure 1 includes one processor, one I/O multiplexor with 6 data channels, one data communications processor and a number of peripheral devices. In addition, the system includes a virtual memory consisting of 230,000 words of 1.2 micro-second memory, and 85 million words of head per track disk storage.

The management of this virtual memory serves to illustrate the involvement of the MCP in dynamic resource

* These references are called descriptors and act as pointers to the actual location of the code or data



* 14 memory modules, each having 98.3 KB; total 1.4 MB.

Figure 1—Configuration of the Federal Reserve Bank of New York B6700 computer system

allocation. This process is diagrammed in Figure 2. Main memory is allocated by the MCP as a resource to current processes. When a program requires additional memory for a segment of code or data, an unused area of sufficient size is sought by the MCP. If it fails to locate a large enough unused area, it looks for an already allocated area which may be overlaid. If necessary, it links together adjacent available and in-use areas in an attempt to create an area large enough for the current demand. When the area is found, the desired segment is read in from disk and the segments currently occupying this area are either relocated elsewhere in core (if space is available), swapped out to disk or simply marked not present. In any case, the appropriate descriptor must be modified to keep track of the address in memory or on disk of all segments involved in the swap. All of these operations are carried out by the MCP; monitoring allows us to understand them better. For additional information on the operation and structure of the B6700 see Reference 3.

B6700 PERFORMANCE STATISTICS

The complexity of the B6700 system provides both the necessity to monitor and the ability to monitor. The pervasive nature of the MCP in controlling the jobs in the system and in allocating system resources made it necessary for the system designers to reserve areas of core memory and specific cells in the program stacks to keep data on system and program status. This design enables us to access and collect data on the following system parameters:

- system core memory utilization
- I/O unit utilization
- I/O queue lengths
- processor utilization
- multiplexor utilization
- multiplexor queue length
- peripheral controller utilization
- system overlay activity
- program overlay activity
- program core memory utilization
- program processor utilization
- program I/O utilization
- program status
- scheduler queue length
- response time to non-trivial requests

These data are vital to the evaluation of our computer system. Table I presents examples of the possible uses for some of these statistics.

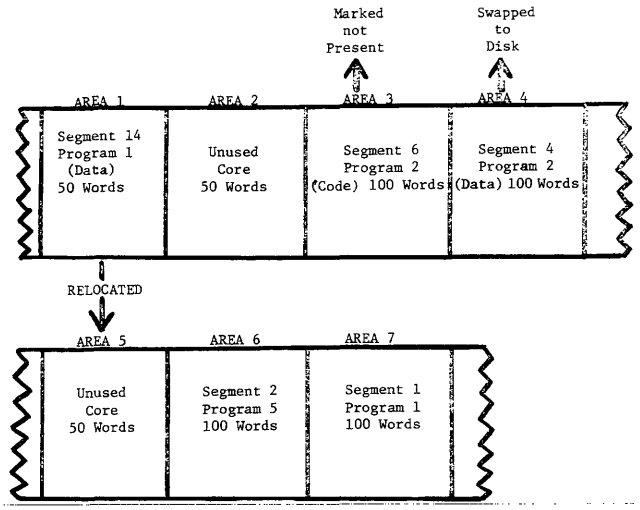


Figure 2—B6700 memory allocation procedure

1. Space is needed for a 300 word segment for one of the current tasks.
2. A large enough unused area is not located.
3. The MCP finds a contiguous location made up of areas 1 through 4 which is 300 words long.
4. Area 1 contains a 50 word data segment. The MCP relocates this segment into area 5, makes note of its new core address and removes area 5 from the unused linkage.
5. Area 2 is unused. It is removed from the unused linkage.
6. Area 3 contains a 100 word code segment. There are no unused areas large enough to contain it. Therefore, it is simply marked not present. Since code cannot be modified during execution, there is no reason to write it out to disk—it is already there.
7. Area 4 contains a 100 word data segment. It is written out to disk and its new location is recorded.
8. The 300 word segment is read into core in the area formerly occupied by areas 1 through 4 and its location is recorded.

TABLE I—Examples of Collected Statistics and Their Possible Uses

Data	Use
System core memory utilization	Determine need for additional memory
I/O unit utilization, I/O unit queue lengths	Determine need for Disk File Optimizer and/or additional disk storage electronic units, printers or disk file controllers
Processor queue length and composition	Determine need for additional processor
	Evaluate effect of job priority on execution
	Determine processor boundedness of mix
	Determine effect of processor utilization on demand for I/O (in conjunction with I/O unit data)
System overlay activity	Determine need for additional memory
	Determine need for better task scheduling
	Determine when thrashing* occurs
Job overlay activity	Evaluate program efficiency
	Evaluate system effect on job execution
Job core memory utilization	Evaluate program efficiency
	Change job core estimates
Scheduler queue length	Determine excess demand for use of system
	Evaluate MCP scheduling algorithm

* Thrashing is the drastic increase in overhead I/O time caused by the frequent and repeated swapping of program code and data segments. It is caused by having insufficient memory to meet the current memory demand.

DESCRIPTION OF THE SPASM SYSTEM

The B6700 System Performance and Activity Software Monitor, SPASM, is designed to monitor the performance of the system as a whole as well as that of individual user programs. It consists of two separate programs, a monitor and an analyzer, both of which are described below. The principal criteria governing its design are:

(a) to make a software monitor capable of gathering all the pertinent data discussed in the previous section,
(b) to minimize the additional load placed upon the system by the monitor itself, and
(c) to provide an easily used means of summarizing and presenting the data gathered by the monitor in a form suitable for evaluation by technical personnel and management.

Ability to gather pertinent data

The Mastor Control Program concept of the B6700 helps in many ways to simplify the acquisition of the data listed in Table I. Such information as a program's core usage, processor and I/O time, and usage of overlay areas on disk are automatically maintained in that program's stack by the MCP. A relatively simple modification to the MCP permits a count of overlays performed for a program to be maintained in its stack. Data describing the status of programs are maintained by the MCP in arrays.

Information on system-wide performance and activity is similarly maintained in reserved cells of the MCP's stack. Pointers to the head of the processor queue, I/O queues and scheduler queue permit the monitor to link through the queues to count entries and determine facts about their nature. Other cells contain data on the system-wide core usage, overlay activity, and the utilization of the I/O multiplexor. An array is used to store the status of all peripheral devices (exclusive of remote terminals) and may be interrogated to determine this information.

All of the above data are gathered by an independently running monitor program. The program, developed with the use of a specially modified version of the Burroughs ALGOL compiler, is able to access all information maintained by the MCP. The program samples this information periodically and stores the sampled data on a disk file for later reduction and analysis.

Minimization of load upon the system

To minimize the additional load on the B6700, the monitor program is relatively simple, and very efficient. A somewhat more sophisticated analyzer program is used to read back the raw data gathered by the monitor and massage it into presentable form. This analysis is generally carried out at a time when its additional load upon the system will be negligible. The system log has indicated that the monitor does indeed present a minimal load requiring about 1/4 of 1 percent processor utilization and 2 1/4 percent utilization of one disk I/O channel.

Easy means of analysis and presentation

The raw data gathered by the monitor can be used directly in some cases; however, to serve best the purpose for which SPASM was designed (i.e., as a management reporting system) several useful presentations have been engineered. The analyzer program, which may be run interactively or in batch mode will produce any of the following:

- Graphs of data versus time
- Frequency distribution histograms of data
- Correlation and regression analyses among data
- Scanning for peak periods

The options are selected via an input language consisting of mnemonics.

(1) *Graphs of data versus time* are produced to show the minute-by-minute variations in parameters of interest. The graphs are "drawn" on the line printer using symbols to represent each curve. Any number of parameters may be plotted on one graph, and a key is printed at the end identifying all symbols used in the graph and listing the mean values of each. To aid in tailoring the most desirable presentation, rescaling of the ordinate and time axes is permitted. In addition, the user may select a specific time interval of interest and plot that interval only. Fig-

Figure 3—Graph of core usage versus time

Figure 4—Histogram of "overlayable" core usage

ure 3 presents a graph of core utilization versus time. Lines may be drawn connecting the points to aid readability. On this graph three parameters are simultaneously plotted, namely "SAVE" core (SYSSVE), "OVERLAYABLE" core (SYSOLY) and "AVAILABLE" core (SYSAVL). The key lists mnemonics, symbols plotted and the mean values of the plotted parameters.

To prepare such a graph for presentation, the analyst, using a remote terminal, first causes the analyzer to "plot" the information on the CRT. The data is then scaled to the analyst's preference, perhaps adding or deleting specific data curves to improve readability. The analyst then re-"plots" the graph on the CRT. When he is satisfied he directs this tailored graph to the line printer via an input command for hard copy.

(2) *Frequency distribution histograms* are useful in that they present a concise picture of the behavior of specific parameters. The analyzer will produce, instead of, or in addition to time graphs, a histogram of each desired parameter. The histogram is automatically scaled to the mode of the distribution, and is printed horizontally. The axis which represents the observed values may be rescaled as above to improve the clarity of the data presentation. These histograms show the distribution of observed values of the parameter in question. Figure 4 presents a histogram of system "OVERLAYABLE" core usage. It covers a longer period of time than the time graph and shows the frequency distribution of this parameter.

(3) *Correlation and regression analyses among data* play an important role in aiding in the forecasting of future needs. If the relationships among several parameters are known, then, should a change be projected in one parameter, its effect on the others can be approximated, and suitable adjustments in configuration or scheduling can be made. For example, if one determines a relationship between the amount of core memory and the overlay rate, one can predict the degree of decrease in the overlay rate should more core be added. The analyzer will allow correlations and/or regressions to be run among any of the parameters. Tables of pertinent statistics are printed as a result.

Figure 5 presents the results of a correlation and regression analysis among core utilization, processor utilization

THERE ARE 6 VARIABLES EACH HAVING 1754 OBSERVATIONS.

	MEAN	VARIANCE	STAN. DEV.
PCOLAY	3.20869441	15.10551621	3.88458156
OCOLAY	1.60584470	5.70391928	2.38828794
COOLAY	1.86622007	7.99887795	2.82840555
SYSSVF	100975.46180160	687123708.33425026	26213.08462160
SYSOLY	105438.86887115	723237908.80538526	26893.08291746
PROCUT	0.71918731	0.11147891	0.33388457

CORRELATION MATRIX

	PCOLAY	OCOLAY	COOLAY	SYSSVF	SYSOLY	PROCUT
PCOLAY	1.00000000E 00					
OCOLAY	8.39456288E-01	1.00000000E 00				
COOLAY	8.35541488E-01	9.09683445E-01	1.00000000E 00			
SYSSVF	6.58429988E-01	6.72702337E-01	6.48192137E-01	1.00000000E 00		
SYSOLY	-2.71466188E-01	-2.80469045E-01	-2.18795845E-01	-4.92990018-01	1.00000000E 00	
PROCUT	3.09396065E-01	3.38802545E-01	3.22533995E-01	2.89541795-01	2.26845385-01	1.00000000E 00

CONTROL INFORMATION

NUMBER OF OBSERVATIONS 1754

RESPONSE VARIABLE PROCUT

ANALYSIS OF VARIANCE TABLE

SOURCE OF VARIATION	D.F.	SUM OF SQUARES CORRELATION FORM	SUM OF SQUARES ORIGINAL UNITS	MEAN SQUARES CORRELATION FORM	MEAN SQUARES ORIGINAL UNITS	F
TOTAL	***	1.00000000	195.42252419			
REGRESSION	5	0.29266691	57.19370705	0.05853338	11.43874141	144.65087959
RESIDUAL	***	0.70733309	138.22881714	0.00040465	0.07907827	

VARIABLE	COEFFICIENT	S.E. COEF.	BETA	S.E. BETA	T	T PROB	PARTIAL R
PCOLAY	0.000481	0.003430	0.052157	0.039926	1.306334	0.808381	0.031230
OCOLAY	0.039692**	0.007373	0.283915	0.052739	5.383417	1.000000	0.127708
COOLAY	-0.014136*	0.006112	-0.119749	0.051780	-2.312453	0.979159	0.055230
SYSSVF	0.000005**	0.000000	0.379125	0.031073	12.201148	1.000000	0.281145
SYSOLY	0.000006**	0.000000	0.481338	0.023491	20.490164	1.000000	0.440079

CONSTANT	-0.45143744	R-SQ	0.29266691
S.E. CONSTANT	0.05741382	ADJ. R-SQ	0.29064365
T CONSTANT	-7.86287011	R	0.54098698
S.E. ESTIMATE	0.28120859	DURBIN WATSON	0.09100287
RHO	0.95446465	F PROBABILITY	1.00000000

Figure 5—Results of regression and correlation analysis

and overlay rate parameters. These results are seen to show that, for example, the overlay statistics are highly correlated to the amount of "SAVE" core in the system. This is understandable since the larger the "SAVE" core the greater the chance of needing to swap segments.

(4) *Scanning for peak periods* is a necessity in most computer systems, especially those operated in a data communication environment where the load fluctuates widely. The analyzer can scan the entire day's data and flag time intervals (of length greater than or equal to some specified minimum) during which the mean value of a parameter exceeded a desired threshold. For example, a period of five minutes or longer in which processor utilization exceeded 75 percent can be easily isolated (see Figure 6). Using this technique a peak period can be automatically determined and then further analyzed in more detail.

```
FOR 0740 SECONDS ENDING 09:50 PROCUT EXCEEDED THRESHOLD
FOR 2160 SECONDS ENDING 11:04 PROCUT EXCEEDED THRESHOLD
FOR 0700 SECONDS ENDING 11:49 PROCUT EXCEEDED THRESHOLD
FOR 1320 SECONDS ENDING 12:31 PROCUT EXCEEDED THRESHOLD
FOR 0480 SECONDS ENDING 12:39 PROCUT EXCEEDED THRESHOLD
FOR 2220 SECONDS ENDING 13:37 PROCUT EXCEEDED THRESHOLD
FOR 4920 SECONDS ENDING 15:03 PROCUT EXCEEDED THRESHOLD
FOR 0420 SECONDS ENDING 15:12 PROCUT EXCEEDED THRESHOLD
```

Figure 6—Periods of peak processor utilization

The design criteria discussed above have been met and a software monitoring system has been developed which is comprehensive, and easily used, and yet presents a negligible load upon the B6700 computer.

CONCLUSION AND OUTLOOK

The SPASM system has proven to be very instrumental in the performance evaluation of the B6700 system at the Bank. Several areas in which it has been and is currently being used are as follows:

- The statistics on processor queue length, multiplexor utilization, and disk controller utilization were used to aid in the analysis of the need for a second processor*, second multiplexor and additional controllers.
- The job core utilization data have been used to evaluate the effect of alternate programming techniques on memory use.
- Disk utilization data have been examined to identify any apparent imbalance of disk accesses among the disk electronics units.
- Processor queue data are being used to determine the effect of task priority on access to the processor.
- System overlay data are being used to determine the adequacy of automatic and manual job selection and scheduling.
- Processor utilization figures, as determined from the processor queue data, were used to determine the effect of core memory expansion on processor utilization.

Some future possible uses planned for SPASM include:

- Use of the scheduler queue statistics to evaluate the efficiency of the current MCP scheduling algorithm and to evaluate the effect changes to that algorithm have on the system performance.
- Use of the response time data to evaluate system efficiency throughout the day with different program mixes.
- Evaluation of resource needs of user programs.
- Evaluation of the effect that the Burroughs Data Management System has on system efficiency.
- Building of a B6700 simulation model using the collected statistics as input.
- Building an empirical model of the B6700 system by using the collected regression data.

* See Appendix A for a discussion of how the processor queue data was used to determine processor utilization.

The SPASM system has enabled us to collect a great deal of data on system efficiency and, consequently, a great deal of knowledge on how well the system performs its functions. This knowledge is currently being used to identify system problems and to aid in evaluating our current configuration and possible future configurations. Mere conjecture on system problems or system configurations in the absence of supporting data is not the basis for a logical decision on how to increase system efficiency. Performance measurement and evaluation are essential to efficient use of the system.

REFERENCES

1. *B6700 Information Processing Systems Reference Manual*, Burroughs Corp. May, 1972.
2. *B6700 Master Control Program Reference Manual*, Burroughs Corp. November, 1970
3. Organick, E. I., Cleary, J. G., "A Data Structure Model of the B6700 Computer System," *Sigplan Notices*, Vol. 6, No. 2, February 1971, "Proceedings of a Symposium on Data Structures in Programming Languages."

APPENDIX A

The use of processor queue data to determine processor utilization

The SPASM system records the length of the processor queue periodically. The processor utilization will be based upon these examinations, taking into account that the monitor itself is processing at this instant of time. If the processor queue is not empty, the monitor is preventing some other job from processing. Consequently, if the monitor were not in the system the processor would be busy with some other task at that instant of time. This is considered to be a processor "busy" sample. On the other hand, if the processor queue is empty at the sample time there is no demand for the processor other than the monitoring program itself. Therefore, if the monitor were not in the system at that instant of time the processor would be idle. This is considered a processor "idle" sample. Processor utilization can therefore be estimated as:

$$\text{processor utilization} = \frac{\text{No. "busy" samples}}{\text{total No. samples}}$$

This sampling approach to determining processor utilization was validated by executing controlled mixes of programs and then comparing the results of the sampling calculation of processor utilization to the job processor utilization given by:

$$\text{processor utilization} = \frac{\text{processor time logged against jobs in the mix}}{\text{elapsed time of test interval}}$$

Table II compares the results of these two calculations.

TABLE II—Comparison of Processor Utilization Statistics By Sampling Technique and By Processor Time Quotient Technique

	Average Processor Utilization (%)	
	Sampling Technique	Processor Time Quotient
Test Series 1	99.1	96.5
Test Series 2	57.6	53.5

In a second type of test, processor idle time was monitored (by means of a set of timing statements around the idling procedure) to gain a close measure of utilization. The total idle time was subtracted from the total elapsed time of the test to obtain the processor busy time and hence the utilization. Over a period of five hours the respective processor utilization calculations were:

Sampling Technique	46.3%
Idle Timing	48.0%

These results make us confident of the validity of using the processor queue sampling technique to accumulate processor utilization statistics during any given time interval.

Evaluation of performance of parallel processors in a real-time environment

by GREGORY R. LLOYD and RICHARD E. MERWIN

SAFEGUARD System Office
Washington, D.C.

INTRODUCTION

The use of parallelism to achieve greater processing thruput for computational problems exceeding the capability of present day large scale sequential pipelined data processing systems has been proposed and in some instances hardware employing these concepts has been built. Several approaches to hardware parallelism have been taken including multiprocessors[1,2,3] which share common storage and input-output facilities but carry out calculations with separate instruction and data streams; array processors[4] used to augment a host sequential type machine which executes a common instruction stream on many processors; and associative processors which again require a host machine and vary from bit[5] to word oriented[6] processors which alternatively select and compute results for many data streams under control of correlation and arithmetic instruction streams. In addition, the concept of pipelining is used both in arithmetic processors[7] and entire systems, i.e., vector machines[8] to achieve parallelism by overlap of instruction interpretation and arithmetic processing.

Inherent in this approach to achieving greater data processing capability is the requirement that the data and algorithms to be processed must exhibit enough parallelism to be efficiently executed on multiple hardware ensembles. Algorithms which must be executed in a purely sequential fashion achieve no benefit from having two or more data processors available. Fortunately, a number of the problems requiring large amounts of computational resources do exhibit high degrees of parallelism and the proponents of the parallel hardware approach to satisfying this computational requirement have shown considerable ingenuity in fitting these problems into their proposed machines.

The advocates of sequential pipelined machines can look forward to another order of magnitude increase in basic computational capability before physical factors will provide barriers to further enhancement of machine speed. When this limit is reached and ever bigger computational problems remain to be solved, it seems likely that the parallel processing approach will be one of the main techniques used to satisfy the demand for greater processing capability.

Computational parallelism can occur in several forms. In the simplest case the identical calculation is carried out on a number of separate data sets (array processing). A more complex case involves different calculations on separate data sets (multiprocessing) and finally, the greatest challenge to the parallel processing approach occurs when a single calculation on a single data set must be decomposed to identify parallel computational paths within a single computational unit. A number of mathematical calculations are susceptible to this type of analysis, e.g., operations on matrices and linear arrays of data.

The computational support required for a phased array radar is representative of problems exhibiting a high degree of parallelism. These systems can transmit a radar beam in any direction within its field of view in a matter of microseconds and can provide information on up to hundreds of observed objects for a single transmission (often called a "look"). The amount of information represented in digital form which can be generated by this type of radar can exceed millions of bits per second and the analysis of this data provides a severe challenge to even the largest data processors. Applications of this radar system frequently call for periodic up dates of position for objects in view which are being tracked. This cyclic behavior implies that a computation for all objects must be completed between observations. Since many objects may be in view at one time, these computations can be carried out for each object in parallel.

The above situation led quite naturally to the application of associative parallel processors to provide part of the computational requirements for phased array radars. A number of studies[9,10,11,12] have been made of this approach including use of various degrees of parallelism going from one bit wide processing arrays to word oriented processors. As a point of reference this problem has also been analyzed for implementation on sequential pipelined machines.[10] One of the main computational loads of a phased array radar involves the filtering and smoothing of object position data to both eliminate uninteresting objects and provide more accurate tracking information for objects of interest. A technique for elimination of uninteresting objects is referred to as bulk filtering and the smoothing of data on interesting objects is carried out with a Kalman filter.

The following presents an analysis of the results of the above studies of the application of associative parallel processors to both the bulk and Kalman filter problems. The two

criteria used to evaluate the application of parallel hardware to these problems are the degree of hardware utilization achieved and the increase in computational thruput achieved by introducing parallelism. The latter measure is simply the ratio of computational thruput achieved by the array of processing elements to the thruput possible with one element of the array. The Parallel Element Processing Ensemble (PEPE) considered as one of the four hardware configurations is the early IC model and is not the improved MSI PEPE currently under development by the Advanced Ballistic Missile Defense Agency.

Finally, a comparison of hardware in terms of number of logical gates is presented to provide a measure of computational thruput derived as a function of hardware complexity. The paper concludes with a number of observations relative to the application of the various associative parallel hardware approaches to this computational requirement.

FILTER COMPUTATIONS

The bulk and Kalman filters play complementary roles in support of a phased array radar. The task assigned to the radar is to detect objects and identify those with certain characteristics e.g. objects which will impact a specified location on the earth, and for those objects so identified, to provide an accurate track of the expected flight path. The bulk filter supports the selection process by eliminating from consideration all detected objects not impacting a specified area while the Kalman filter provides smoothed track data for all impacting objects. Both filters operate upon a predictive basis with respect to the physical laws of motion of objects moving in space near the earth. Starting with an observed position, i.e., detection by a radar search look, the bulk filter projects the position of the object forward in time, giving a maximum and minimum range at which an impacting object could be found in the next verification transmission. Based upon this prediction the radar is instructed to transmit additional verification looks to determine that this object continues to meet the selection criteria by appearing at the predicted spot in space following the specified time interval.

Those objects which pass the bulk filter selection criteria are candidates for precision tracking by the radar and in this case the Kalman filter provides data smoothing and more precise estimates of the object's flight path. Again a prediction is made of the object's position in space at some future time based upon previously measured positions. The radar is instructed to look for the object at its predicted position and determines an updated object position measurement. The difference between the measured and predicted position is weighted and added to the predicted position to obtain a smoothed position estimate. Both the bulk and Kalman filter are recursive in the sense that measurement data from one radar transmission is used to request future measurements based upon a prediction of a future spatial position of objects. The prediction step involves evaluation of several terms of a Taylor expansion of the equations of motion of spatial objects. Detailed discussion of the mathematical basis for these filters can be found in the literature on phased array radars.[18,19]

The computations required to support the bulk filter are shown in Figure 1. The radar transmissions are designated as either search or verify and it is assumed that every other transmission is assigned to the search function. When an object is detected, the search function schedules a subsequent verification look typically after fifty milliseconds. If the verification look confirms the presence of an object at the predicted position another verification look is scheduled again after fifty milliseconds. When no object is detected on a verification look, another attempt can be made by predicting the object's position ahead two time intervals i.e., one hundred milliseconds, and scheduling another verification look. This procedure is continued until at least M verifications have been made of an object's position out of N attempts. If $N - M$ attempts at verification of an object's position result is no detection then the object is rejected. This type of filter is termed an M out of N look bulk filter.

Turning now to the Kalman filter the computational problem is much more complex. In this case a six or seven element state vector containing the three spatial coordinates, corresponding velocities, and optionally an atmospheric drag coefficient is maintained and updated periodically for each tracked object. A block diagram of this computation is shown in Figure 2. The radar measurements are input to state vector and weighting matrix update procedures. The weighting matrix update loop involves an internal update of a covariance matrix which along with the radar measurements is used to update a weighting matrix. The state vector update calculation generates a weighted estimate from the predicted and measured state vectors. The Kalman filter computation is susceptible to decomposition into parallel calculations and advantage can be taken of this in implementations for a parallel processor.

COMPUTATIONAL MODELS

Bulk filter

The bulk filter is designed to eliminate with a minimum expenditure of computational resources a large number of uninteresting objects which may appear in the field of view of a phased array radar. A model for this situation requires

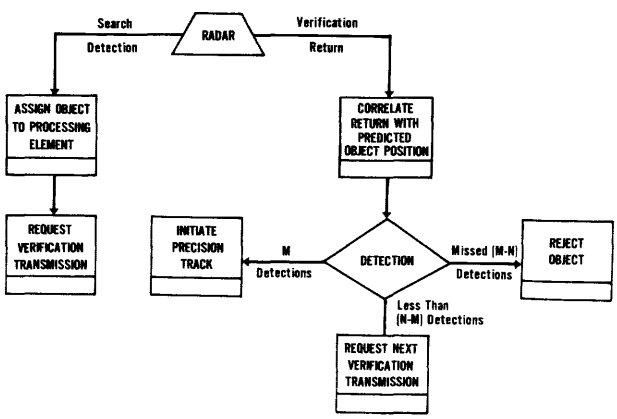


Figure 1 -Bulk filter flow diagram

assumptions for the number and type of objects to be handled, efficiency of the filter in eliminating uninteresting objects, and radar operational parameters. These assumptions must produce a realistic load for the filter which would be characteristic of a phased array radar in a cluttered environment. The assumptions, which are based upon the Advanced Ballistic Missile Agency's Preliminary Hardsite Defense study, are:

1. The radar transmits 3000 pulses, i.e. looks, per second and every other one of these is assigned to search.
2. New objects enter the system at a rate of 100 per 10 milliseconds (Ms) all of which are assumed to be detected on one search look.
3. Fifteen objects are classified as being of interest, i.e. impacting a designated area (must be precision tracked), and 85 of no interest (should be eliminated from track).
4. Following detection an attempt must be made to locate each object not rejected by the filter every 50 Ms.
5. The filter selection criteria is 5 (M) detections out of 7 (N) attempts. Failure to detect the object three times in the sequence of 7 looks results in rejection.
6. The filter is assumed to reduce the original 100 objects to 70 at the end of the third; 45 at the end of the fourth; 30 at the end of the fifth; 25 at the end of the sixth; and 20 at the end of the seventh look; thus failing to eliminate 5 uninteresting objects.

Based upon the above assumptions the bulk filter accepts 500 new objects every 50 Ms. When operational steady state is reached, the processing load becomes 100 search and 290 verify calculations every 10 Ms. Each object remains in the filter for a maximum of 350 Ms and for a 50 Ms interval 1950 filter calculations are required corresponding to 10,000 new objects being detected by the radar per second.

The above process can be divided into two basic steps. The first involves analysis of all radar returns. For search returns the new data is assigned to an available processor. For verify returns each processor must correlate the data with that

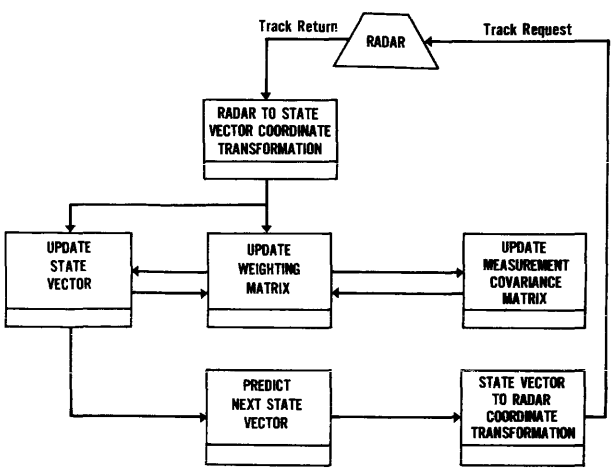


Figure 2—Kalman filter flow diagram

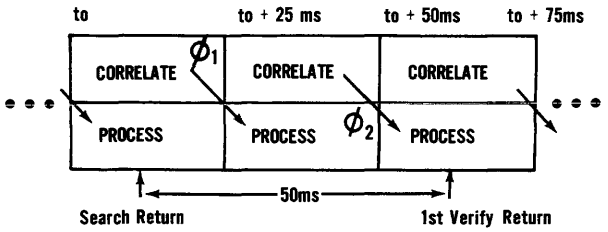


Figure 3—Correlation and arithmetic phases

being processed to determine if it represents new positional information for an object being tracked by that processor. For all objects in process, new data must be received every 50 Ms or it is considered to have not been redetected and hence subject to rejection by the filter. The associative processors studied were unable to carry out the required calculations within the pulse repetition rate of the radar (330 μsec). To achieve timely response, the processing was restructured into correlation and arithmetic cycles as shown in Figure 3. During the first 25 msec interval, the processors correlate returns from the radar with internal data (predicted positions). During the subsequent 25 msec interval, processors carry out the filter calculations and predict new object positions. This approach allowed all required processing to be completed in a 50 msec interval. Objects which fail the selection criteria more than two times are rejected and their processor resources are freed for reallocation.

Kalman filter

The Kalman filter computation requires many more arithmetic operations than the bulk filter. The radar becomes the limiting factor in this case since only one object is assumed for each look. Assuming a radar capable of 3000 transmissions per second and a 50 Ms update requirement for each precision track, a typical steady state assumption would be 600 search looks and 2400 tracking looks per second (corresponding to 120 objects in precision track). At this tracking load it must again be assumed that the 50 Ms update interval is divided into 25 Ms correlation and compute cycles as was done for the bulk filter and shown in Figure 3. This implies that 60 tracks are updated every 25 Ms along with the same number of verify looks being received and correlated.

EVALUATION APPROACH

The three quantities of interest in determining the relation between a parallel processor organization and a given problem are: resources required by the problem, resources available from the processor configuration, and time constraints (if any). A more precise definition of these quantities follows, but the general concept is that the processor capabilities and problem requirements should be as closely balanced as possible.

Quantitative resource measures and balance criteria are derived from Chen's[14] analysis of parallel and pipelined computer architectures. Chen describes the parallelism inherent in a job by a graph with dimensions of parallelism

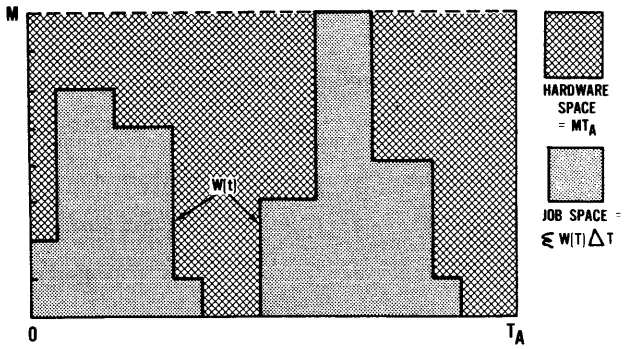


Figure 4—Hardware and job space diagram

width (number of identical operations which may be performed in parallel) and execution time. The ratio ρ is defined for a job as the area under the step(s) showing parallelism (width $W>1$) divided by the total area swept out by the job. The hardware efficiency factor, η, is the total job work space (defined as the product of execution time and the corresponding job width W summed over all computations) over the total hardware work space (defined as the product of total execution time and the number, M, of available parallel processors). This provides a measure of utilization of a particular hardware ensemble for each segment of a computation. A modification of Chen's η allows consideration of time constraints. Hardware space will now be defined as a product of the total time *available* to carry out the required computation times M, the number of processors available. Call this ratio $\bar{\eta}$. The work space is as defined above except that periods of no processor activity may be included, i.e., job width $W=0$. Figure 4 illustrates these concepts showing a computation involving several instruction widths carried out in an available computation time T_a. The stepwise value of η varies during job execution and the average value for the whole job becomes: (T_a is divided into N equal time intervals $=\Delta T$, $W(T_i)>0$ for K steps).

$$\bar{\eta}=\frac{\sum_{i=0}^{N} W(T_i)\Delta T}{MT_a} \quad \text{where} \quad \eta=\frac{\sum_{i=0}^{N} W(T_i)\Delta T}{MK\Delta T} \qquad (1,2)$$

Note that under this interpretation, $\bar{\eta}$ measures the *fit* between this particular problem and a given configuration. If $\bar{\eta}=1.0$ the configuration has *precisely* the resources required to solve the problem within time constraints, assuming that the load is completely uniform (with non-integral width in most cases). Although $\bar{\eta}$ will be much less than 1.0 in most cases, it is interesting to compare the values obtained for processors of different organizations and execution speeds, executing the same job (identical at least on a macroscopic scale). Implicit in the stepwise summation of the instruction time—processor width product are factors such as the suitability of the particular instruction repertoire to the problem (number of steps), hardware technology (execution time), and organizational approach (treated in the following section).

A criterion π is expressed as the inverse ratio of time of execution of a given job with parallel processors to the execution time with only one such processor (speedup over the job). Expressing π in terms of job width W gives for any job step

$$\pi=\frac{\text{sequential processor execution time}}{\text{parallel processor execution time}}=W(T) \qquad (3)$$

Similarly, averaging this quantity over an entire job during the available time gives:

$$\bar{\pi}=\frac{\sum_{i=0}^{N} W(T_i)\Delta T_i}{Ta} \qquad (4)$$

or simply:

$$\bar{\pi}=\bar{\eta}M \qquad (5)$$

which states that the speed of execution of a computation on parallel hardware as contrasted to a single processing element of that hardware is proportional to the efficiency of hardware utilization times the number of available processing elements. Again, $\bar{\pi}$ measures the equivalent number of parallel processors required assuming a uniform load (width $=\bar{\pi}$, duration $=Ta$).

PROCESSOR ORGANIZATIONS

General observations

In the analysis which follows, job parallelism is calculated on an instruction by instruction step basis. For the purposes of this discussion, consider a more macroscopic model of job parallelism. Sets of instructions with varying parallelism widths will be treated as phases (Φ_i), with phase width defined as the maximum instruction width within the phase. (see Figure 5, for a three phase job, with instructions indicated by dashed lines).

Given this model, it is possible to treat the parallel component in at least three distinct ways (see Figure 6). The simplest approach is to treat a parallel step of width N as N steps of width one which are executed serially. This would correspond to three loops (with conditional branches) of iteration counts W_1, W_2, W_3, or one loop with iteration count $\max[W_1, W_2, W_3]$. The worst case execution time (macroscopic model) for width N would be $T=Nts$.

Parallel processing, in its purest sense, devotes one processing element (PE) to each slice of width one, and executes the total phase in $T=ta$, where ta is the total execution time for any element. Variations on this basic theme are possible. For example, STARAN, the Goodyear Aerospace associative

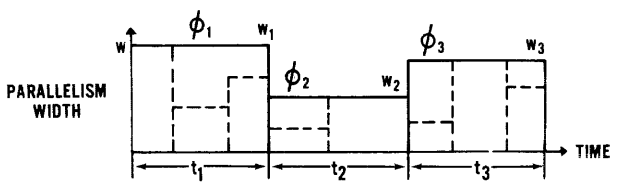


Figure 5—Three phase job space diagram

processor,[5,9,12] is actually an ensemble of bit-slice processors[16] arranged in arrays of 256 each having access to 256 bits of storage. The ensemble is capable of bitwise operations on selected fields of storage. Since the bulk filter algorithm requires 768 bits of storage for the information associated with one filter calculation, i.e. track, a "black box" model devotes three PE's to each object in track (generally one of the three is active at any instruction step).

The converse of the STARAN case is exemplified by the Parallel Element Processing Ensemble (PEPE),[6] which devotes M PE's to N tracks, $M<N$. In this case, the total processing time for one phase would be $T=[N\div M]ta$, since each PE may process up to $[N\div M]$ tracks sequentially. Note that for parallel correlation of K returns (associative operations such as "between limits search"), at least K PE's must be available, since objects which are illuminated by a single beam must be handled by separate processors.

A third approach is analogous to the pipelining of instruction execution. Assuming that each phase has execution time tp, one could use one sequential processor to handle execution of each phase, buffering the input and output of contiguous phases to achieve a total execution time of $T=(N-1+m)tp$ for an M stage process. The Signal Processing Element (SPE) designed by the US Naval Research Laboratory[15] can utilize this strategy of functional decomposition, linking fast microprogrammed arithmetic units under the control of a master control unit to achieve $tp\leq M^{-1}ts$ for "sequential" machines of the CDC 7600, IBM 370/195 class $(T\leq[M^{-1}(N-1)+1]ts)$.

One other factor of considerable importance is the number of control streams active in each processor array. The simplest arrangement is a single control stream, broadcast to all elements from a central sequencing unit. Individual PE's may be deactivated for part of a program sequence by central direction, or dependent upon some condition determined by each PE. Dual control units mean that arithmetic and correlative operation can proceed simultaneously, allowing the two phase strategy outlined earlier to work efficiently (one control stream would require an "interruptible arithmetic" strategy, or well defined, non-overlapping, search/verify and arithmetic intervals). These two control streams can act on different sets of PE's (e.g. each PE has a mode register which determines the central stream accepted by that PE), or both control streams can share the same PE on a cycle stealing basis (PEPE IC model).

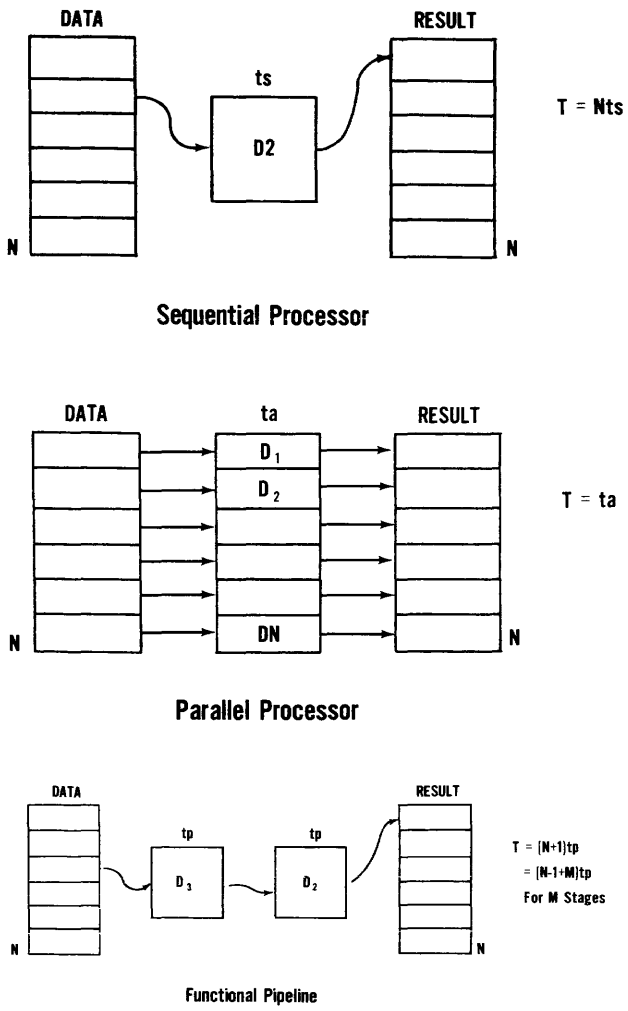


Figure 6—Decomposition of parallelism in three processor organizations

Configurations considered

Table I presents the basic data on the four hardware configurations considered for the bulk filter problem. Sizing estimates are based upon the assumptions described previously, i.e. 1950 tracks in processing at any given time (steady state). Over any 25 Ms interval, half of the tracks are being correlated, half are being processed arithmetically.

The Kalman filter results compare the performance of STARAN, PEPE (IC model), and the CDC 7600 in sustaining a precise track of 120 objects (1 observation each 50 Ms) using a basic model of the Kalman filter algorithm. The STARAN solution attempts to take advantage of the parallelism within the algorithm (matrix-vector operations). Twenty-one PE are devoted to each object being tracked. PEPE would handle one Kalman filter sequence in each of its PE's, performing the computations serially within the PE.

COMPARATIVE RESULTS

Bulk filter

Table II presents the values for η, $\bar{\eta}$, $\bar{\pi}$, and execution time for each of the 4 processor configurations. As has been explained earlier η and $\bar{\eta}$ differ only in the definition of hardware space used in the denominator of the η expression. It is interesting to note that although the efficiency $\bar{\eta}$ over the constrained interval is not large for any of the parallel processors, all three do utilize their hardware efficiency over the actual arithmetic computation (η). The implication is that some other task could be handled in the idle interval, a

TABLE I—Bulk Filter Processor Configuration Comparison

	STARAN 3 PE/TRACK	HONEYWELL PE/TRACK	PEPE (IC model) PE/20 TRACK	CDC 7600 SEQUENTIAL
Number of PE's (1950 track load)	30 Arrays=7680 PE's	1950	100	1
32 bit fixed point Add time/PE	18.0 μsec	.75 μsec	.25 μsec	27.5–55 n.s. (60 bit)
Control Streams	Single—(Standard option) all PE's correlate or perform arithmetic functions	Double—Each PE may be in correlation *or* arithmetic mode	Double—EACH PE may perform correlation *and* arithmetic functions simultaneously	Single pipelined
Approximate gate count/PE (not including storage)	82 (21,000/256 PEarray)	2,400	9,000	170,000
Gate Count for configuration (PE's only)	630,000	4.68×10^6	900,000	170,000
Adds/sec$\times10^6$ (~MIPS)	427	2600	400	18
Gates/track	320*	2400	450	87

* Based on a 30 Array configuration—246 are required for the algorithm.

more sophisticated filter algorithm could be used, or the PE's could be built from slower, less expensive logic. It should be stressed that the gate counts given are strictly processing element gates, not including memory, unit control, or other functions.

Kalman filter

As noted above, 60 precision tracks must be updated in 25 Ms while 60 track looks are being correlated. Benchmark data for the CDC 7600 indicates that a Kalman filter calculation consisting of 371 multiplies, 313 add/subtracts, 2 divides, 6 square roots, and 1 exponentiation will require approximately 0.3 Ms (18 Ms for 60 tracks). This leaves a reserve of 7 Ms out of a 25 Ms interval for correlation. An analysis of the STARAN processor applied to the Kalman filter[12] indicates that with 21 processing elements assigned to each precision track the calculation can be carried out in slightly less than 25 Ms. This performance is achieved by decomposing the filter calculation into 56 multiplies, 61 add/subtracts, 3 divides, 4 square roots and is achieved at the cost of 322 move operations. Figure 7 shows the processor activity for the first 15 instructions of the Kalman filter sequence. One bank of STARAN processing elements (5 256 element arrays) containing 1280 processors is required to update tracks for 60 objects in one 25 Ms interval and correlate returns during the other. The PEPE configuration would require 60 processing elements (two track files per element) taking advantage of this hardware's ability to do arithmetic calculations and correlations simultaneously, achieving a 45 percent loading (11.3 Ms execution time per Kalman filter sequence) of each PEPE processing element. Table III summarizes the Kalman filter results.

OBSERVATIONS AND CONCLUSIONS

It should be emphasized that this study was not an attempt to perform a qualitative evaluation of the processor organizations described in the studies.[9,10,11,12] Each of the proposed configurations is more than capable of handling the required calculations in the time available. System cost is really outside the scope of this paper. In particular, gate count is *not* a good indicator of system cost. The circuit technology (speed, level of integration) and chip partitioning (yield, number of unique chips) trade-offs possible within the current state of the art in LSI fabrication relegate gate count to at most an order of magnitude indicator of cost.

Each of the three parallel processor organizations represents a single point on a trade-off curve in several dimensions (i.e. processor execution speed, loading, and cost, control stream philosophy, etc.). Given an initial operating point, determined by the functional requirements of the problem, the system designer must define a set of algorithms in sufficient detail to

TABLE II—Results of Bulk Filter Analysis

	STARAN 3 PE/ TRACK	HONEY-WELL PE/TRACK	PEPE (IC model) PE/20 TRACKS	CDC 7600 SE-QUEN-TIAL
Correlation time	1.8 msec	15.9 msec	2.1 msec	
$\eta_{\phi 1}$	.036	.035	.68	
$\pi_{\phi 1}$	139	34	68	
$\bar{\eta}_{\phi 1}$	.0026	.022	.057	
$\bar{\pi}_{\phi 1}$	9.9	22	5.7	
Arithmetic time	14.5 msec	5.1 msec	3.85 msec	
$\eta_{\phi 2}$	.66	.80	.79	
$\pi_{\phi 2}$	2450	780	79	
$\bar{\eta}_{\phi 2}$	.38	.16	.122	
$\bar{\pi}_{\phi 2}$	1470	159	12.2	
Total time	16.3 msec	21 msec	5.95 msec	22 msec
η	.30	.11	.75	1.0
π	2270	216	75	1.0
$\bar{\eta}$	.19	.093	.18	.88
$\bar{\pi}$	1480	181	18	.88
η.MIPS	128	286	300	18

convince himself that he can operate within any given constraints. Fine tuning of the system is accomplished by restructuring the algorithms, redefining the operating point, or both. In the two cases treated in this paper, elapsed time is the crucial measure of system performance (in a binary sense—it does or does not meet the requirement). The purpose of the η and $\bar{\eta}$ calculations, as well as the step by step processor activity diagrams is to provide some insight beyond the elapsed time criteria which might be helpful in restructuring algorithms, or modifying some aspect of the system's architecture such as control stream philosophy. The properties of the processor activity diagrams are of significant interest in determining the number of PE's that are required to handle the given load (uniform load implies fewer PE's and higher η). The measures used in this paper are of some interest because of the fact that they are functions of problem width *and* instruction execution time, allowing factors such as the selection of a particular instruction set to enter into the values of the resultant tuning parameters.

Several more specific observations are in order. First, for the particular bulk filter case considered, the CDC 7600 can easily handle the computational load. Proponents of the parallel processor approach would claim that quantity production of PE's, utilizing LSI technology, would enable them to produce equivalent ensembles at less than a CDC 7600's cost. In addition, computation time for the parallel ensembles is only a weak function of the number of objects in the correlation phase, and essentially independent of object load in the arithmetic phase. Therefore, it would be simple to scale up the capabilities of the parallel processors to handle loads well beyond the capability of a single, fast sequential processor. The functional pipelining approach advocated by the Naval Research Laboratory would appear to be the strongest challenger to the parallel approach in terms of capabilities and cost (and to a somewhat lesser extent, flexibility). Very rough estimates indicate that the bulk filter case presented here could be handled by no more than two arithmetic units (each with ~10,000 gates) and a single microprogrammed control unit (~5,000 gates). Tasks which stress the correlative capabilities of parallel arrays rather than the parallel arithmetic capabilities should show the parallel array architecture to its greatest advantage.

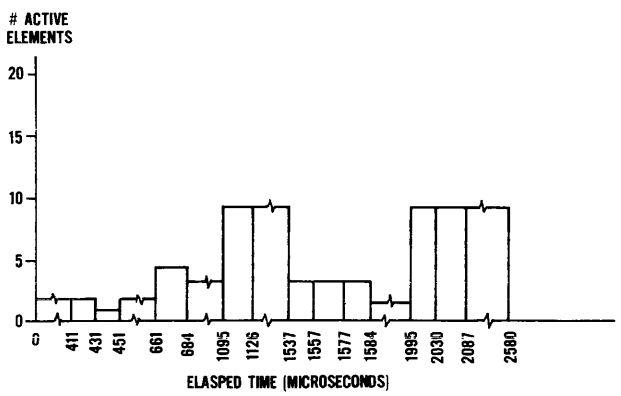


Figure 7—STARAN Kalman filter loading (one track, first 15 instructions)

TABLE III—Results of Kalman Filter Analysis

	STARAN	PEPE (IC model)	CDC 7600
Time	25 Ms	11.3 Ms	18 Ms
η	~.5	>.9	1
π	640	>54	1
$\bar{\eta}$	~.5	.45	.72
$\bar{\pi}$	640	27	.72
$\eta\mu$MIPS	36	108	13
gates/track	875	4500	1400

NOTE: Correlation time for the Kalman filter is not significant (~100 μs) since each track is assigned a unique track number number (120 total). Accordingly, only total time figures are presented.

ACKNOWLEDGMENTS

The authors wish to express their gratitude to Messrs. Brubaker and Gilmore, Goodyear Aerospace Corporation for providing STARAN logic gate counts and Kalman filter computation timing estimates; to Mr. W. Alexander, Honeywell Inc. for providing PEPE and Honeywell Associative Processor Logic gate counts; and to Mr. J. Burford, Control Data Corporation for providing the CDC 7600 logic gate counts. The study reports[9,10,11] were prepared under the direction of the Advanced Ballistic Missile Defense Agency, whose cooperation is greatly appreciated."

REFERENCES

1. Conway, M. E., "A Multi-Processor System Design," *Proceedings AFIPS*, FJCC, Vol. 24, pp. 139-146, 1963.
2. Stanga, D. C., "UNIVAC 1108 Multi-Processor System," *Proceedings AFIPS*, SJCC, Vol. 31, pp. 67-74, 1967.
3. Blakeney, G. R., et al, "IBM 9020 Multiprocessing System," *IBM Systems Journal*, Vol. 6, No. 2, pp. 80-94, 1967.
4. Slotnik, D. L., et al, "The ILLIAC IV Computer, *IEEE Transaction on Computers*, Vol. C-17, pp. 746-757, August 1968.
5. Rudolph, J. A., "STARAN, A Production Implementation of an Associative Array Processor," *Proceedings AFIPS*, FJCC, Vol. 41, Part I, pp. 229-241, 1972.
6. Githens, J. A., "A Fully Parallel Computer for Radar Data Processing," *NAECON 1970 Record*, pp. 302-306, May 1970.
7. Anderson, D. W., Sparacio, F. J., Tomasula, R. M., "System/360 MOD 91 Machine Philosophy and Instruction Handling," *IBM Journal of Research and Development*, Vol. II, No. 1, pp. 8-24, January 1967.
8. Hintz, R. G., "Control Data Star 100 Processor Design," *Proceedings COMPCON '72*, pp. 1-10, 1972.
9. Rohrbacher, *Terminal Discrimination Development*, Goodyear Aerospace Corporation, Contract DAHC60-72-C-0051, Final Report, March 31, 1972.
10. Schmitz, H. G., et al, *ABMDA Prototype Bulk Filter Development*, Honeywell, Inc., Contract DAHC60-72-C-0050, Final Report, April 1972, HWDoc #12335-FR.
11. *Phase I—Concept Definition Terminal Discrimination Development Program*, Hughes Aircraft Company, Contract DAHC60-72-C-0052, March 13, 1972.

12. Gilmore, P. A., *An Associative Processing Approach to Kalman Filtering*, Goodyear Aerospace Corporation, Report GER 15588, March 1972.
13. Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, Vol. 82D, pp. 35-45, 1960.
14. Chen, T. C., "Unconventional Superspeed Computer Systems," *Proceedings AFIPS*, SJCC, Vol. 39, pp. 365-371, 1971.
15. Ihnat, J., et al, *Signal Processing Element Functional Description, Part I*, NRL Report 7490, September 1972.
16. Shore, J. E., *Second Thoughts on Parallel Processing*, Naval Research Laboratory Report #7364, December 1971.

A structural approach to computer performance analysis

by P. H. HUGHES and G. MOE

University of Trondheim
Norway

INTRODUCTION

The performance analysis of computer systems is as yet a rather unstructured field in which particular aspects of systems or items of software are studied with the aid of various forms of models and empirical measurements. Kimbleton[1] develops a more general approach to this problem using three primary measures of system performance. The approach to be described here represents a similar philosophy but deals only with throughput as the measure of system performance. This restriction results in a model which is convenient and practical for many purposes, particularly in batch-processing environments. It is hoped that this approach will contribute to the development of a common perspective relating the performance of different aspects of the system to the performance of the whole.

The present work arises out of a continuing program of performance evaluation begun in 1970 on the UNIVAC 1108 operated by SINTEF (a non-profit engineering research foundation) for the Technical University of Norway (NTH) at Trondheim. The status of the Computing Centre has since been revised, such that it now serves the newly formed University of Trondheim which includes NTH.

Early attention focused on trying to understand EXEC 8 and to identify possible problem areas. One of the early fruits of the work is described in Reference 2. However the main emphasis for most of 1971 was on the development of benchmark techniques supported by a software monitoring package supplied by the University of Wisconsin as a set of modifications to the EXEC 8 operating system.[3,4] It was in an effort to select and interpret from the mass of data provided by software monitoring that the present approach emerged.

The operation of a computer system may be considered at any number of logical levels, but between the complexities of hardware and software lies the relatively simple functional level of machine code and I/O functions, at which processing takes physical effect. The basis of this paper is a general model of the processes at this physical level, to which all other factors must be related in order to discover their net effect on system performance.

THE GENERAL NATURE OF THE WORKLOAD

At the level we shall consider, a computer is a network of devices for transferring and processing information. A program is a series of requests for action by one or more devices, usually in a simple, repetitive sequence.

The way in which instructions are interpreted means that the central processor is involved every time I/O action is to be initiated, so that every program can be reduced to a cycle of requests involving the CPU and usually one or more other devices.

In a single program system, devices not involved in the current program remain idle, and the overlap between CPU and I/O activity is limited by the amount of buffer space available in primary store. Performance analysis in this situation is largely a matter of device speeds and the design of individual programs.

Multiprogramming overcomes the sequential nature of the CPU-I/O cycle by having the CPU switch between several such programs so as to enable all devices to be driven in parallel. The effectiveness of this technique depends upon several factors:

(i) the buffering on secondary storage of the information flow to and from slow peripheral devices such as readers, printers and punches ('spooling' or 'symbiont activity').
(ii) the provision of the optimum mix of programs from those waiting to be run so that the most devices may be utilised (coarse scheduling).
(iii) a method of switching between programs to achieve the maximum processing rate (dynamic scheduling).

These scheduling strategies involve decisions about allocating physical resources to programs and data. The additional complexity of such a system creates its own administrative overhead which can add significantly to the total workload. The success of the multiprogramming is highly sensitive to the match between physical resources and the requirements of the workload. In particular there must be:

(iv) provision of sufficient primary store and storage management to enable a sufficient number of programs to be active simultaneously.
(v) a reasonable match between the load requirements on each device and the device speed and capacity so as to minimise 'bottleneck' effects whereby a single overloaded device can cancel out the benefits of multiprogramming.

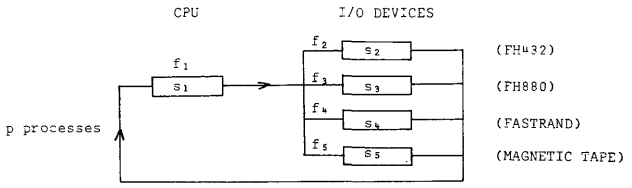


Figure 1—Configuration used by the model. Names in parentheses denote corresponding equipment

Such a system, if properly configured and tuned, can achieve a throughput rate several times greater than a single program system. Performance analysis becomes much more complex, but correspondingly more important.

A MODEL OF THE PHYSICAL WORKLOAD

We wish to model a number of independent processes, each consisting of a cycle of CPU-I/O requests. In real life, the nature of these processes changes dynamically in an extremely complex way and we introduce several simplifying assumptions:

(i) the processes are statistically identical and are always resident in core store.
(ii) requests by a process are distributed among devices according to a stationary, discrete probability distribution function f.
(iii) the time taken to service a request on a particular device i is drawn from a stationary distribution function s.

Figure 1 illustrates the network we have been using. The names in parentheses refer to the particular equipment in use at our installation. FH432, FH880, and FASTRAND constitute a hierarchy of fast, intermediate, and large slow drums respectively.

We restrict the model to those devices which satisfy the constraint that any process receiving or waiting to receive service from the device must be resident in core store. If a process requires service from a device which does not satisfy this constraint (e.g., a user terminal) it is no longer 'active' in the terms of the model. Normally it will be replaced in core by some other process which is ready to proceed. This restriction rules out an important class of performance variables such as system response time, but has corresponding advantages in dealing with throughput.

At any instant the number of active, in-core processes p is discrete, but the average over a period of time may be non-integer, as the mix of programs that will fit into core store changes. In addition p includes intermittent processes such as spooling which will contribute fractionally to its average value. We will refer to p as the *multiprogramming factor*. This is to be distinguished from the total number of open programs (including those not in core store) which is sometimes referred to as the degree of multiprogramming.

In the first instance, all distribution functions have been assumed to be negative exponential with appropriate mean values for each device. First-in first-out queuing disciplines are also assumed. This is not strictly valid under EXEC 8 in the case of the CPU, but the assumption does not upset the general behaviour of the model.

Throughput and load

We may define the throughput of such a network to be the number of request cycles completed per second for a given load. The load is defined by two properties

(i) the distribution of requests between the various devices
(ii) the work required of each device.

This 'work' cannot be easily specified independently of the characteristics of the device, although conceptually the two must be distinguished. The most convenient measure is in terms of the time taken to service a request, and the distribution of such service times with respect to a particular device.

For a given load, the distribution of requests among N devices is described by a fixed set of probabilities $f_i (i=1, \ldots N)$. The proportion of requests going to each device over a period of time will therefore be fixed, regardless of the rate at which requests are processed. This may be stated as an *invariance rule* for the model, expressible in two forms with slightly different conditions:
For a given load distribution f

(i) the ratio of request service rates on respective devices is invariant over throughput changes and
(ii) if the service times of devices do not change, the ratio of utilizations on respective devices is invariant over throughput changes.

Behaviour of the model

Figures 2(a), 2(b) show how the request service rate and the utilization of each device vary with the number of active processes p, for a simulation based on the UNIVAC 1108 configuration and workload at Regnesentret.

These two figures illustrate the respective invariance rules just mentioned. Furthermore the two sets of curves are directly related by the respective service times of each device j. In passing we note that the busiest device is not in this case the one with the highest request service rate, which is always the CPU. Since we have assumed that service times are independant of p (not entirely true on allocatable devices), it is clear that the shape of every curve is determined by the same function $F(p)$ which we will call the *multiprogramming gain function*, shown in Figure 2(c).

This function has been investigated analytically by Berners-Lee.[5] Borrowing his notation, we may express the relationship of Figure 2(b) as

$$X_j(p) = F(p) x_j \quad \text{for} \quad j=1, 2, \ldots N \qquad (1)$$

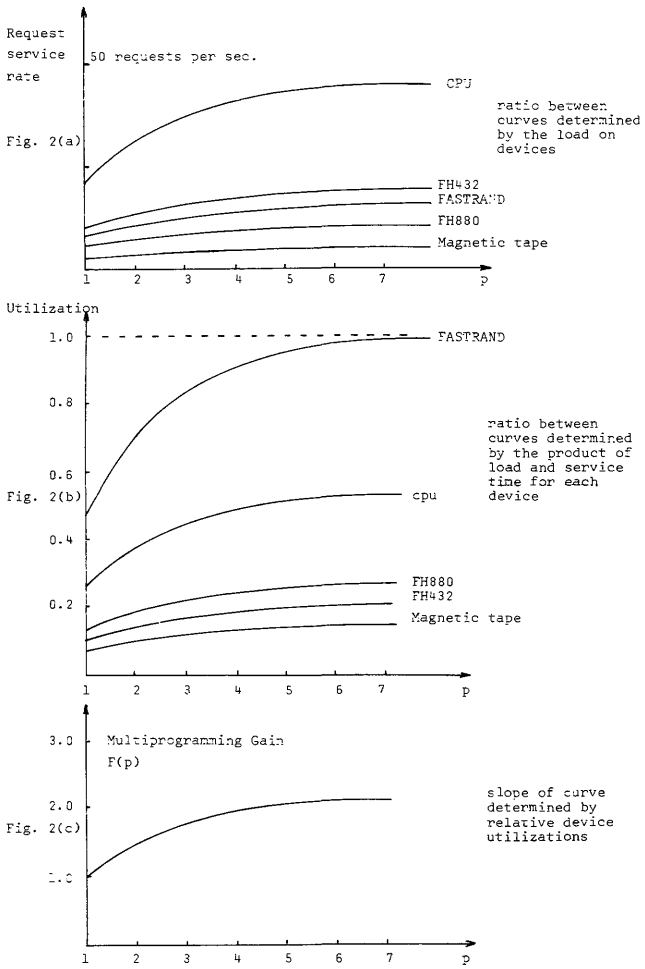


Figure 2—Behaviour of model as a function of the multiprogramming factor p

where $X_j(p)$ is the utilization of device j when p processes are active and x_j is identical with $X_j(1)$, the corresponding utilization when only one process is active.

Now when only one process is active, one and only one device will be busy at a time, so that

$$\sum_{j=1}^{N} x_j = 1 \tag{2}$$

Summing equations (1) for all devices, we have

$$\sum_{j=1}^{N} X_j(p) = F(p) \tag{3}$$

That is, the multiprogramming gain for some degree of multiprogramming p is equal to the sum of the device utilizations.

It is instructive to consider the meaning of the function $F(p)$. Multiprogramming gain comes about only by the simultaneous operation of different devices. If there are p parallel processes on N devices, then some number of processes q satisfying $q \leq \min(p, N)$ are actually being serviced at any instant, while $(p-q)$ processes will be queuing for service.

It follows that the time-averaged value of q must be $F(p)$, so that one may regard $F(p)$ as the true number of processes actually receiving service simultaneously. Since q is limited by the number of devices N it follows that the *maximum value of* $F(p)$ *must be* N which we would expect to be the case in a perfectly balanced system, at infinitely large p.

Improving system throughput

We shall consider in detail two ways of improving the throughput of such a system:

(i) by increasing the multi-programming factor
(ii) by improving the match between system and workload

Later we shall encounter two further ways by which throughput could be improved:

(iii) by reducing the variance of request service times
(iv) by improving the efficiency of the software so that the same payload is achieved for a smaller total workload.

Increasing the multiprogramming factor

We may increase the number of active processes by either acquiring more core store or making better use of what is available by improved core store management, or smaller programs. The latter two alternatives may, of course, involve a trade-off with the number of accesses required to backing store.

The maximum gain obtainable in this way is limited by the utilization of the most heavily used device—the current "bottleneck." If the present utilization of this device is X_m, and the maximum utilization is unity, then the potential relative throughput gain is $1/X_m$ (Figure 3). The practical limit will of course be somewhat less than this because of diminishing returns as p becomes large. A simple test of whether anything is to be gained from increasing p is the existence of any device whose utilization approaches unity. If such a device does not exist then p is a "bottleneck." However, the converse does not necessarily apply if the limiting device contains the swapping file.

Matching system and workload

This method of changing performance involves two effects which are coupled in such a way that they sometimes conflict with each other. They are:

(a) Improvement in monoprogrammed system performance
(b) Improvement in multiprogramming gain

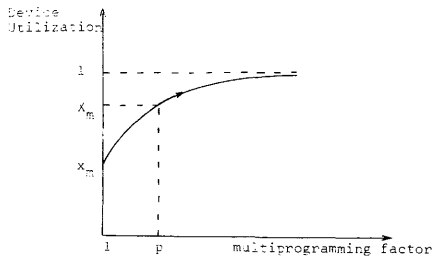


Figure 3(a)—The effect of increasing multiprogramming on the utilization of the limiting device m

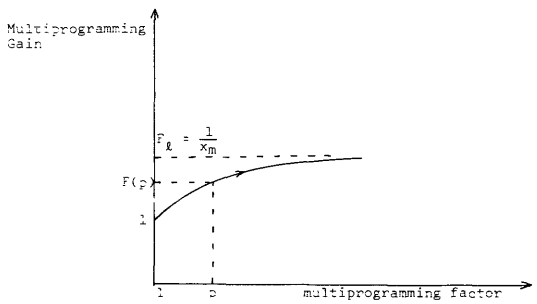


Figure 3(b)—The effect of increasing multiprogramming on the multiprogramming gain function $F(p)$

These two components of system performance are seen in (4) where r is the total no. of requests processed per second on all devices when $p=1$, and R is the corresponding total at the operational value of p (Fig. 2(a)).

$$R=F(p)r \tag{4}$$

Clearly we may improve the total performance R by improving either r or $F(p)$, but we shall see that any action we take to improve one has some effect on the other.

At $p=1$ the mean time to perform any request is $\sum f_i s_i$ hence

$$r=\frac{1}{\sum f_i s_i} \tag{5}$$

In considering the potential multiprogramming gain $F(p)$, it is useful to examine the limiting value F_l as p becomes large.

Applying equation (1) to the limiting device m we have

$$F(p)=\frac{X_m}{x_m}$$

and, as p becomes large,

$$F_l=\frac{1}{x_m} \tag{6}$$

The only way of improving F_l is to reduce the monoprogrammed utilization x_m. But since m is the limiting device and

$$\sum_{i=1}^{N} x_i=1$$

it follows that x_m must have a lower bound of $1/N$, at which point all x_i must be equal to $1/N$ and $F_l=N$. This is the condition for a *balanced* system.

The limiting throughput corresponding to F_l is obtained by substituting in (4) using (5) and (6).

$$R_l=\frac{1}{\sum f_i s_i}\cdot\frac{1}{x_m}$$

By definition

$$x_m=\frac{f_m s_m}{\sum f_i s_i}$$

so that

$$R_l=\frac{1}{f_m s_m} \tag{7}$$

For a balanced system,

$$f_m s_m=f_i s_i=\frac{\sum f_i s_i}{N} \quad \text{for} \quad i=1,\ldots N \tag{8}$$

and from (4) $R_{l\,(\text{balance})}=Nr$

It is important to note that while a balanced system is a necessary and sufficient condition for a maximum potential multiprogramming gain F_l, it is not necessarily an appropriate condition for a maximum throughput R at some finite value of p. This is because of the coupling between $F(p)$ and r.

We shall now use equations (5) to (8) to examine informally the effect, at high and low values of p, of two alternative ways of improving the match between system and workload. They are

(i) to reduce device service times
(ii) to redistribute the load on I/O devices

Effect of reducing service time

Let us consider reducing service time s_j on device j. From (5), r will always be improved, but it will be most improved (for a given percent improvement in s_j) when $f_j s_j$ is largest with respect to $\sum f_i s_i$ i.e. when $j=m$, the limiting device. From (7), if $j\neq m$, R_l is not affected by the change but if $j=m$ then R_l is inversely proportional to the service time.

Thus we may expect that for the limiting device, an improvement in service time will always result in an improvement in throughput but this will be most marked at high values of p. Speeding up other devices will have some effect at low values of p, diminishing to zero as p increases (Fig. 4(a)).

If a limiting device j is speeded up sufficiently, there will appear a new limiting device k. We may deal with this as two successive transitions with $j=m$ and $j\neq m$ separated by the boundary condition $f_j s_j=f_k s_k$.

Redistributing the load

This alternative depends on changing the residence of files so that there is a shift of activity from one device to another. Here we must bear in mind two further constraints. Firstly this is not possible in the case of certain devices, e.g. the CPU in a single CPU system. Secondly, a conservation rule

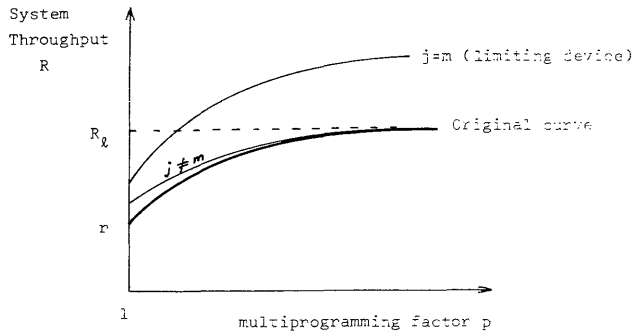


Figure 4(a)—The effect of reducing service time on device j

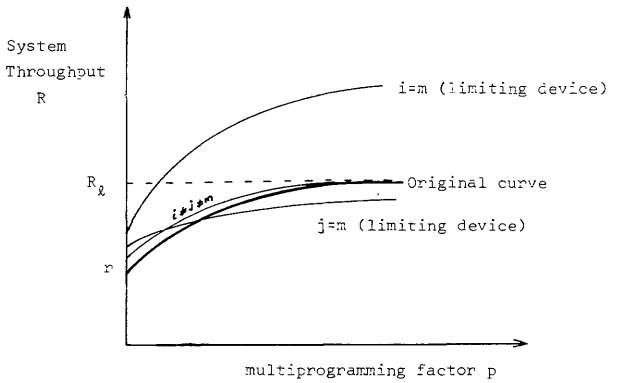


Figure 4(b)—The effect of redistributing the load from device i to a faster device j

usually applies such that work removed from one device i must be added to some other device j.
That is

$$\sum f_i = 1 \tag{9}$$

Consider the effect of switching requests from any device i to a faster device j. Since $s_j < s_i$, equation (5) tells us that r will always be increased. Equation (7) tells us that R_l will be improved while $i=m$, reduced while $j=m$, and unaffected while $i \neq m \neq j$. (Figure 4(b)). The situation is again complicated by the fact that as a particular set of requests is switched from i to j a new limiting device may appear. In this case we may consider the effect as two separate transitions separated by the boundary condition $f_i s_i = f_k s_k$ where k is the new limiting device.

The effect of switching requests to a faster device always results in a throughput improvement at low values of p. As p is increased, the throughput improvement will enlarge if the donor device is the limiting device, diminish to zero if the limiting device is not affected, and become negative if the recipient device is the limiting device.

We should note that moving requests from a faster device j to a slower device i is the exact converse of the above. In particular, if $j=m$, we can expect a throughput improvement at high values of p and a throughput worsening near $p=1$.

In the preceding discussion we have ignored the question of device capacity. In practice, the distribution of files and the capacity of the respective devices must be matched to achieve the most cost-effective I/O load distribution. Consideration of Figure 4(b) will show that the effect of load redistribution depends both on the multiprogramming factor and on how the load on the limiting device is affected by the change. Redistribution is most effective if it reduces the load on the limiting device. Successive applications of this criterion will eventually lead to one of two situations:

either

(1) the CPU becomes the limiting device (CPU-bound situation).

or

(2) the I/O load is balanced among devices so that their respective utilizations are equal and not less than the CPU utilization (balanced I/O-bound situation).

Further load redistribution will be effective at lower values of p, where some gains can be made by moving requests from slower devices to faster ones. At higher values of p this will have a reduced effect in the CPU-bound case and a deleterious effect in the balanced I/O bound case.

One may sum up the criteria for load distribution in a very generalised way as follows. If there is low multiprogramming one should try to *minimise* the I/O load by making maximum use of the faster devices. If there is high multiprogramming and an I/O bound situation, one should try to *balance* the I/O load so that all devices are equally utilized. If there is high multiprogramming and a CPU-bound situation, there is little to be gained by redistributing the I/O load, except perhaps savings on device capacity. Clearly the optimum distribution for a particular case cannot be expressed in such a rule of thumb, and requires detailed calculation.

Use of invariance rules

The effect of a specific change on the behaviour of the system may be understood by applying the invariance rules governing the ratios of request service rates and utilizations on each device. If the service time of a device is reduced (without changing the allocation of files between devices), e.g., by replacing with a faster device or by optimising arm movement, the ratio of utilizations will only change in respect of that device, and the ratio of request rates to all devices will remain constant. However, the magnitude of all request rates will increase. In redistributing the load, the same logic applies, except that the ratios of both request rates and utilizations will change with respect to the affected devices.

Determining $F(p)$

The precise shape of $F(p)$ for given device utilizations depends upon the assumptions we make about the distributions of service times on the various devices.

The simulation results presented so far assume exponential distributions and in this case we may use the formulation developed by Berners-Lee. In Reference 5 he shows that

$$F(p) = S(p-1)/S(p) \tag{4}$$

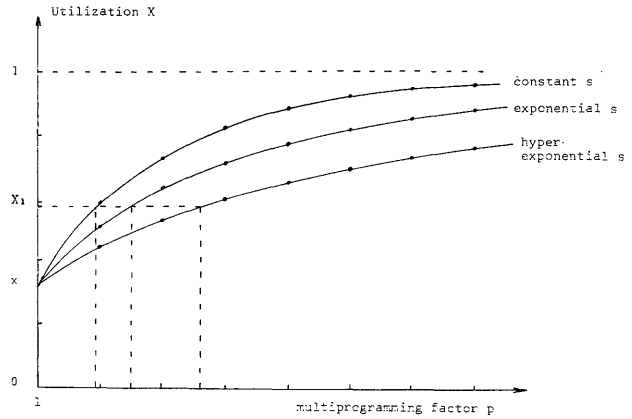


Figure 5—Different cpu utilization curves $X=xF(p)$ obtained with different distributions of service time s. (The same distribution was used on all devices)

where

$$S(p) = \sum x_1^{n_1} x_2^{n_2} \ldots x_N^{n_N} \qquad (5)$$

summed over all combinations of n_i

subject to $n_i \geq 0$ and $n_1 + n_2 + \ldots n_N = p$

The n_i thus represent all possible combinations of queue sizes at each device i.

Using this relationship, $F(p)$ can be economically computed by a simple program provided the x_i are known, and p and N are small.

In practice we may monitor the actual device utilizations X_i for a real system, and compute the x_i from the relation

$$x_i = \frac{X_i}{\sum_{i=1}^{N} X_i} \text{ obtained from (1) and (3)}$$

We have found this program to be more convenient than the simulation when exponential distributions can be assumed. It gives precisely the same results.

Effect of service time distributions

Different shapes of $F(p)$ are obtained by repeating the simulation under different assumptions about the distribution of request service times. Figure 5 shows that an observed utilization X_1 will imply different values of the effective number of processes p under the different assumptions. Although we can estimate a reasonable range for the true value of p there is no means of observing it directly since it is a model parameter based on simplifying assumptions. It is therefore important in using the model to establish its sensitivity to changes in distributions and to establish what distributions actually obtain in practice.

The exponential assumption has yielded results borne out by experiment in the two cases to be described, but it is felt that there must certainly be many cases where this assumption will prove too simplified.

From the figure we may infer that for a given number of parallel processes p, the effectiveness of multiprogramming is increased as the variance of the request service times is reduced. This is our third way of improving throughput.

Variance is an important control parameter in the model, and allows us to reproduce in a general way many practical effects. Some examples of these are:

- the continuously changing pattern of real workloads
- alternative queuing disciplines and time-slicing
- movable-head discs with wide ranges of service times

Work continues on establishing the effects of variance in more detail.

Interactive loads

Within its own terms of reference the model seems to be quite applicable to interactive loads, but applications studied so far have had only a small interactive element so it is too early to be conclusive about this.

We have already stressed that in its present form the model cannot handle questions involving response time. If interactive loads are to be properly considered, it is necessary to introduce core allocation and paging or swapping explicitly. Many studies have of course been done in this area. One which is rather close to the present approach is described by Florkowski.[6]

APPLICATIONS

Expansion of core-store

During the Spring of 1971 an evaluation was attempted to predict the effect on batch throughput of increasing the primary core store. The eventual decision to order more core was based upon the usual combination of hearsay, ad hoc reasoning, and benchmark tests. In this case a certain amount of core went into the construction of the benchmark[4] and it was run under software monitoring. However the insights described in this paper had not then been obtained. More than one year later we reexamined the records of those tests to determine how consistent they were with the behaviour of the model.

In the original core store of 128 k words, the operating system EXEC 8 used about half of the space, leaving approximately 64 k words available for user programs. To predict the effect of adding a core module of 64 k words, a benchmark was run at Computas A/S, Oslo, Norway on a similar configuration equipped with 192 k words. For the sake of comparison, the benchmark was run at the same installation with one core module down. In both cases a version of EXEC 8 including the Wisconsin modifications was used.

Table I (a) shows the subsystem utilizations obtained with the smaller core. From this table we observe that no subsystem has a higher utilization then 0.59. As we have dis-

TABLE I—Expansion of Core-Store

(a) Device utilization with 128 k words of core.

	CPU	FH432	FH880	FAST-RAND	MAGN. TAPE
BENCHMARK	0.59	0.26	0.26	0.56	0.25

(b) Device utilizations with 192 k words of core.

	CPU	FH432	FH880	FAST-RAND	MAGN. TAPE
MODEL	0.79	0.35	0.35	0.75	0.33
BENCHMARK	0.77	0.39	0.38	0.74	0.35

(c) Device utilizations normalised with respect to the CPU to compare relative device loads in the two benchmark tests.

	CPU	FH432	FH880	FAST-RAND	MAGN. TAPE
128 k words	1.00	0.44	0.44	0.95	0.42
192 k words	1.00	0.51	0.49	0.96	0.45

cussed previously the limit of the utilization of the bottleneck subsystem is 1.00 as p is increased. The system is rather far from this maximum and a natural explanation is that the system has insufficient core store. There are of course other possible explanations:

(i) unsaturated system caused by insufficient backlog
(ii) too few jobs opened simultaneously
(iii) the operating system is not able to make full use of the available core

Alternative (i) is eliminated at once because there is a significant backlog when the benchmark is running except for a short period in the beginning and at the end of the test. Point (ii) is eliminated if the mean storage requirements of open runs exceed the available core. In this case the number of open runs was set to 4, and the mean program size was $22k$, giving a product in excess of the available 64 k. Point (iii) is more difficult to handle. The safest way of investigating this alternative is to run a benchmark at an installation having bigger core store. In the following we assume that the batch throughput is limited by the core store and we will use the model to predict the effect of adding 64 k words.

The system is represented by the queuing network shown in Figure 1.

We assume negative exponential distributed service-times and a "first in first out" queuing discipline. Under these assumptions the model (in this case the analytical model is used) gives the results shown in Figure 6.

When adding 64 k words of core, the user core space is approximately doubled so that it is reasonable to expect a system working at a multiprogramming factor equal to twice the old one. Although the multiprogramming factor includes the EXEC activity as well as the user-activity, it is assumed that a certain increase in the user activity causes a corresponding increase in the EXEC activity.

From Figure 6 we observe that a value of 2.7 has to be used for the multiprogramming factor to match the model output to the figures measured in benchmark MARK1 given in Table I (a). From Figure 6 we also find that if the multiprogramming factor is increased to 5.4 the cpu utilization is increased to 0.79. The corresponding relative increase in throughput is given by

$$\frac{0.79-0.59}{0.59}=0.34(\pm 0.02)$$

The increase in throughput measured by the benchmark was 0.28 ($\pm$0.05) based on elapsed time. The large error margin is due to an end-effect in the small core test whereby one job was running alone for 1.7 minutes out of total test time of 24 minutes. In Table I (b) a comparison is given between the subsystem utilizations measured in the large core test and the corresponding figures given by the model.

The improvement 'predicted' by the model seems a little too optimistic, perhaps because our estimation of the new value of p is too high. However, in view of practical difficulties with the benchmark tests, the correspondence seems as close as can be expected.

In Table I (c) we given the utilization figures for the two benchmark tests. The figures are normalized using the cpu utilization as a base. All normalized utilization figures for the benchmark test run on 192 k words core are higher than those for 128 k. This observation is partly explained by looking at the total cpu-time in the two benchmarks. In the first case it is 14.15 minutes, in the second it is 13.90 minutes. By further examination of the monitor output we also find differences in both number of accesses and service-time for the different I/O-subsystems between the two tests. The differences are difficult to explain from the available data.

These discrepancies show that care must be taken in using either a benchmark or a model. When using a simple model to investigate the effect of system-changes it is easy to overlook significant side effects. On the other hand when using a benchmark one runs the risk of including undesirable effects

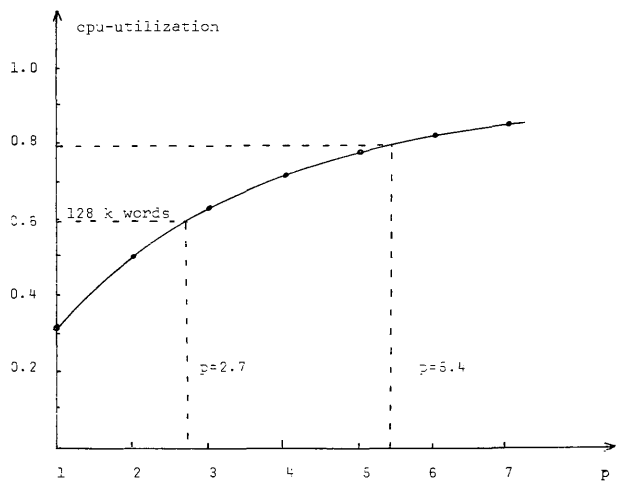


Figure 6—Cpu utilization as a function of the multiprogramming factor. Workload defined by benchmark run with 128 k words core

due to changed test conditions. In both cases emphasis must be placed on careful analysis of the expected effects of a change and, in the case of a benchmark, on the monitor output from the test. A combination of the two techniques is clearly the safest.

Redistributing the load

After the addition of core store which coincided with the introduction of a new version of the operating system, it became apparent that the Fastrand drum, a movable-head, secondary storage of some 200 million characters, and 92 ms average access time, was the limiting device. The first reaction was to think in terms of adding a disc system with a much shorter net access time (of the order of 30 ms) and positioning overlap between disc units. The model predicted a throughput improvement of as much as 30 percent with one such arrangement. However, following a report from the University of Wisconsin, we tested the effect of transferring all symbiont activity (i.e., 'spooling' of read and print files) from Fastrand to a faster drum, the FH880, which had a very low utilization. It turned out that a very large percentage of Fastrand accesses were made in connection with these files, and some directory accesses seem to have disappeared altogether.

The effect can be seen from a pair of monitored benchmark tests run before and after the change. The tests we shall compare are:

Test A Symbiont files on Fastrand
Test B Symbiont files on FH880 drum

The change is effectively a redistribution of the I/O load so that we may compare the practical tests with the behaviour of the model under the appropriate redistribution of f. In order to use the model realistically, we need ideally an independant estimate of the total number of Fastrand accesses involved in symbiont activity. Since this is not available, we will make use of the observed fall in Fastrand accesses between the two tests, but we will assume that all of the difference is transferred to the FH880 drum, ignoring side-effects that came to light in the course of the tests.

We shall also assume that the mean service times on each device are not effected by the change and that service times are distributed exponentially. Then since mean service times are constant we may use the device utilizations as coefficients of the load distribution for each device.

Step 1 is to use the utilizations of test A shown in line 1 of Table II (a) to determine a value of p for the benchmark. This is obtained from curve (a) in Figure 7 which shows the variation of CPU utilization with p for test A. The curve was obtained from the analytic formulation, using integer values of p. The value of p corresponding to the required CPU utilization is found by interpolation at 4.5.

Step 2 is to calculate the "expected" load distribution, using the observed fall in Fastrand accesses between tests A and B. This is shown in Table II (b). Line 4 of this table shows the expected new load distribution. The corresponding utilization ratio is then calculated from the device utilizations for test A using the relationship.

$$\text{New coefficient} = \text{old coefficient} \times \frac{\text{expected new accesses}}{\text{observed old accesses}}$$

This is shown in line 2 of Table II (a).

Step 3 is to use the new load distribution, stated as a ratio of utilizations at some arbitrary value of p, as input to the model. The resultant utilization curve for the CPU is shown in Figure [7(b)]. Since p for test A is already determined, and is not affected by load distribution, we may read off the expected CPU utilization. From our invariance rule, the other device utilizations are then obtainable using the known ratios of their input values. Expected and observed utilizations are shown in Table II (c).

TABLE II—Redistribution of Load

(a) Device utilizations before redistribution and 'expected' ratios after redistribution.

	CPU	FH432	FH880	FASTRAND	MAGN. TAPE
Observed utilization (test A)	0.62	0.26	0.14	0.90	0.18
Expected ratios (test B)	0.62	0.26	0.24	0.57	0.18

(b) Device Requests—observed and 'expected'.

	CPU	FH432	FH880	FASTRAND	MAGN. TAPE
1. Observed (test A)	67024	33626	7351	14763	11284
2. Observed (test B)	62210	30473	11129	9326	11282
3. Observed shift	−4814	−3153	+3778	−5437	−2
4. Expected shift	0	0	(+5437)	−5437	0
5. Expected load (test B)	67024	33626	12788	9326	11284

(c) Device utilizations after re-distribution—Model 'prediction' and benchmark.

	CPU	FH432	FH880	FASTRAND	MAGN. TAPE
Model 'prediction'	0.77	0.32	0.30	0.71	0.22
Observed (test B)	0.74	0.29	0.27	0.68	0.22

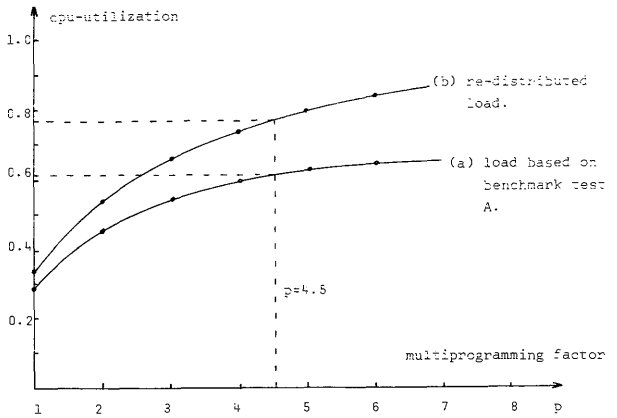


Figure 7(a)—Deduction of p from observed cpu utilization
Figure 7(b)—Deduction of cpu utilization for re-distributed load

The expected improvement in throughput is given by the relative improvement in CPU utilization

$$\frac{0.77-0.62}{0.62} = 0.24\ (\pm 0.02)$$

The increase in throughput measured by the benchmark is 0.27 ($\pm$0.03) based on elapsed time. There is again an error margin in the benchmark figure because of the appearance of an end-effect in test B in which only one job was active for the last 1.1 minutes. The correspondence between benchmark and model results is surprisingly good, in view of the known side effects and discrepancies. We will now review these briefly.

Side-effects

In applying the model to this case, we have deliberately ignored known side-effects of the change (Table II, line (3)). This is because we wished to show how the model may be used in practice to make a prediction where side-effects would not be known. Provided that the loads on the old limiting device and the new limiting device are correct, other side effects may be quite large without making a significant difference to overall performance.

In the present case these were as follows

(i) Effect of full Fastrand.
The Fastrand device at the installation normally operates in a heavily loaded condition, with about 80 percent of its space taken up with catalogued files and heavy competition for the remaining 20 percent by jobs using temporary files. In these circumstances the acquisition of space for a new file seems to require a great deal of extra directory activity as the available space is randomly spread over the device. This situation does not pertain on 880 where there is a much smaller total area to search. Consequently, of the 5437 accesses removed from Fastrand only 3778 were observed on 880, the remainder being directory accesses that were no longer required. This is consistent with independant benchmark tests which show that the net total of Fastrand accesses is 1800-2000 less on an empty Fastrand than on a full one. Since Read and Print files comprise the majority of transient Fastrand files used by the benchmark, we would expect to lose a correspondingly high proportion of these accesses.

(ii) Reduced 432 accesses.
This effect has not been satisfactorily explained.

(iii) Reduced CPU requests.
This effect is really a consequence of the net reduction in I/O accesses. CPU requests are not monitored directly but it is implicit in the model that the sum of CPU requests is equal to the sum of all I/O requests.

Discrepancies in the benchmark

There are always practical difficulties in obtaining two precisely comparable tests with an alteration only in the test variable. We have already referred to the end-effect in test B. There was also a 3 percent loss of CPU time due to a program going into error. Other slight discrepancies in activity charged to the user amounted to $\pm$3 percent of the total load on FH432 and FH880. Another important end-effect is the size of the residual print-backlog at the end of each test. The faster test leaves a longer back-log which also affects the observed shift of accesses to FH880.

In spite of the discrepancies, the model 'prediction' is quite good because we were able to use good estimates for the relative loads on the old and new limiting devices. The complementary nature of the model and benchmark techniques is again evident, each compensating for the weaknesses of the other.

STRUCTURAL PERSPECTIVE

Factors affecting the basic parameters

So far we have examined system and model behaviour in terms of the basic parameters introduced. We shall now take a more general look at some of the factors which affect the values of these parameters.

The average number of active processes p is influenced by

(1) the primary storage requirements of programs
(2) the number of programs opened in parallel by the system
(3) the management of primary storage, including the amount devoted to system resident
(4) the amount of primary storage available.

The distribution of service time s for a device is dependent on both device speed and the work to be performed.

For I/O devices, service time is influenced by

(1) the no. of bits to be transferred (e.g. block length or buffer size)

(2) logical address within the file
(3) disposition of the file on some device
(4) the previous position of the read/write heads (if a movable head device)
(5) positioning time, latency and transfer rate of the device, as appropriate

For a CPU, service time is influenced by

(1) the particular instructions, registers and storage banks involved
(2) the time-slice permitted by the dynamic scheduler
(3) the speed of the CPU logic and storage
(4) cycle-stealing

Finally, there is the distribution f of requests among devices. Factors affecting this are:

(1) the basic processing requirements of the user programs
(2) the additional administrative load imposed by the system: e.g. swapping, directory look-up, transfer of non-resident executive functions, compiler scratch files, program changes etc.
(3) the allocation of both user and system files to specific I/O devices.

In the case of each of these sets of factors we may distinguish a number of levels at which decisions are taken which eventually affect the value of the physical load parameters. These levels are described in Figure 8. Starting with the highest level, they may be designated:

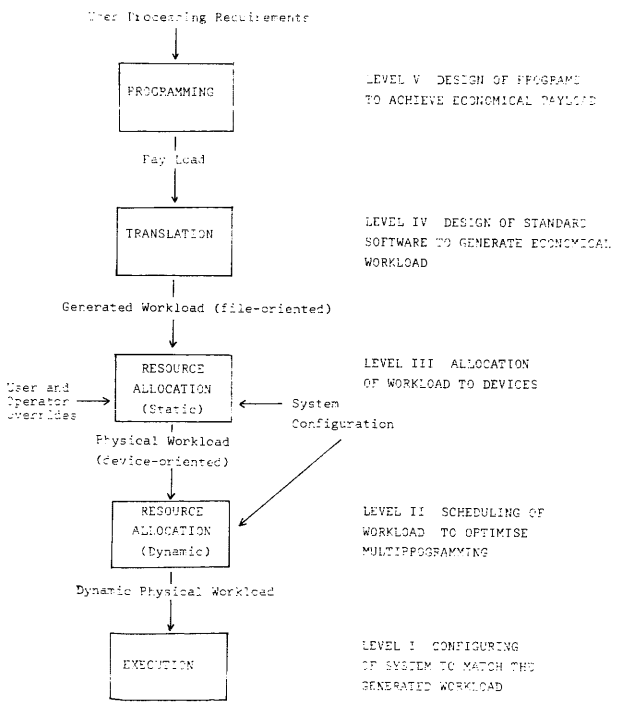


Figure 8—Structural view of performance analysis

	Decision Level	*Associated Input*
5	Programming	User Processing Requirements
4	Translation	User Payload
3	Static Resource Allocation	Generated Workload
2	Dynamic Resource Allocation	Physical Workload
1	Hardware Characteristics	Dynamic Workload

Before dealing with each level in turn, some general comments are in order. Firstly, in existing systems these levels are hopelessly confused, especially the higher ones. Secondly, these levels are not a sequential progression for any given program, but rather reflect different kinds of decision which should be clearly distinguished. Clearly, decisions about static resource allocation are taken at a great many different points in the life of a program, some by the user or programmer, some by the installation, and some by the system itself at run time. Thirdly, the purpose behind making these levels distinct is to clarify how different system components contribute to system performance, what decisions can be taken at what level to affect it, and to distinguish clearly between different kinds of performance question—e.g., computer selection, the adoption of new levels of the operating system, compiler optimization, disc scheduling, hardware configuration. From the point of view of a particular installation, the cost-benefit of a given change must depend upon the contribution it makes to overall system performance through the basic parameters we have presented.

Level 5 programming

At this level the users processing requirements are given explicit shape in some programming language, or by the selection of a particular application package. Processing is defined in terms of records, files and algorithms. The net result from this level of decision making is the identifiable workload of the computer. This is the 'payload', which the user wishes to pay for and which the installation seeks to charge for. Unfortunately, accounting information provided by the system is often only available on a device-oriented rather then file-oriented basis, and compounded with effects from lower levels in the system.

This is the level at which benchmarks are prepared for purposes of computer selection or testing of alternative configurations. In the case of computer selection, the aim must be to withhold as many decisions as possible about how the users requirements will be implemented, since this is so dependent upon available facilities. The alternative is to follow up the ramifications of each system. Thus in general, a quantitative approach to computer selection is either inaccurate or costly, or both.

Benchmarks for a given range of computers are a more feasible proposition since they can be constructed on the output of level 5, leaving open as many resource allocation

decisions as possible, so that the most can be made of alternative configurations.

Level 4 translation

By this is intended the whole range of software including input/output routines and compilers by which the punched instructions of the programmer are translated into specific machine orders and the results translated back again into printed output.

Factors involved at this level will include optimization of code, efficiency of compilers, buffer sizes determined by standard I/O routines. It is clear that these will in turn affect the core size of programs, the length of CPU requests and the number and length of accesses to different files. The output from this stage is the generated workload, and its relation to the payload might be thought of as the 'efficiency' of the system software. Improving this efficiency is the fourth way of improving system performance, referred to in a previous section.

Level 3 static resource allocation

At this level, decisions are concerned with matching processing requirements to the capacity of the configuration. Decisions are made about the allocation of files to specific devices, taking account of configuration information about physical device capacity, number of units, etc. We also include here the decision to 'open' a job, implying the assignment of temporary resources required for its execution. This is done by the coarse scheduling routines, which also decide on the number and type of jobs to be simultaneously open and hence the job mix which is obtained. Both user and operator may override such decisions, subordinating machine efficiency to human convenience.

The decisions taken at this level may influence the maximum number of parallel processes and the relative activity on different I/O devices.

Level 2 dynamic resource allocation

By 'dynamic' we mean decisions taken in real time about time-shared equipment, namely the CPU and primary store. The number of active parallel processes is that number of processes which are simultaneously requesting or using devices in the system. To be active a process must first have primary store, and the dynamic allocation of primary store governs the number of processes active at any instant.

Given the processes with primary store, the system must schedule their service by the CPU, which in turn gives rise to requests for I/O devices. The rules for selection among processes and the timeslice that they are allowed will influence the instantaneous load on devices. In terms of the model, this will influence the shape of the distributions of load f and service time s which in turn influence the shape of the gain function $F(p)$.

Level 1 execution

At this level, the final load has been determined, so that the remaining effects on performance are due to the physical characteristics of the devices. Unfortunately, it is difficult to express the load in terms which are independent of the specific device. The function f gives the distributions of requests, but the service time s is a compound of load and device speed as we have discussed. However, it is at least a quantity which can be directly monitored for a given workload and configuration, and one may estimate how it is affected by changes in device characteristics.

In principle, the important parameters of our model can be monitored directly at this level of the system, although we have not yet succeeded in obtaining an empirical value for p. However, the model depends for its simplicity and power on the correct use of the *distributions* of these parameters and investigations continue in this area.

CONCLUSION

We have presented a set of concepts which have been developed in an effort to master the performance characteristics of a complex computer system. These concepts, together with the simple queuing model which enables us to handle them, have proven their usefulness in a variety of practical situations, some of which have been described.

The application of these concepts depends upon having the necessary information provided by monitoring techniques, and conversely provides insight in the selection and interpretation of monitor output. While such abstractions should whenever possible be reinforced by practical tests, such as benchmarks, they in turn provide insight in the interpretation of benchmark results.

In its present form the model is strictly concerned with throughput and is not capable of distinguishing other performance variables such as response time. This severely restricts its usefulness in a timesharing environment, but is very convenient in situations where throughput is of prime concern.

Consideration of the distinct types of decisions made within the computer complex, suggests that it may be possible to assess the effect of different system components on overall performance in terms of their effect on the basic parameters of the model.

It is thought that the approach described may be particularly useful to individual computer installations seeking an effective strategy for performance analysis.

REFERENCES

1. Kimbleton, S. R., "Performance Evaluation—A Structured Approach," *Proceedings AFIPS Spring Joint Computer Conference*, 1972, pp. 411-416.
2. Strauss, J. C., "A Simple Thruput and Response Model of EXEC 8 under Swapping Saturation," *Proceedings AFIPS Fall Joint Computer Conference*, 1971, pp. 39-49.

3. Draper, M. D., Milton, R. C., *UNIVAC 1108 Evaluation Plan*, University of Wisconsin Computer Center, Technical Report No. 13, March 1970.
4. Hughes, P. H., "Developing a Reliable Benchmark for Performance Evaluation," *NordDATA 72 Conference*, Helsinki, 1972, Vol. II, pp. 1259-1284.
5. Berners-Lee, C. M., "Three Analytical Models of Batch Processing Systems," *British Computer Society Conference on Computer Performance*, University of Surrey, Sept. 1972, pp. 43-52.
6. Florkowski, J. H., "Evaluating Advanced Timesharing Systems," *IBM Technical Disclosure Bulletin*, Vol. 14, No. 5, October 1971.

Simulation—A tool for performance evaluation in network computers

by EDWARD K. BOWDON, SR., SANDRA A. MAMRAK and FRED R. SALZ

University of Illinois at Urbana-Champaign
Urbana, Illinois

INTRODUCTION

The success or failure of network computers in today's highly competitive market will be determined by system performance. Consequently, existing network computer configurations are constantly being modified, extended, and hopefully, improved. The key question pertaining to the implementation of proposed changes is "Does the proposed change improve the existing system performance?" Unless techniques are developed for measuring system performance, network computers will remain expensive toys for researchers, instead of becoming cost effective tools for progress.

In order to analyze and evaluate the effects of proposed changes on system performance, we could employ a number of different techniques. One approach would be to modify an existing network by implementing the proposed changes and then run tests. Unfortunately, for complex changes this approach becomes extremely costly both in terms of the designer's time and the programmer's time. In addition, there may be considerable unproductive machine time.

Alternatively, we could construct a mathematical model of the envisioned network using either analytical or simulation techniques. Queueing theory or scheduling theory could be employed to facilitate formulation of the model, but even for simple networks the resulting models tend to become quite complex, and rather stringent simplifying assumptions must be made in order to find solutions. On the other hand, simulation techniques are limited only by the capacity of the computer on which the simulation is performed and the ingenuity of the programmer. Furthermore, the results of the simulation tend to be in a form that is easier to interpret than those of the analytical models.

To be of value, however, a simulation model must be accurate both statistically and functionally. In order to ensure that the analysis of proposed changes based on the simulation results are realistic, the model's performance must be measured against a known quantity: the existing network.

In this paper we present a simulation model for a hypothetical geographically distributed network computer. Since the model was developed for a hypothetical network, we needed to ensure that the results were valid and that no gross errors existed in the model. Our approach was to design a general n node network simulator and then to particularize the input parameters to describe ILLINET (the computer communications network at the University of Illinois). For a given period, system accounting records provided exact details of the resources used by each task in the system including CPU usage, input/output resources used, core region size requested, and total real time in the system. Using the first three of these parameters as input data, we could simulate the fourth. Comparison of the actual real time in the system to the simulated real time in the system authenticated the accuracy of the model. Extrapolating from these results, we could then consider the more general network with reasonable assurance of accurate results.

MODEL DEVELOPMENT

We begin the development of our network model by focusing our attention on ILLINET. This system contains a powerful central computer with copious backup memory which responds to the sporadic demands of varying priorities of decentralized complexes. The satellite complexes illustrated in Figure 1 include:

(1) Simple remote consoles.
(2) Slow I/O.
(3) Faster I/O with an optional small general purpose computer for local housekeeping.
(4) Small general purpose computers for servicing visual display consoles.
(5) Control computers for monitoring and controlling experiments.
(6) Geographically remote satellite computers.

This network was selected for study because it represents many of the philosophies and ideas which enter into the design of any network computer. The problems of interest here include the relative capabilities of the network, identification of specific limitations of the network, and the

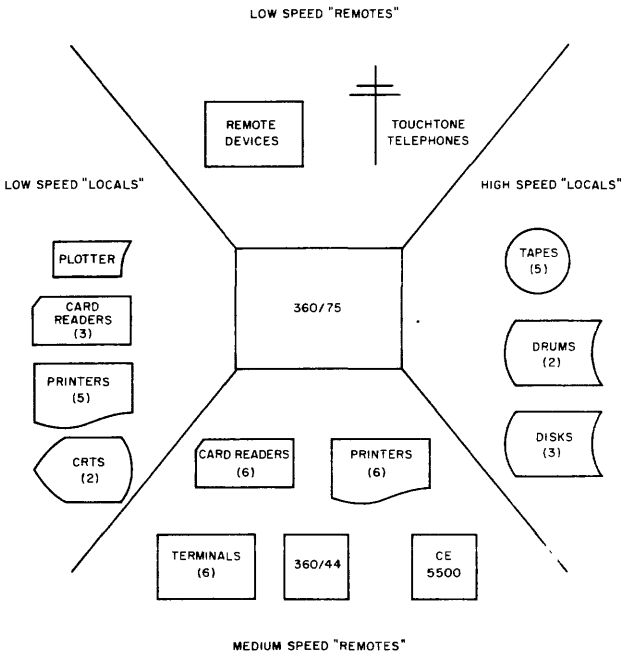


Figure 1—ILLINET—University of Illinois computer communications network

interrelationship between communication and computing. From a long range viewpoint, one of the more interesting problems is the effect on system performance of centralized vs. distributed control in the operating system.

From a postulation of the essential characteristics of our computer network, we have formulated a GPSS model for a three node network, illustrated in Figure 2. Jobs entering the nodes of the network come from three independent job streams, each with its own arrival rate.

A single node was isolated so that performance could be tested and optimized for the individual nodes before proceeding to the entire network. A node in a network

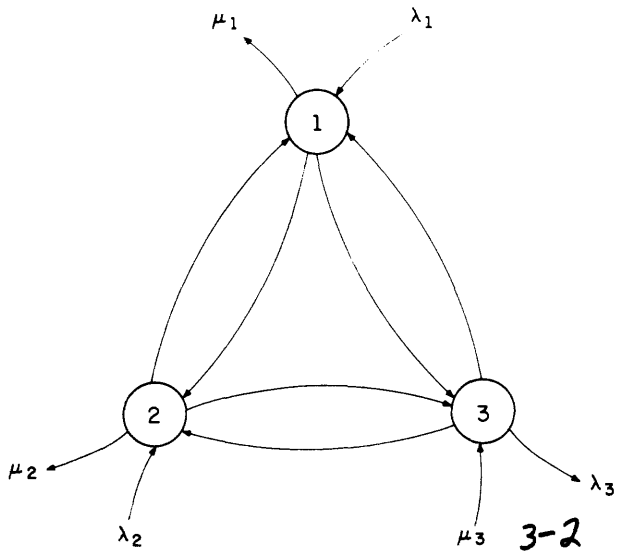


Figure 2—Hypothetical network computer

computer conceptually performs three major functions: queue handling and priority assignment; processor allocation; and resource allocation other than the CPU (such as main storage, input/output devices, etc.).

The goal of the single node optimization was to develop a priority scheme that would minimize the mean flow time of a set of jobs, while maintaining a given level of CPU and memory utilization. The IBM 360/75 was taken as the model node, the present scheduling scheme of the 360/75 under HASP (Houston Automatic Spooling Priority System)[1] was evaluated, and as a result a new priority scheme was devised and analyzed using the simulation model.

NODE DESCRIPTION

The logical structure of the HASP and OS/360 systems currently in use on ILLINET is illustrated in Figure 3 and briefly described in the following paragraphs.

Job initiation

Under the present HASP and O.S. system jobs are read simultaneously from terminals, tapes, readers, disks, and other devices. As a job arrives, it is placed onto the HASP spool (which has a limit of 400 jobs). If the spool is full, either the input unit is detached, or the job is recycled back out to tape to be reread later at a controlled rate.

Upon entering the system, jobs are assigned a "magic number," Y, where the value of Y is determined as follows:

$$Y=SEC+.1*IOREQ+.03*LINES. \quad (1)$$

SEC represents seconds of CPU usage, *LINES* represents printed output, and *IOREQ* represents the transfer to or

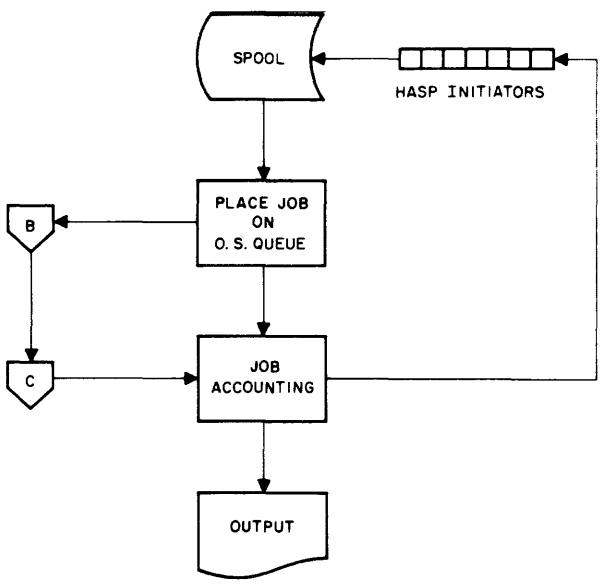


Figure 3a—Logical structure of HASP

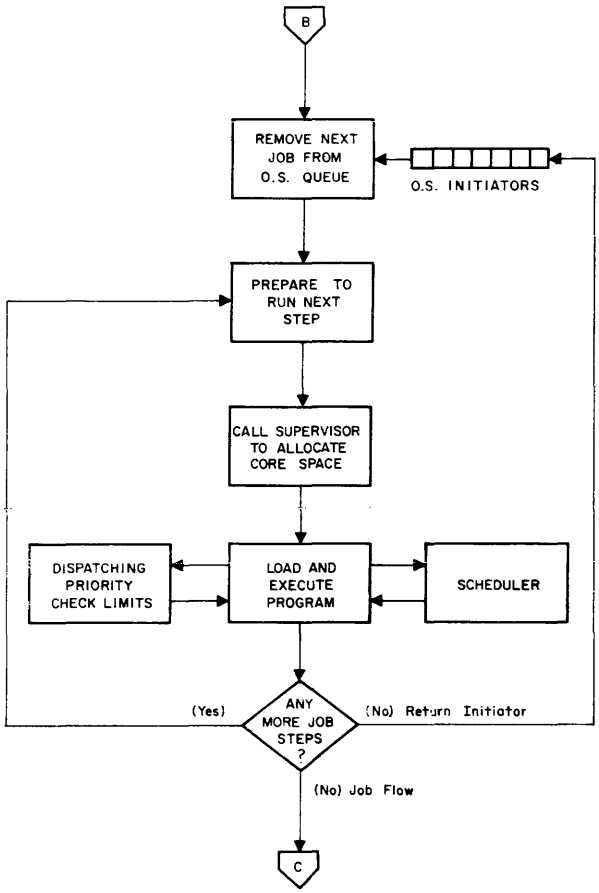


Figure 3b—Logical structure of O.S.

from core storage of blocks of data. Based on this magic number, a "class" assignment is given to each job.

Any one of seven initiators can be set to recognize up to five different classes of jobs, in a specific order. It is in this order that a free initiator will take a job off the spool and feed it to O.S. For example, if an initiator is set CBA, it will first search the spool for a class C job; if not found, it will look for a class B. If there is no B job, and no A job either, the initiator will be put in a wait state. Once the job is selected, it is put on the O.S. queue to be serviced by the operating system.

O.S. initiation

After a job is placed on the O.S. queue, there is no longer any class distinction. Another set of initiators selects jobs on a first-come, first-served basis and removes them from the O.S. queue. It is the function of these initiators to take the job through the various stages of execution.

The control cards for the first (or next) step is scanned for errors, and if everything is satisfactory, data management is called to allocate the devices requested. The initiator waits for completion.

The O.S. supervisor is then called to allocate core space. The first block of contiguous core large enough to contain the step request is allocated to the job. If no such space is available, the initiator must wait, and is therefore tying up both the O.S. and HASP initiators. No procedures in O.S. exist for compacting core to avoid fragmentation. Once core is allocated, the program is loaded, and the job is placed on a ready queue with the highest non-system priority.

O.S. scheduler

Jobs are selectively given control of the CPU by the O.S. scheduler. The job with the highest dispatching priority is given control until an interrupt occurs—either user initiated or system initiated.

HASP dispatcher

Every two seconds, a signal is sent by the dispatcher to interrupt the CPU, if busy. All of the jobs on the ready queue are then reordered by the assignment of new dispatching priorities based on resources used in the previous 2 second interval. The job that has the lowest ratio of CPU time to I/O requests will get the highest dispatching priority. (For example, the jobs that used the least CPU time will tend to get the CPU first on return from the interrupt.) During this period, HASP updates elapsed statistics and checks them against job estimates, terminating the job if any have been exceeded.

Job termination

When execution of the job is completed, control is returned to the HASP initiator to proceed with job termination. Accounting is updated, the progression list is set to mark completion, and Print or Punch service is called to produce the actual output. Purge service is then called to physically remove the job from the system. The initiator is then returned to a free state to select a new job from the spool.

The main goal of the HASP and O.S. system is to minimize the mean flow time and hence the mean waiting time for all jobs in the system, provided that certain checks and balances are taken into account. These include prohibiting long jobs from capturing the CPU during time periods when smaller jobs are vying for CPU time, prohibiting shorter jobs from completely monopolizing the CPU, and keeping a balance of CPU bound and I/O bound jobs in core at any given time. At this point the question was asked: "Could these goals be achieved in a more efficient way?"

PROPOSED PRIORITY SCHEME

In a single server queueing system assuming Poisson arrivals, the shortest-processing-time discipline is optimal

with respect to minimizing mean flow-time (given that arrival and processing times of jobs are not known in advance of their arrivals).[2] This result is also bound to the assumption that jobs are served singly and totally by the server and then released to make room for the next job.

Processing time

With respect to OS/360 there are several levels and points of view from which to define processing or service time. From the user's point of view processing time is, for all practical purposes, the time from which his program is read into HASP to the time when his output has been physically produced—punched, filed, plotted and/or printed. Within this process there are actually three levels of service:

(1) The initial HASP queuing of the job, readying it for O.S.; a single server process in the precise sense of the word.
(2) The O.S. processing of the job; a quasi single server process where the single-server is in fact hopping around among (usually) four different jobs.
(3) The final HASP queueing and outputting of the job; again a true single-server process.

The second level of service was used as a reference point and processing time was defined as the total time a job is under O.S. control, whether it is using the CPU or not. The total time a job is under control of O.S. consists of four time elements:

(1) Waiting for core—this quantity is directly related to the region of core requested by a job and can be represented by $\alpha \cdot R$ where α is a statistical measure of the relationship of core region requests to seconds waiting and R is the region size requested.
(2) Direct CPU usage—this quantity can be measured in seconds by a control clock and is denoted by CPUSEC.
(3) Executing I/O—this quantity includes the time needed for both waiting on an I/O queue and for actually executing I/O. It is directly related to the number of I/O requests a job issues and can be represented by $\beta \cdot IO$ where β is a statistical measure of the relationship of the number of I/O requests to seconds waiting for and executing I/O, and IO is the number if I/O requests issued.
(4) Waiting on the ready queue—this quantity is heavily dependent on the current job configuration. Since the O.S. queue configuration a job encounters is unknown when the job enters HASP, this waiting time is not accounted for in the initial assignment.

The total job processing time, *PRT*, may be expressed as follows:

$$PRT=\alpha \cdot R+CPUSEC+\beta \cdot IO \qquad (2)$$

This number, calculated for each job, becomes an initial priority assignment (the lower the number the higher the job's priority). A summary of the dynamics of the proposed priority scheme is depicted in Figure 4.

Dynamic priority assignment

Once the initial static priority assignment has been determined for each job, a dynamic priority assignment algorithm is used to ensure that the checks and balances listed previously are achieved. The restraints which are enforced by the dynamic priority assignment are needed

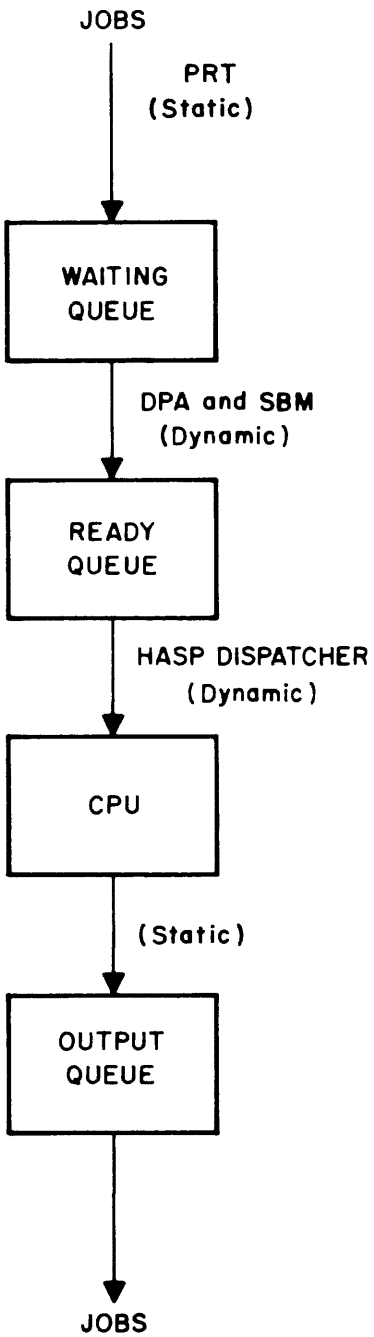


Figure 4 Proposed priority assignment scheme

before a job enters O.S., so the dynamic priority assignment is made while jobs are waiting on the HASP queue. The dynamic priority assignment, DPA, was implemented by measuring a job's waiting time at its current priority. The job's priority is increased if the time spent at the current level exceeds an upper limit established for that level.

System balance

At this point jobs are on the HASP queue, ordered by a dynamically adjusted *PRT* priority assignment, ready to be picked up by an initiator that is free. As soon as an initiator chooses a job, that job leaves HASP control and enters O.S. control. Jobs then move between the O.S. run and ready queues under the influence of the HASP dispatcher discussed earlier. The HASP dispatcher guarantees the highest CPU use level possible, given that the set of jobs has been initiated according to its DPA values. However, CPU and memory utilization may fall below some predetermined level because a particular set of initiated jobs simply does not maintain system balance.[3] Therefore, one more dynamic assignment, SBM—System Balance Measure, was introduced. An SBM assignment enables jobs to take positions in the job queue independently of their DPA. If the utilization of CPU and memory is below a predetermined level, the system is said to be out of balance, and the next job initiated for processing should be the one that best restores balance (where the one with the lowest DPA is chosen in case of ties).

NODE VALIDATION

The proposed priority assignment was then implemented on the simulator and statistics for jobs run on the ILLINET system were collected and used to determine the frequency distributions for the variables needed to create the simulation of the IBM 360/75. The model thus created was used for three distinct purposes. The first of these uses was as a tool to collect data not directly or easily accessible from the actual ILLINET system. It was in this capacity that the simulator yielded α and β factors needed for the *PRT* formula. The second use of the model was as a tuning instrument for finding the best adjustments of values for the DPA and SBM. The third use of the model was for evaluating the advantages (or disadvantages) of the new priority scheme by comparing various system measures for identical job streams run under both the old and new schemes. (A fourth use of the model might also be included here. The convincing results that it provided became the deciding factor in obtaining from the system designers the money and personnel needed to implement and test the proposed priority scheme under real-time conditions.)

Parameter determinations

The first step in the proposed priority scheme was to assign an initial static priority to a job, based on a prediction of how much processing time that job required. The prediction was to be made from user estimates of the resources required by a job. In our previous discussion a formula was devised to make this initial priority assignment, based on user predictions of CPU seconds, kilobytes of core and number of I/O requests and on two statistical measures—α and β. Recall α is a measure of the relationship of core region request to seconds waiting for this request, and β is a measure of the relationship of the number of I/O requests to seconds waiting for and executing this I/O so that

$$PRT=\alpha \cdot COREQ+CPUSEC+\beta \cdot IO \qquad (3)$$

Neither the α nor β measure was immediately available from the present monitoring data of the 360. The simulation was used in two different ways to determine these measures. The particular GPSS simulation being used, while allocating core in the same way as the 360, does not set up all the I/O mechanisms actually used by the 360 when a job issues a request. The simulator assigns some time factor for the request and links the job to a waiting-for-I/O chain for the duration of the assigned time. The approach used in obtaining the β factor was to create a job stream for which the total time in O.S. was known and for which all the components contributing to time in O.S. were known except for the IO factor. Of the four factors that contribute to a job's time in O.S. only actual CPU time could be positively known. A job stream in which jobs did not have to wait for core and in which jobs essentially had the CPU to themselves when they were in a ready state was required. Thus the equation was reduced to:

$$\text{Time in O.S.} = CPUSEC+\beta \cdot IO \qquad (4)$$

where β was the only unknown.

Using a light job stream (average arrival rate of one job every 40 seconds and a CPU utilization of 8 percent with 3 percent of the jobs waiting for core) an exponential distribution of wait times distributed around a mean of 54 msec gave the closest match between the simulated and real O.S. time distributions. β was assigned the value 0.038 seconds per I/O request, since in an exponential distribution 50 percent of the values assigned will be less than 69 percent of the mean. The simulation was then used with a heavier job stream (one job every 15 seconds) for the determination of α. Statistics were produced correlating the size of a step's core request and the number of milliseconds it had to wait to have the request filled. A least squares fit of the data yielded the relationship:

$$WC=\max\{0, \cdot 7K^2-100K\} \qquad (5)$$

where *Wc* is milliseconds of wait time and K is the number of kilobytes core requested. The *PRT*, in its final form thus became:

$$PRT=CPUSEC+38IO+\max\{0, \cdot 7K^2-100K\} \qquad (6)$$

where *CPUSEC* is the number of CPU milliseconds required, IO is the number of I/O requests, and K is kilobytes of core.

Dynamic tuning

The values for the maximum waiting times for jobs with given PRTs were determined by putting the PRTs into a loose correspondence with the existing class divisions. A small job is guaranteed thirty minute turnaround time and a slightly larger job is guaranteed a turnaround of ninety minutes. Since the simulations generally were not set to simulate more than ninety minutes of real time, the guaranteed turnaround for very large jobs was set to an arbitrarily high value. Since test runs using the *PRT* were showing very satisfactory values for CPU and core utilization, about 96 percent and 73 percent respectively, a simple system balance measure was adopted. The SBM, System Balance Measure, adjustment checks CPU and memory use individually every two seconds and signals the initiator to choose as its next job the one that would best restore the utilization of the respective resource. The criterion for resource underuse is less than 30 percent utilization. The criterion for choosing a job to restore CPU use is the highest CPU/IO ratio. The criterion for choosing a job to restore memory use is the largest core request that fits into core at the time.

NODE PERFORMANCE

After the proposed priority scheme was developed, the simulation model was used to evaluate the node performance under each of three conditions:

(1) Using the existing magic number technique.
(2) Using the static *PRT* technique.
(3) Using the dynamically adjusted *PRT* (including the DPA and SBM measures).

For each of these tests, an hour of real time was simulated, with identical job streams entering the system. Table I illustrates the results of these tests including O.S. and turnaround times for the job streams, as well as CPU and core utilization values.

In this evaluation we are particularly interested in three measures of system performance:

(1) Turnaround time—system performance from the user's point of view.
(2) System throughput—system performance from the system manager's point of view.
(3) System balance—system performance from the device utilization point of view.

In particular, we note the striking decrease in overall turnaround time for jobs processed under the proposed *PRT* scheduling algorithms. When the resource utilization is kept above some critical level and a maximum waiting time is specified, we observe that the turnaround time for the entire system can, in fact, increase.

TABLE I—System Performance Measures for Three Priority Schemes

	PRESENT SCHEME	PRT (Static)	PRT (Dynamic)
Mean HASP Time*			
Class A	100	11	28
B	100	8	9
C	100	98	17
D	—	—	—
A–D	100	12	18
Mean O.S. Time*			
Class A	100	74	94
B	100	54	69
C	100	47	91
D	—	—	—
A–D	100	70	87
Mean Turnaround Time*			
A–D	100	22	29
% CPU Utilization	94	96	98
% CORE Utilization	75	73	73
Total Jobs Processed	482	560	515

* Relative time units (times are normalized to 100 units for each priority class).

NETWORK MODELING

Having developed a simulation model for a single node, we now turn to the problem of constructing a model for a network of three such nodes as illustrated in Figure 2. Jobs entering the system come from three independent job streams with different arrival rates. At selected intervals, the relative "busyness" of each center is examined. Based on this information, load-leveling is performed between centers.

The three node network model was written in IBM's GPSS (General Purpose Simulation System),[4] and run on an IBM 360/75. Once the simulation language and computer were selected, the next step was to formulate a design philosophy.

DESIGN PHILOSOPHY

Simulated time unit

A major decision regarding any simulation model is the length of the simulated time unit. A small time unit would be ideal for a computer system simulation. However, other ramifications of this unit must be considered. It is desirable to simulate a relatively long real-time period in order to study the effect of any system modifications. This would be extremely lengthy if too small a time unit were chosen, requiring an excessive amount of computer time. Also, depending on the level of simulation, the accuracy could actually deteriorate as a result of the fine division of time. These and other considerations led to the selection of 1 millisecond as the clock unit.

Using a time unit of 1 millisecond immediately results in the problem of accounting for times less than 1 ms. Of

course, these times could not be ignored, but at the same time, could not be counted as a full clock unit. A compromise approach was used—that of accumulating all of these small pieces into a sum total of "system overhead," to be run during initiation/termination.*

System time chargeable to a job therefore, is executed during initiation/termination. Additionally, nonchargeable overhead is accounted for at each interrupt of a job in the CPU, and at the reordering of dispatching priorities of jobs on the ready queue.

Entity representation

Before proceeding to write a simulation program, careful consideration had to be given to the way in which actual system entities were to be represented by the simulator. The properties of a given system feature to be simulated had to be defined and the GPSS function most closely matching the requirements selected. For example, representation of the HASP Spools was of primary importance, and GPSS offers a number of possibilities—queues, chains, etc. The requirement that transactions be re-ordered at any time ruled out the queue representation, and the optional automatic priority ordering possible with a user chain led to its selection. Chains also offered the best method of communication between nodes of the network since it is possible to scan the chains and remove any specific job. This is essential for the implementation of any load-leveling or system balancing algorithm.

The structure of GPSS played another important part in determining the representation of jobs. A job could have been represented as a table (in the literal programming and not GPSS sense) that would contain the information about this job, and be referenced by all other transactions in the simulation. This would have led to a simulation within the simulation, an undesirable effect. Therefore, jobs are represented as transactions which keep all pertinent information in parameters. Unfortunately, this led to some rather complex timing and communication considerations which had to be resolved before the simulator could run.

Timing and communication

There is no direct method of communication between two transactions in GPSS, so whenever such contact was necessary, alternate procedures were devised. For example, at some points an initiator must know the size of core requested by a given job. The receiving transaction must put itself in a blocked state, while freeing the transaction from which the information is required. The information is then put in a savevalue or other temporary location by the sending transaction. After signalling the receiving transaction that the information is present, this transaction puts itself in a blocked state, and thus allows the receiving transaction to regain control of the simulator in order to pick up the contents of the savevalue. This procedure is non-trivial, since in an event-driven simulation, there may be any number of transactions ready to run when the sending transaction is blocked. The priorities of the ready transactions, and knowledge of the scheduling algorithms of the simulation language itself, must be analyzed to ensure correct results.

During the simulation, the jobs waiting to be executed are not the only transactions waiting to use the simulated CPU. Transactions representing the scheduler and dispatcher also require this facility. Therefore, we must ensure that only one transaction enters the CPU at any given time since this is not a multiprocessing environment. Logic switches are set and facility usage tested by every transaction requesting the CPU.

GPSS IMPLEMENTATION

With this design philosophy, we proceed to outline the representation of the various HASP and O.S. entities at each node. The logical structure of the simulation process occurring at each node, shown in the flow charts of Figures 5 through 8, is summarized in the following paragraphs.[5]

Jobs

Each job is represented by one GPSS transaction with parameters containing information such as creation time, number of milliseconds that will be executed, size of core requested, etc. The parameters are referenced throughout the simulation to keep a record of what was done, and indicate what the next step will be. In this way, by moving the transaction from one section of the model to another, different stages of execution can be indicated.

HASP and O.S. initiators

There are two sets of initiators, one for HASP, and another for O.S., each requiring the same information about the jobs they are servicing. The HASP initiator for a specific job must be dormant while the O.S. initiator is running. Therefore, seven transactions are created, each of which represents either a HASP or O.S. initiator. Each transaction is created as a HASP initiator and put in an inactive state awaiting the arrival of jobs. After the initia-

* It is impossible to measure accurately (within 10 percent) the amount of system time required for a given job. In a period of simulated time T, an error $E_s = T_s N_s e$ will be accumulated, where N_s is the number of relatively short ($\sim$1 ms) amounts of system time needed, T_s is the total time spent on each of these short intervals, and e is the percentage error in the simulated time given to this operation. Similarly, the error for long intervals E_l can be shown to be $T_l N_l e$ where T_l and N_l are as above for some longer periods ($\sim$1000 ms). The simulation shows the ratio $T_s N_s / T_l N_l$ is approximately 2, resulting in a greater error with the smaller time unit.

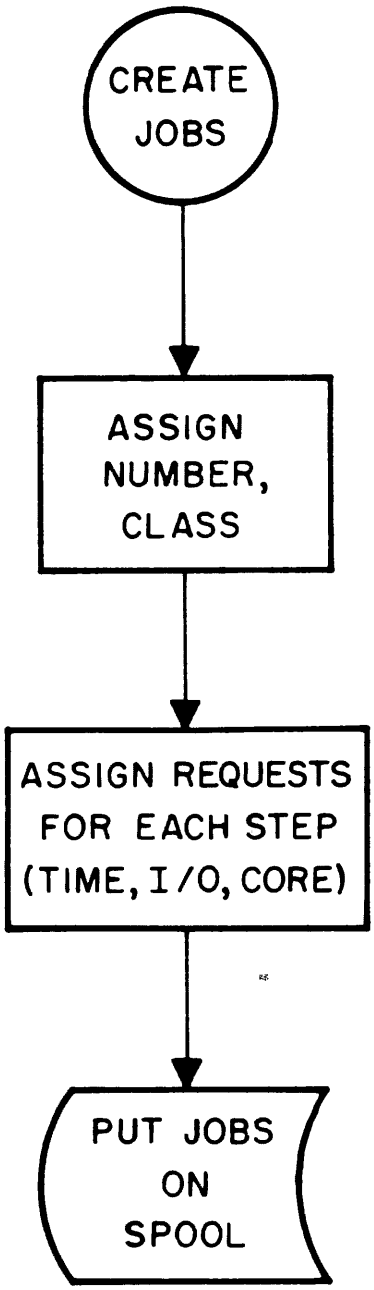


Figure 5—Job creation flow chart

tor completes initiation of a job and places it on the O.S. queue, the HASP initiator becomes the O.S. initiator. This O.S. initiator flows through the core allocation and other resource allocation routines to request core space, and finally places the job on the ready queue to run. This initiator then becomes dormant waiting for the job (or job step) to complete. At each step completion, the initiator is awakened to request resources for the succeeding step. When the entire job completes, the initiator is returned to an inactive state where it again performs its HASP function. Whenever an initiator or job is to be put in an inactive state, it must be taken off the current events chain and placed on a chain specifically representing that wait condition.

Queues

All HASP and O.S. queues are represented by user chains as discussed earlier. In addition to facilitating ordering of objects on the queues, chains gather the proper waiting time and size statistics automatically.

CPU—scheduler

The scheduler has the responsibility of determining which task will next get control of the CPU. The scheduler is represented by one high priority transaction that unlinks jobs from the ready queue and lets them seize the

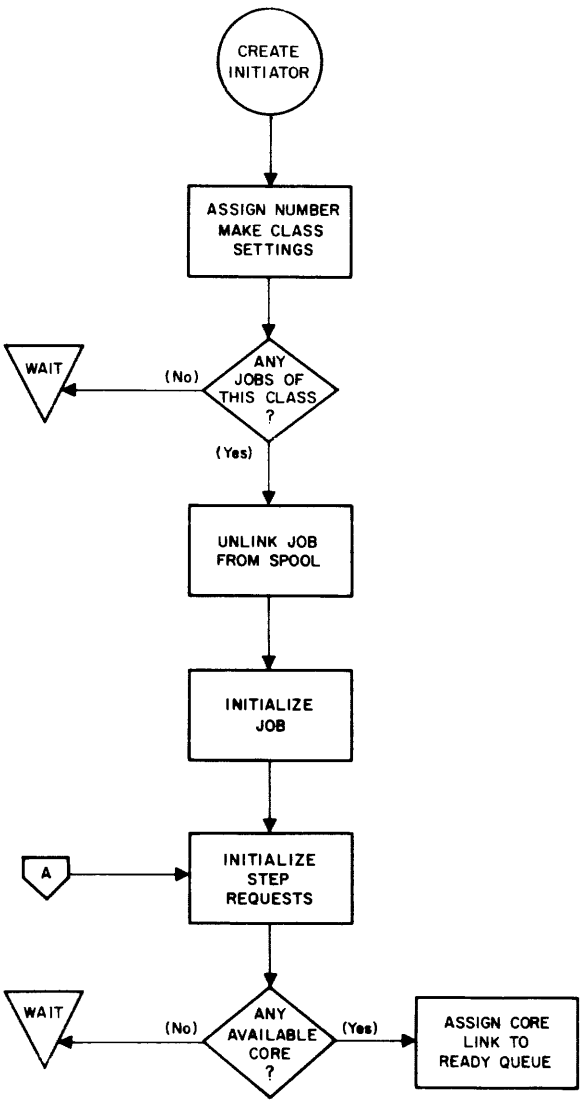


Figure 6—HASP and O.S. initiator flow chart

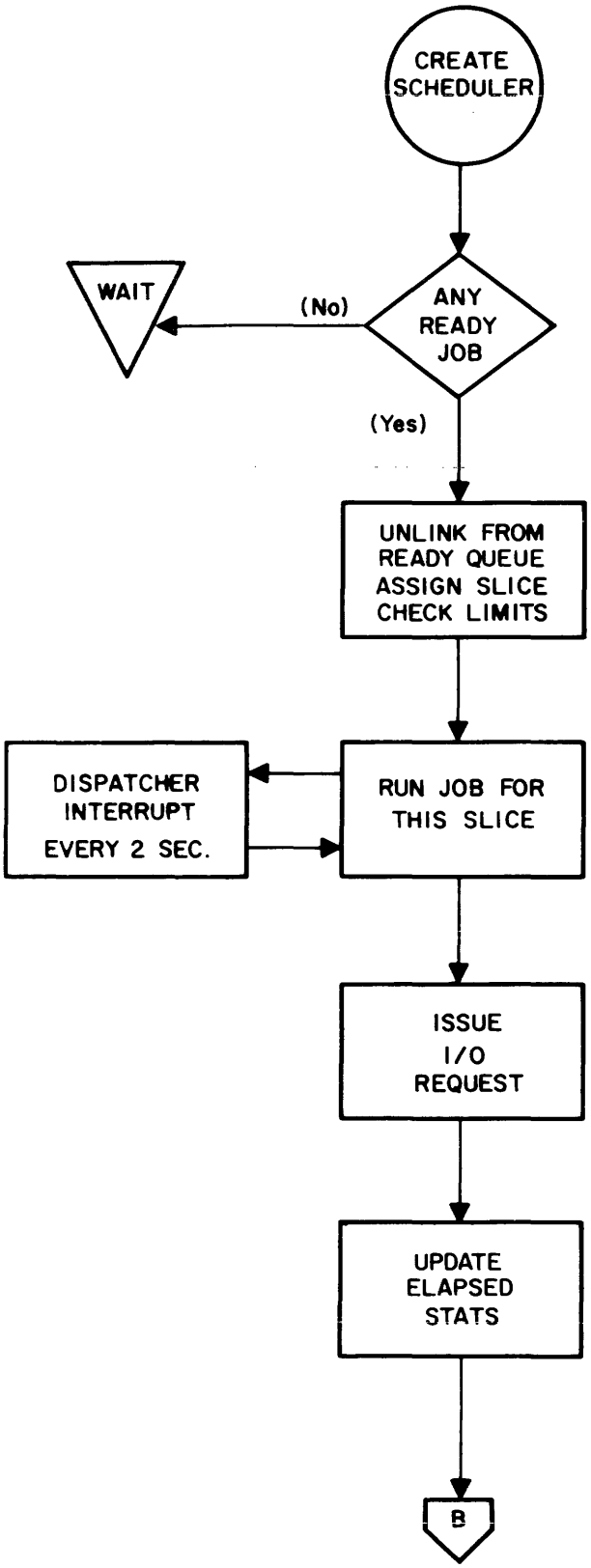


Figure 7a—CPU-Scheduler flow chart

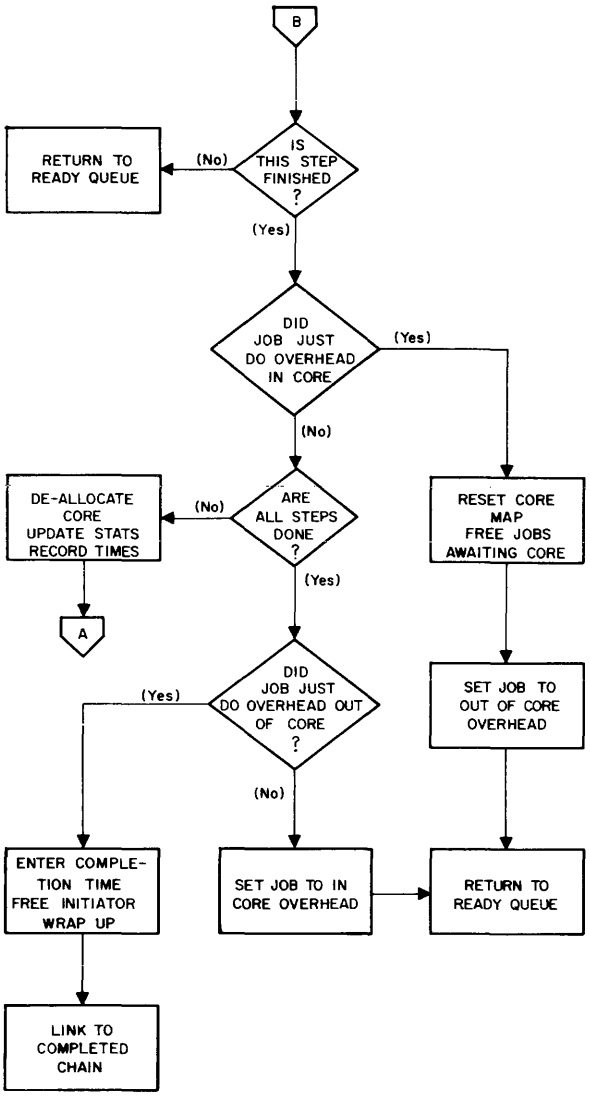


Figure 7b—CPU-Scheduler flow chart (cont.)

facility corresponding to the CPU. While this job is advancing the clock in the facility, no other transactions are permitted to enter. Although GPSS automatically permits only one job in each facility, this is not sufficient protection against more than one transaction entering the CPU. Therefore, this condition is explicitly tested by all transactions requesting the CPU. Multiple transactions are allowed to enter blocks representing I/O requests, and other system processes, since these functions in the real system are actually carried out in parallel. When the CPU is released, control is returned to the scheduler, which allocates the facility to the next job on the ready queue.

Dispatching priority assignment

The HASP dispatching priority assignment is carried out by one final transaction. Every 2 seconds this transac-

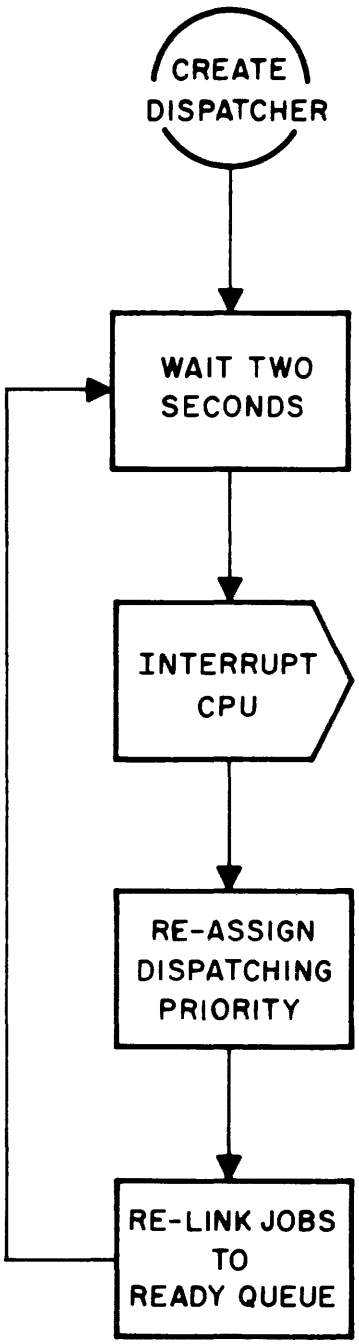


Figure 8—Dispatcher flow chart

tion is given control of the simulator, and proceeds to re-assign the dispatching priority of the jobs on the ready queue and those jobs currently issuing I/O requests. The job in control of the CPU (if any) is interrupted, and placed back on the ready queue according to its new priority.

When all of the re-ordering is complete, the scheduler is freed, and the dispatcher is made dormant for another two seconds.

NETWORK PERFORMANCE

After having tested the simulation model, a hypothetical network consisting of three nodes with independent job streams and arrival rates, was investigated. Network balance was maintained using a load-leveling algorithm on the network that periodically (about five times the arrival rate of jobs at the busiest center) examined the queues at each center. The number of jobs in the queue of the busiest center was compared to the number of jobs in the queue of the least busy center, and this ratio used as the percentage of jobs to be sent from the busiest center to the lightest center. This distributed the number of jobs in the network so that each center would be utilized to the maximum possible degree. Naturally, the users submitting jobs to the highest center experienced an increase in turnaround time, but this is outweighed by the increased throughput for the network.

To demonstrate how the simulator could be used to evaluate loadleveling or other network balancing algorithms, two simulation runs were made: the first having no communication between centers, and the second with the load-leveling algorithms implemented as described above. Performance data for the two networks, evaluated according to the criteria outlined earlier, is illustrated in Table II.

TABLE II—System Performance Measures for a Hypothetical Network

	Without Load Leveling	With Load Leveling
Turnaround Time*		
Center 1	99	89
Center 2	100	80
Center 3	44	95
Network	90	87
Average Queue Length		
Center 1	125	42
Center 2	4	25
Center 3	~0	23
Network	43	30
CPU Utilization		
Center 1	.985	.982
Center 2	.902	.952
Center 3	.518	.931
Network	.802	.955
Core Utilization		
Center 1	.685	.713
Center 2	.698	.684
Center 3	.326	.668
Network	.569	.688
System Throughput**		
Center 1	330	256
Center 2	196	256
Center 3	98	234
Network	624	746

* Relative time units.
** Jobs per hour.

CONCLUSIONS

NETWORK SIMULATION RESULTS

Validation

Our simulation model for a network center is a valid tool for measuring and evaluating network performance *only if* we accurately simulate the intercommunication among the network centers and the control of jobs within each center. Therefore, an essential goal of our simulation effort was to verify the accuracy of representing the interaction among the simulated entities of ILLINET. Frequently, spot checks were made and tests were designed to ensure that the proper correspondence existed between the real and simulated environments. Hence, the evaluation measurements taken, effectively predict the expected system performance of future networks.

System evaluation

The GPSS simulation of the IBM 360/75 was used to develop and test a new priority scheme suggested as an alternative to the present system used on the 360. An initial static priority assignment was determined which uses user estimates of CPU, I/O requests and core required by a job. Subsequent priority adjustments are made to reward a job for its long wait in the system or to restore some minimum level of utilization of the CPU and core. Test runs of the new priority scheme on the simulator suggest very substantial improvements in terms of minimizing turnaround time and utilizing system resources. The emphasis is on evaluating scheduling disciplines since the only degree of freedom open to a network manager to affect system congestion is a choice of scheduling algorithms with priority assignments. (A network manager can seldom affect the arrival rate, service rate or the network configuration on a short term basis.)

The simulation was then extended to a three node network to study the effect of implementing load-leveling and other network balancing algorithms. Simulation runs show an improved turnaround time for heavily loaded centers and at the same time a larger increase in total throughput and utilization of network resources.

THE ROAD AHEAD

Until recently, efforts to measure computer system performance have centered on the measurement of resource (including processor) idle time. A major problem with this philosophy is that it assumes that all tasks are of roughly equal value to the user and, hence, to the operation of the system.

As an alternative to the methods used in the past, we have proposed a priority assignment technique designed to represent the worth of tasks in the system.[6] We present the hypothesis that tasks requiring equivalent use of resources are not necessarily of equivalent worth to the user with respect to time. We would allow the option for the user to specify a "deadline" after which the value of his task would decrease, at a rate which he can specify, to a system determined minimum. Additionally, the user can exercise control over the processing of his task by specifying its reward/cost ratio which, in turn, determines the importance the installation attaches to his requests. The increased flexibility to the user in specifying rewards for meeting deadlines yields increased reward to the center. The most important innovation in this approach is that it allows a computing installation to maximize reward for the use of resources while allowing the user to specify deadlines for his results. The demand by users upon the resources of a computing installation is translated into rewards for the center. Thus, the computing installation becomes cost effective, since, for a given interval of time, the installation can process those tasks which return the maximum reward.

Using our network simulator to demonstrate the efficacy of this technique is the next step in the long road to achieving economic viability in network computers.

ACKNOWLEDGMENTS

We are particularly grateful to Mr. William J. Barr of Bell Laboratories for inviting us to write this paper. His constant encouragement and his previous endeavors in the Department of Computer Science at the University of Illinois at Urbana-Champaign have made this work possible.

This research was supported in part by the National Science Foundation under Grant No. NSF GJ 28289.

REFERENCES

1. Simpson, T. H., *Houston Automatic Spooling Priority System—II (Version 2)*, International Business Machines Corporation, 1969.
2. Schrage, L., "A Proof of the Optimality of the Shortest Remaining Processing Time Discipline," *Operations Research*, Vol. 16, 1968.
3. Denning, P., "The Working Set Model for Program Behavior," *Communications of the ACM*, Vol. 11, No. 5, 1968.
4. ———, *General Purpose Simulation System/360 Introductory User's Manual GH20-0304-4*, International Business Machines Corporation, 1968.
5. Salz, F., *A GPSS Simulation of the 360/75 Under HASP and O.S. 360*, University of Illinois, Report No. UIUCDCS-R72-528, 1972.
6. Bowdon, E. K., Sr., and Barr, W. J., "Cost Effective Priority Assignment in Network Computers," *Proceedings of the Fall Joint Computer Conference*, 1972.

ACCNET—A corporate computer network

by MICHAEL L. COLEMAN

Aluminum Company of America
Pittsburgh, Pennsylvania

INTRODUCTION

The installation of a Digital Equipment Corporation DEC 10, in close proximity to an existing IBM 370/165, initiated an investigation into the techniques of supporting communication between the two machines. The method chosen, use a mini-computer as an interface, suggested the possibility of broadening the investigation into a study of computer networks—the linking of several large computer systems by means of interconnected mini-computers. This paper explains the concept of a network and gives examples of existing networks. It discusses the justifications for a corporate computer network, outlines a proposed stage by stage development, and analyzes and proposes solutions for several of the problems inherent in such a network. These include: software and hardware interfaces, movement of files between dissimilar machines, and file security.

WHAT IS A NETWORK?

A computer network is defined to be "an interconnected set of dependent or independent computer systems which communicate with each other in order to share certain resources such as programs or data—and/or for load sharing and reliability reasons."[19] In a university or a research environment, the network might consist of interconnected time-sharing computers with a design goal of providing efficient access to large CPUs by a user at a terminal. In a commercial environment a network would consist primarily of interconnected batch processing machines with a goal of efficiently processing a large number of programs on a production basis. One example of the use of a network in a commercial environment would be preparing a program deck on one computer, transmitting it to another computer for processing, and transmitting the results back to the first computer for output on a printer.

OTHER NETWORKS

Functioning networks have been in existence for several years.[4,19,36] These include: CYBERNET, a large commercial network consisting of interconnected Control Data Corporation machines;[38] the Distributed Computer System (DCS) at the University of California at Irvine;[18] the Michigan Educational Research Information Triad, Inc. (MERIT), a joint venture between Michigan State University, Wayne State University, and the University of Michigan;[2,12,30] the OCTOPUS System at the Lawrence Berkeley Laboratory;[41] the Triangle Universities Computation Center (TUCC) Network, a joint undertaking of the Duke, North Carolina State, and North Carolina Universities;[54] ad the TSS Network, consisting of interconnected IBM 360/67s.[39,47,53] But perhaps the most sophisticated network in existence today is the one created by the Advanced Research Projects Agency (ARPA), referred to as the ARPA network.[9,15,20,22,28,33,34,40,42,44,46] The ARPA network is designed to interconnect a number of various large time-shared computers (called Hosts) so that a user can access and run a program on a distant computer through a terminal connected to his local computer. It is set up as a message service where any computer can submit a message destined for another computer and be sure it will be delivered promptly and correctly. A conversation between two computers has messages going back and forth similar to the types of messages between a user console and a computer on a time-shared system. Each Host is connected to the network by a mini-computer called an Interface Message Processor (IMP). A message is passed from a Host to its IMP, then from IMP to IMP until it arrives at the IMP serving the distant Host who passes it to its Host. Reliability has been achieved by efficient error checking of each message and each message can be routed along two physically separate paths to protect against total line failures.

The ARPA network was designed to give an end-to-end transmission delay of less than half a second. Design estimates were that the average traffic between each pair of Hosts on the network would be .5 to 2 kilobits per second with a variation between 0 and 10 kilobits per second and the total traffic on the network would be between 200 and 800 kilobits per second for a 20 IMP network.[20] To handle this load, the IMPs were interconnected by leased 50KB lines.

For the initial configuration of the ARPA network, communication circuits cost $49,000 per node per year and the network supports an average traffic of 17 kilobits

per node. Each IMP costs about $45,000 and the cost of the interface hardware is an additional $10,000 to $15,000.[23] The IMPs are ruggedized and are expected to have a mean time between failures of at least 10,000 hours—less than one failure per year. They have no mass storage devices and thus provide no long term message storage or message accounting. This results in lower cost, less down time, and greater throughput performance.[46]

TYPES OF NETWORKS

There are three major types of networks: Centralized, Distributed, and Mixed.[19]

A Centralized network is often called a "Star" network because the various machines are interconnected through a central unit. A network of this type either requires that the capabilities of the central unit far surpass those of the peripheral units or it requires that the central unit does little more than switch the various messages between the other units. The major disadvantage of a network of this type is the sensitivity of the network to failures in the central unit, i.e., whenever the central unit fails, no communication can occur. The most common example of this type of network is one consisting of a single CPU linked to several remote batch terminals.

A Distributed network has no "master" unit. Rather, the responsibility for communication is shared among the members; a message may pass through several members of the network before reaching its final destination. For reliability each unit in the network may be connected to at least two other units so that communication may continue on alternate paths if a line between two units is out. Even if an entire unit is disabled, unaffected members can continue to operate and, as long as an operable link remains, some communication can still occur. The ARPA network is an example of a Distributed network.

A Mixed network is basically a distributed network with attached remote processors (in most cases, batch terminals) providing network access to certain locations not needing the capability of an entire locally operated computer system. These remote locations are then dependent on the availability of various central CPUs in order to communicate with other locations.

Within a network, two types of message switching may occur: circuit switching and packet switching. Circuit switching is defined as a technique of establishing a complete path between two parties for as long as they wish to communicate and is comparable to the telephone network. Packet switching is breaking the communication into small messages or packets, attaching to each packet of information its source, destination, and identification, and sending each of these packets off independently to find its way to the destination. In circuit switching, all conflict and allocation of resources must be resolved before the circuit can be established thereby permitting the traffic to flow with no conflict. In packet switching, there is no dedication of resources and conflict resolution occurs during the actual flow. This may result in somewhat uneven delays being encountered by the traffic.[15]

WHY A NETWORK?

By examining the general characteristics of a network in the light of a corporate environment, specific capabilities which provide justification for the establishment of a corporate computer network can be itemized.[25] These are:

load balancing
avoidance of data duplication
avoidance of software duplication
increased flexibility
simplification of file backup
reduction of communication costs
ability to combine facilities
simplification of conversion to remote batch terminal
enhancement of file security

Load balancing

If a network has several similar machines among its members, load balancing may be achieved by running a particular program on the machine with the lightest load. This is especially useful for program testing, e.g., a COBOL compilation could be done on any IBM machine in the network and achieve identical results. Additionally, if duplicate copies of production software were maintained, programs could be run on various machines of the network depending on observed loads.

Avoidance of data duplication

In a network, it is possible to access data stored on one machine from a program executing on another machine. This avoids costly duplication of various files that would be used at various locations within the corporation.

Avoidance of software duplication

Executing programs on a remote CPU with data supplied from a local CPU may, in many cases, avoid costly software duplication on dissimilar machines. For example, a sophisticated mathematical programming system is in existence for the IBM 370. With a network, a user could conversationally create the input data on a DEC 10 and cause it to be executed on the 370. Without a network, the user would either have to use a more limited program, travel to the 370 site, or modify the system to run on his own computer.

Flexibility

Without a network each computer center in the corporation is forced to re-create all the software and data files it wishes to utilize. In many cases, this involves complete reprogramming of software or reformatting of the data files. This duplication is extremely costly and has led to considerable pressure for the use of identical hardware

and software systems within the corporation. With a successful network, this problem is drastically reduced by allowing more flexibility in the choice of components for the system.

Simplification of file backup

In a network, file backup can be achieved automatically by causing the programs which update the file to create a duplicate record to be transmitted to a remote machine where they could be applied to a copy of the data base or stacked on a tape for batch update. This would eliminate the tedious procedure of manually transporting data from one machine to another; the resulting inherent delay in the updates would be eliminated.[11]

Reduction of communication costs

The substitution of a high bandwidth channel between two separate locations for several low bandwidth channels can, in certain cases, reduce communication costs significantly.

Ability to combine facilities

With a network, it is possible to combine the facilities found on different machines and achieve a system with more capability than the separate components have individually. For example, we could have efficient human interaction on one machine combined with a computational ability of a second machine combined with the capability of a third machine to handle massive data bases.

Simplification of conversion

Converting a site from its own computer to a remote batch terminal could be simplified by linking the computer at the site into the network during the conversion.

Enhancement of file security

By causing all references to files which are accessible from the network to go through a standard procedure, advanced file security at a higher level than is currently provided by existing operating systems may be achieved. This will allow controlled access to records at the element level rather than at the file level.

EXISTING SITUATION

The existing configuration of the DEC 10 installation provides a 300 (to be extended to 1200) baud link to the 370 via a COMTEN/60, a mini-computer based system which provides store-and-forward message switching capability for the corporate teletype network. This link is adequate to support the immediate needs of a Sales Order Entry System but is totally inadequate for the general capability of making the computational power and the massive file storage of the 370 available to a user on the DEC 10.

Five DATA 100 terminals provide remote batch service into the 370 for users at various locations including three plants and a research center. Most of the other plants have medium scale computer systems to support their local data processing needs. All make extensive use of process control mini-computers and two have UNIVAC 494 systems which can handle both real-time control and batch data processing.

Approximately 25 interactive CRTs scattered throughout various sales offices across the country have recently been installed to upgrade our Sales Order Entry System. Each terminal is connected to the DEC 10 on a dial-up 300 baud line.

PROPOSED SOLUTION

The most obvious solution to the problem of 370-DEC 10 communication would be to connect the DEC 10 to the 370 in a "back-to-back" fashion. To provide an upward flexibility, however, it is proposed that rather than connecting the machines in that way, they will be connected using a mini-computer as an interface. By designing the system which controls their interaction with a network approach, additional communication links may be obtained with a relatively small software investment. For example, if in the future, our research center obtains a large computer that they wish to incorporate into the communications process of the other two, an additional mini-computer would be placed there and connected via a communication line to the other.

This approach has several advantages. First, by going through a mini-computer, each of the two interfaces can be very carefully debugged in isolation and thus not affect the other machine. Second, once an IBM interface to the mini-computer is designed, one can connect any IBM machine into the network without rewriting any of the other interfaces. We would not have to write an IBM to UNIVAC interface, an IBM to CDC interface, an IBM to Honeywell interface, etc. Third, the only change necessary in the existing portion of the network, as the network expands, would be to inform the mini-computers of the presence of the other machines.

System description

In order to effectively describe a system as potentially complex as this one, we shall make use of techniques being developed under the classification of "Structured Programming."[17,37,48,55,56] The system will be broken down into various "levels of abstraction," each level being unaware of the existence of those above it, but being able to use the functions of lower levels to perform tasks and supply information. When a system is specified in terms

of levels, a clear idea of the operation of the system may be obtained by examining each level, starting from the top, and continuing down until further detail becomes unimportant for the purposes of the specification.

Let us now examine the first few levels of a portion of the proposed system. The top-most level is level 6, under that is level 5, and so on. We shall look at what occurs in the case of a user at a terminal on the DEC 10 submitting a program to a distant IBM 370 under HASP.

- Level 6

  On level 6 is found user and program processes. All interaction with the user or with a program written by the user occurs on this level. In fact, after this level is completely specified, the User Manual for the system can be written. In our example, an examination of what is happening would show the following steps:

  User creates the input file and a file for the output;

  User logs onto the network specifying his ID number;

  User types "SUBMIT" command specifying the input file, the output file, and the Host on which the program is to be run. This submit command calls on the HASP Submit-Receive function on level 5;

  User waits a brief period until he gets an "OK" from the terminal signifying that the program has been submitted. He is then free to either perform other actions or to sign off of the network;

  At some later time the user receives an "output ready" message on his terminal;

  User can now examine his output file.

- Level 5

  On level 5 is found the HASP Submit-Receive function, HSR, and functions to perform network access control, file access control, and remote program control. Let us examine the actions of the HSR function applied to our example:

  The HSR function obtains the name of the HASP-READER process of the specified Host. It then calls on a level 4 function to pass the input file to that process. When the level 4 function which controls process-to-process communication is completed, it will return a value corresponding to the job number that HASP has assigned;

  The HSR function sends an "OK" to the user. It then obtains the name of the HASP-WRITER process on the specified Host and calls on a level 4 to pass the job number and to specify the output file to the HASP-WRITER. Control returns when the output file is complete;

  The HSR function then sends an "OUTPUT READY" message to the user.

- Level 4

  On level 4 is found the functions which control the file descriptors, file access, and process-to-process communication. Examining the actions of the process-to-process communication function, PPC, applied to our example, we find:

  The PPC function converts the name of the process into a "well-known port" number and then establishes a logical link to the desired process;

  It then formulates a message containing the information to be passed and uses a level 3 function to transmit the message;

  It then receives a message in reply (which contains the job number in one case, and the output, in another). It passes this up to level 5 after destroying the links.

- Level 3

  Level 3 contains, among others, the function which transfers a message from one Host to another. To do this it:

  Takes the message, breaks it into pages, and calls a level 2 function to transmit each page;

  When the last page has been transmitted, it waits for an acknowledgment;

  If the acknowledgment indicates that a reply is being sent, it receives each page of the reply and passes up to level 4.

- Level 2

  On level 2 is handled the passing of pages. The steps are:

  The page is transferred from the Host to its IMP;

  The page is then translated into the standard network representation and broken into packets;

A level 1 function is called to transmit each packet.

- Level 1

At level 1 is handled the details of transmitting a packet from IMP to IMP. This includes retransmission in case of errors.

Stages of development

In order to allow the concept of a corporate computer network to be evaluated at minimum expense, it is desirable to break the development into discrete stages, each stage building on the hardware and software of the previous stage to add additional capability.

- Stage 1

This first stage would connect the DEC 10 to the local IBM 370/165 by using a single mini-computer. It would allow a user on the DEC 10 to conversationally build a program on a terminal and submit it to the 370 to be run under HASP. His output would be printed either at the 370, at the DEC 10, or at his terminal. This stage would also support the transfer of files consisting solely of character data to be transferred from one machine to the other.

The mini-computer hardware required for the stage would include: one CPU with 16-24K of memory, power monitor and restart, autoload, and teletype; two interfaces, one to the 370 and one to the DEC 10; a real time clock; and a cabinet. The approximate purchase price would be $25,000 to $35,000 with a monthly maintenance cost of approximately $300. In addition, a disk and controller should be rented for program development. This cost is approximately $500 per month and would be carried for the remaining stages.

- Stage 2

The second stage would remove the restriction on file transfer and allow files consisting of any type of data to be accessed from the other machine. At this stage, strict security controls would be integrated into the system.

The additional hardware required for this stage would include an additional CPU with 8K of memory and adaptors to interconnect the two CPUs. The approximate purchase cost would be $9,000-$12,000, with a monthly maintenance cost of approximately $75.

- Stage 3

This stage would expand the network to include computers at other locations. Additional hardware at the original site would include one synchronous communication controller for each outgoing line at a cost of $2,000-$2,500 with a maintenance cost of $25, and appropriate modems. Total cost for the original site, assuming two outgoing lines, would be between $36,000 and $49,500, excluding disk rental, modems, and communication lines.

- Stage 4

This stage could be developed in parallel with stage 3. It would add the capability for a user on a terminal attached to one machine to submit and interact with a program executing on the other machine. No additional hardware would be required.

- Stage 5

This stage consists of the design and implementation of automatic back-up procedures. Most of the preliminary analysis can be done in parallel with stages 2-4. These procedures would automatically create duplicate transactions of updates to critical files and have them routed to an alternate site to be applied to the back-up data base. No additional hardware is required.

HANDLING OF FILES IN A NETWORK

The handling of files in a non-homogeneous, distributed network poses several complex problems. These include control of access and transfer of information between dissimilar machines.

Control of access

That any system supporting multiple, simultaneous use of shared resources requires some sort of flexible, easy to use method of controlling access to those resources seems obvious to everyone (with the possible exception of the designers of IBM's OS/360), the main problem being how to provide the control at a reasonable cost. Restricting ourselves just to file access control, we see many potential methods with varying degrees of security and varying costs.[10,13,14,31,43] All provide control at the file level, some at the record level, and others at the element level. By designing our system with a Structured Programming approach, it should be possible to modify the method we choose, upgrading or downgrading the protection until a cost-benefit balance is reached.

Most designers of file access control systems have mentioned encryption of the data—we shall be no different. Apparently finding the cost prohibitive, they have failed to include this capability in their final product. In the proposed network, however, translation between the data representations of dissimiliar machines will be performed (see below), so the added cost of transforming from a "scrambled" to an "unscrambled" form will be small.

Each file access control system is based on a method which associates with each user-file pair a set of descriptors listing the rights or privileges granted to that user for that file (e.g., Read Access, Write Access, Transfer of Read Access to another user). Conceptualized as entries in a matrix, these descriptors are almost never stored as

such due to its sparceness. Rather, they are stored as lists, either attached to each element of a list of users or attached to each element of a list of files.

Assuming that we have a system for controlling file access, one design question for a distributed network is where to store the file access descriptors? For example, let us look at a network with three machines: A, B, and C, and a file, F, located at A but created by a user at B. To be accessible from the other machines, the file must be known by them and therefore, each machine must have a file descriptor stating that file F is located at A. If we also distribute the file access descriptors, an unauthorized user at C could gain access to the file by obtaining control of his machine and modifying the file access descriptors. Hence, each file access descriptor should be stored at the same location as the file it protects.

Transfer of information

The complexity of transferring information between two machines is increased by an order of magnitude when dissimilar machines are involved.[1,7,8] Using ASCII as the standard network code allows the interchange of files containing character data but does not address the problem of different representations of numerical data, e.g., packed decimal, short floating point, long floating point, etc.

Two alternatives present themselves: either allow each machine to translate from the representation of every other machine to its own or use a standard network representation and have each machine translate between its own and the network's. The first is attractive when only a few different types of machines will be allowed on the network (If there are N different types of machines, then N(N-1) translation routines might have to be written). The second alternative requires more effort in developing the standard network representation, but is really the only choice when the number of different types is larger than three or four.

Another problem is the large amount of translation that must take place. It may not be desirable to place this CPU laden task on a time-sharing machine for fear of degrading response time so the solution seems to lie in executing the translation within the IMPs. If performing translation interferes with the ability of the IMP to perform communication, an additional CPU can be attached to each in order to perform this task. With hardware costs decreasing 50 percent every two or three years, this seems an attractive solution.

INTERFACES

IMP—Host interface

The ARPA network is optimized toward supporting terminal interaction.[28] A commercial network must be optimized toward maximizing throughput of lengthy data files which produces large peak loads requiring high bandwidth channels between each Host and its IMP. In order to allow an IMP to communicate with its Host with a minimum of CPU intervention by either party, data must be transferred directly between the memory of the IMP and the memory of the Host. This can be achieved by connecting to an equivalent of the memory bus of the DEC 10 or multiplexor channel of the 370. With this type of interconnection, it will be necessary to configure the software so that each member of the communicating partnership appears to the other member as if it were a peripheral device of some sort, presumably a high-speed tape drive. Communication, therefore, would take place by one member issuing a READ while the other member simultaneously issues a WRITE.[24]

IMP-IMP interface

The IMPs will be linked by standard synchronous communication interfaces. Initial plans call for 40.8KB full duplex leased lines, but 19.2KB lines could also be used. A Cyclical Redundancy Check will provide detection of errors and cause the offending packet to be retransmitted.

Software interfaces

One of the main reasons for using mini-computers between the Hosts is to insure that the number of interface programs which must be written only grows linearly with the number of different types of Hosts. The effort in writing subsequent versions of the IMP-Host interface can be minimized by at least two methods:

1. Put as much of the system software as possible into the IMPs. Make use of sophisticated architecture[3] (e.g., multi-processor mini-computers, read-only memory) to obtain the power required.
2. For that portion of the system which resides in the Host, write the software using a standard, high-level language (e.g., FORTRAN) for as much of the code as possible.

REFERENCES

1. Anderson, Robert, et al, *Status Report on Proposed Data Reconfiguration Services*, ARPA Network Information Center Document No. 6715, April 28, 1971.
2. Aupperle, Eric, "MERIT Computer Network: Hardware Considerations" *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 49-63.
3. Bell, G., Cady, R., McFarland, H., Delagi, B., O'Laughlin, J., Noonan, R., "A New Architecture for Mini-Computers—The DEC PDP-11," *Proc. AFIPS 1970 SJCC*, Vol. 36, AFIPS Press, Montvale, N.J., pp. 657-675.
4. Bell, G., Habermann, A. N., McCredie, J., Rutledge, R., Wulf, W., "Computer Networks," *Computer*, IEEE Computer Group, September/October, 1970, pp. 13-23.
5. Bjorner, Dines, "Finite State Automation—Definition of Data Communication Line Control Procedures," *Proc. AFIPS 1970 FJCC*, Vol. 37, AFIPS Press, Montvale, N.J., pp. 477-491.
6. Bowdon, Edward, K., Sr., "Network Computer Modeling" *Proc. ACM Annual Conference*, 1972, pp. 254-264.

7. Bhushan, Abhay, *The File Transfer Protocol*, ARPA Network Information Center Document No. 10596, July 8, 1972.
8. Bhushan, Abhay, *Comments on the File Transfer Protocol*, ARPA Network Information Center Document No. 11357, August 18, 1972.
9. Carr, C. Stephen, Crocker, Stephen D., Cerf, Vinton G., "Host-Host Communication Protocol in the ARPA Network" *Proc. AFIPS 1970 SJCC*, Vol. 36, AFIPS Press, Montvale, N.J., pp. 589-597.
10. Carroll, John M., Martin, Robert, McHardy, Lorine, Moravec, Hans, "Multi-Dimensional Security Program for a Generalized Information Retrieval System," *Proc. AFIPS 1971 FJCC*, Vol. 39, AFIPS Press, Montvale, N.J., pp. 571-577.
11. Casey, R. G, "Allocation of Copies of a File in an Information Network," *Proc. AFIPS 1972 SJCC*, Vol. 40, AFIPS Press, Montvale, N.J., pp. 617-625.
12. Cocanower, Alfred, "MERIT Computer Network: Software Considerations," *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 65-77.
13. Conway, R. W., Maxwell, W. L., Morgan, H. L., "On the Implementation of Security Measures in Information Systems" *Comm. of the ACM*, Vol. 15, April, 1972, pp. 211-220.
14. Conway, Richard, Maxwell, William, Morgan, Howard, "Selective Security Capabilities in ASAP—A File Management System," *Proc. AFIPS 1972 SJCC*, Vol. 40, AFIPS Press, Montvale, N.J., pp. 1181-1185.
15. Crocker, Stephen D., Heafner, John F., Metcalfe, Robert M., Postel, Jonathan B., "Function-Oriented Protocols for the ARPA Computer Network," *Proc. AFIPS 1972 SJCC*, Vol. 40, AFIPS Press, Montvale, N.J., pp. 271-279.
16. deMercado, John, "Minimum Cost-Reliable Computer Communication Networks," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 553-559.
17. Dijkstra, E. W., "The Structure of the 'THE' Multiprogramming System," *Comm. of the ACM*, Vol. 11, May, 1968.
18. Farber, David, "Data Ring Oriented Computer Networks" *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 79-93.
19. Farben, David J., "Networks: An Introduction," *Datamation*, April, 1972, pp. 36-39.
20. Frank, Howard, Kahn, Robert E., Kleinrock, Leonard, "Computer Communication Network Design—Experience with Theory and Practice," *Proc. AFIPS 1972 SJCC*, Vol. 40, AFIPS Press, Montvale, N.J., pp. 255-270.
21. Frank, Howard, "Optimal Design of Computer Networks," *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 167-183.
22. Frank, H., Frisch, I.T., Chou, W., "Topological Considerations in the Design of the ARPA Computer Network," *Proc. AFIPS 1970 SJCC*, Vol. 36, AFIPS Press, Montvale, N.J., pp. 581-587.
23. Frank, Ronald A., "Commercial ARPA Concept Faces Many Roadblocks," *Computerworld*, November 1, 1972.
24. Fraser, A. G., "On the Interface Between Computers and Data Communications Systems," *Comm. of the ACM*, Vol. 15, July, 1972, pp. 566-573.
25. Grobstein, David L., Uhlig, Ronald P., "A Wholesale Retail Concept for Computer Network Management," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 889-898.
26. Hansen, Morris H., "Insuring Confidentiality of Individual Records in Data Storage and Retrieval for Statistical Purposes," *Proc. AFIPS 1971 FJCC*, Vol. 39, AFIPS Press, Montvale, N.J., pp. 579-585.
27. Hansler, E., McAuliffe, G. K., Wilkov, R. S., "Exact Calculation of Computer Network Reliability," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 49-54.
28. Heart, F. E., Kahn, R. E., Ornstein, S. M., Crowther, W. R., Walden, D. C., "The Interface Message Processor for the ARPA Computer Network," *Proc. AFIPS 1970 SJCC*, Vol. 37, AFIPS Press, Montvale, N.J., pp. 551-1567.
29. Hench, R. R., Foster, D. F., "Toward an Inclusive Information Network," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 1235-1241.
30. Herzog, Bert, "MERIT Computer Network," *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 45-48.
31. Hoffman, Lance J., "The Formulary Model for Flexible Privacy and Access Controls," *Proc. AFIPS 1971 FJCC*, Vol. 39, AFIPS Press, Montvale, N.J., pp. 587-601.
32. Hootman, Joseph T., "The Computer Network as a Marketplace," *Datamation*, April, 1972, pp. 43-46.
33. Kahn, Robert, "Terminal Access to the ARPA Computer Network" *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 972, pp. 147-166.
34. Kleinrock, Leonard, "Analytic and Simulation Methods in Computer Network Design," *Proc. AFIPS 1970 SJCC*, Vol. 36, AFIPS Press, Montvale, N.J., pp. 569-579.
35. Kleinrock, Leonard, "Survey of Analytical Methods in Queueing Networks," *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 185-205.
36. Lichtenberger, W., (Ed), *Tentative Specifications for a Network of Time-Shared Computers*, Document No. M-7, ARPA, September 9, 1966.
37. Liskov, B. H., "A Design Methodology for Reliable Software Systems," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 191-199.
38. Luther, W. J., "Conceptual Bases of CYBERNET," *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 111-146.
39. McKay, Douglas B., Karp, Donald P., "IBM Computer Network/440," *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 27-43.
40. McQuillan, J. M., Crowther, W. R., Cosell, B. P., Walden, D. C., Heart, F. E., "Improvements in the Design and Performance of the ARPA Network," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 741-754.
41. Mendicino, Samuel F., "OCTOPUS: The Lawrence Radiation Laboratory Network," *Computer Networks*, R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 95-110.
42. Metcalfe, Robert M., "Strategies for Operating Systems in Computer Networks," *Proc. ACM Annual Conference*, 1972, pp. 278-281.
43. Needham, R. M., "Protection Systems and Protection Implementations," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 571-578.
44. Ornstein, S. M., Heart, F. E., Crowther, W. R., Rising, H. K., Russell, S. B., Michel, A., "The Terminal IMP for the ARPA Computer Network," *Proc. AFIPS 1972 SJCC*, Vol. 40, AFIPS Press, Montvale, N.J., pp. 243-254.
45. Roberts, Lawrence G., "Extensions of Packet Communication Technology to a Hand Held Personal Terminal," *Proc. AFIPS 1972 SJCC*, Vol. 40, AFIPS Press, Montvale, N.J., pp. 295-298.
46. Roberts, Lawrence G., Wessler, Barry D., "Computer Network Development to Achieve Resource Sharing," *Proc. AFIPS 1970 SJCC*, Vol. 36, AFIPS Press, Montvale, N.J., pp. 543-549.
47. Rutledge, Ronald M., Vareha, Albin L., Varian, Lee C., Weis, Allan H., Seroussi, Salomon F., Meyer, James W., Jaffe, Joan F., Angell, Mary Anne K., "An Interactive Network of Time-Sharing Computers," *Proc. ACM Annual Conference*, 1969. pp. 431-441.
48. Sevcik, K. C., Atwood, J. W., Grushcow, M. S., Holt, R. C., Horning, J. J., Tsichritzis, D., "Project SUE as a Learning Experience," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 331-338.
49. Stefferud, Einar, "Management's Role in Networking," *Datamation*, April, 1972, pp. 40-42.
50. Thomas, Robert H., Henderson, D., Austin, "McROSS—A Multi-Computer Programming System," *Proc. AFIPS 1972 SJCC*, Vol. 40, AFIPS Press, Montvale, N.J., pp. 281-293.

51. Tobias, M. J., Booth, Grayce M., "The Future of Remote Information Processing Systems," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 1025-1035.
52. Walden, David C., "A System for Interprocess Communication in a Resource Sharing Computer Network," *Comm. of the ACM*, Vol. 15, April, 1972, pp. 221-230.
53. Weis, Allan H., "Distributed Network Activity at IBM," *Computer Networks*, R. Rustin (Ed.), Prentice Hall, Englewood Cliffs, N.J., 1972, pp. 1-25.
54. Williams, Leland H., "A Functioning Computer Network for Higher Education in North Carolina," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 899-904.
55. Wulf, William A., "Systems for Systems Implementors—Some Experience From BLISS," *Proc. AFIPS 1972 FJCC*, Vol. 41, AFIPS Press, Montvale, N.J., pp. 943-948.
56. Wulf, W. A., Russell, D. B., Habermann, A. N., "BLISS: A Language for Systems Programming," *Comm. of the ACM*, Vol. 14, December, 1971, pp. 780-790.

A system of APL functions to study computer networks

by T. D. FRIEDMAN

IBM Research Laboratory
San Jose, California

A collection of programs and procedural conventions will be described which were developed as part of a larger study of modeling and design of computer networks. This work was conducted under Navy Contract N0014-72-C-0299, and was based on approaches developed by Dr. R. G. Casey, to whom I am indebted for his helpful suggestions and encouragement. The programs are written on the APL terminal system. For proper understanding of the programming language used, it is desirable for the reader to refer to Reference 1.

These programs make it possible to create, modify and evaluate graph theoretic representations of computer networks in minutes while working at the terminal.

CONCEPTUAL FRAMEWORK

The overall conceptual framework for representing networks will first be discussed.

We assume a set of fixed nodes located at geographically dispersed locations, some of which contain copies of a given data file. Certain nodes are interconnected by transmission links. Together, the nodes and links constitute a particular network configuration. Each node is assigned a unique identification number, and the links are likewise identified by link numbers. An arbitrary configuration of an n-node identified by link numbers. An arbitrary configuration of an n-node network is represented by a *link list*, which consists of an m-by-3 array of the m link numbers, each of which is followed by the identification of the two nodes it connects. For example, a six-node network with all possible links provided would be represented by the following link list:

1	1	2
2	1	3
3	1	4
4	1	5
5	1	6
6	2	3
7	2	4
8	2	5
9	2	6
10	3	4
11	3	5
12	3	6
13	4	5
14	4	6
15	5	6

This network is depicted schematically in Figure 1. (Note that each link is represented in the figure by the least significant digit of its identification number.)

Any other configuration of a six-node network is necessarily a subset of the preceding network. One such subset is the following configuration, representing the network shown in Figure 2.

1	1	2
4	1	5
10	3	4
14	4	6
15	5	6

For certain operations, it is useful to represent a network by its *connection matrix*, in which the ith element of the jth row identifies the link connecting node i to j. If no link is present, or if $i=j$, then the element is zero.

Thus, the fully connected six-node network described above would be characterized by the connection matrix:

0	1	2	3	4	5
1	0	6	7	8	9
2	6	0	10	11	12
3	7	10	0	13	14
4	8	11	13	0	15
5	9	12	14	15	0

Likewise, the configuration represented in Figure 2 would be characterized by the connection matrix

0	1	0	0	4	0
1	0	0	0	0	0
0	0	0	10	0	0
0	0	10	0	0	14
4	0	0	0	0	15
0	0	0	14	15	0

BASIC NETWORK FUNCTIONS

In this section, functions for modeling and evaluating the networks are described.

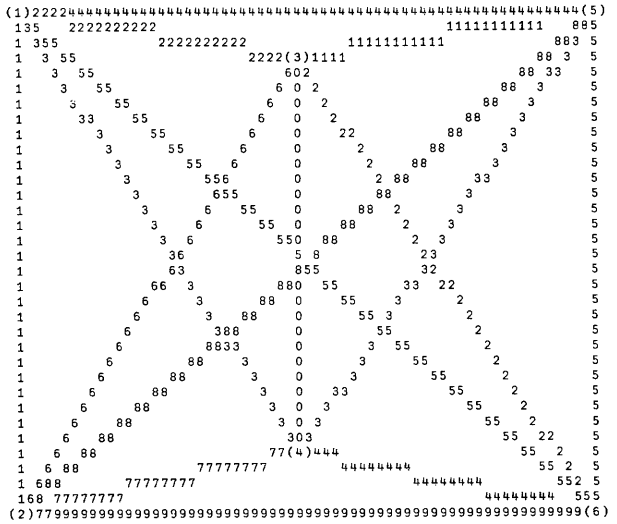


Figure 1

Network topology

An APL function, FORMCM, creates the connection matrix as output when given the link list and the node count.

The function OKCM determines whether a network configuration connects all nodes together. It is possible to determine whether all nodes are connected by calculating the inner product of a Boolean form of the connection matrix, repeated n-1 times for an n-node network. However, the OKCM function carries out this determination with somewhat less processing, by performing n-1 successive calculations of

$$CMS \leftarrow CMS \vee \vee/(CMS \vee . \wedge BCM)/BCM,$$

where BCM is a Boolean form of the connection matrix, CM, i.e., BCM[I;J]=1 if CM[I;J]≠0 or if I=J; BCM [I;J]=0 otherwise; and CMS is initially defined as BCM[1]. OKCM determines that all nodes are connected if and only if at the conclusion CMS consists only of 1's.

The function UNION finds the union of two link lists, and presents the link list of the result in ascending order. For example, if we define LINKS1 as

2	1	3
4	1	5
6	2	3
7	2	4
10	3	4
11	3	5
15	5	6

and LINKS2 as

1	1	2
4	1	5
10	3	4
14	4	6
15	5	6

then, LINKS1 UNION LINKS2 results in:

1	1	2
2	1	3
4	1	5
6	2	3
7	2	4
10	3	4
11	3	5
14	4	6
15	5	6

The CSP program searches for the shortest path spanning two nodes in a given network. It operates by first seeking a direct link between the nodes if one exists. If one does not, it follows each link in turn which emanates from the first node, and calls itself recursively to see if the second node can eventually be reached. A control parameter C is needed to limit the depth of recursion so as to avoid infinite regress.

SPAN is a function to call CSP without requiring the user to specify the recursion control parameter, C. SPAN operates by calling CSP with the control parameter set to the node count minus 1, since at most n-1 links are required for any path in an n-node network.

CSP and SPAN return only one set of links spanning the two given nodes, and this set consists of the fewest number of links possible. However, more than one such set of that number of links may be possible. The function MCSP operates exactly like CSP, except it finds all sets of the minimum number of links which span the two nodes. MSPAN corresponds to SPAN by allowing the user to omit the control terms, but calls MCSP rather than CSP.

Network traffic

We assume that the average amount of query and update activity emanating from each node to each file is known. This information is provided in a table called FILEACT, i.e., *file act*ivity.

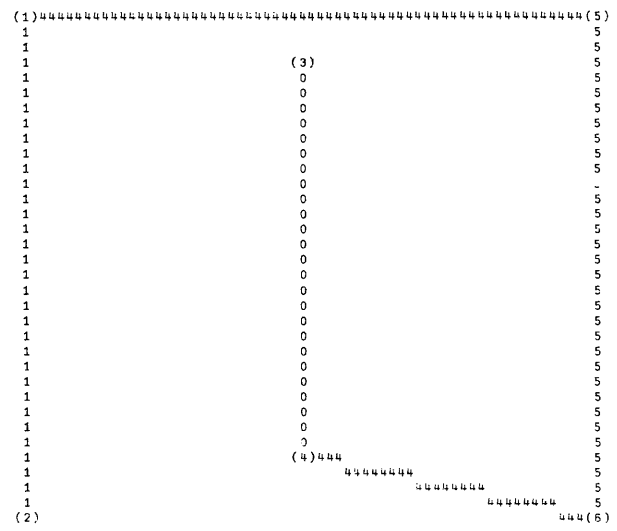


Figure 2

On the basis of these data, the activity on each link of the network can be calculated when the location of the files is specified. The function LUACT calculates the *l*ink *u*pdate *act*ivity, using CSP to find the sets of links between each node and each copy of the file. It then adds the appropriate update activity from the FILEACT table to the total transmission activity of each link.

The function LQACT derives the query activity on each link using a similar approach. However, a query from a node need only be transmitted to the *nearest* copy of the file. A function, PATH, finds the set of links connecting a node to the nearest copy of a file located at one or more nodes. LQACT uses PATH to select the links affected, then accumulates the total query activity on all links.

LACT finds the total activity on all links as the sum of LUACT and LQACT.

In the experiments run to date, five separate transmission capacities were available for each link, namely, 100, 200, 400, 1000 and 2000 kilobits per second. It is most economic to employ the lowest capacity line needed to handle the activity of each link. The function MINCAP accomplishes this goal by examining the activity of all links as determined by LUACT and LQACT, and it then calculates the minimal capacity needed in each link. If the activity of any link exceeds the maximal capacity possible, this is noted in an alarm.

Cost calculations

It is assumed that the costs are known for maintaining each possible link at each possible capacity in the network. This information is kept in TARTABLE, i.e., the *tar*riff *table*. In addition, the costs to maintain any file at any node are given in the FMAINT table.

By reference to the TARTABLE, the function FORMLTAR determines the monthly cost of all links, called LTAR. Using LTAR and the FMAINT table, the function TARRIF calculates the monthly cost of a configuration when given the nodes at which the files are located.

A function, GETTARRIF, derives this same cost data starting only with the link list and the locations of the files. In turn, it calls FORMCM to develop the connection matrix, then it calls LQACT and LUACT to determine activity on all links, then MINCAP to determine minimal link capacities, then FORMCTAR to derive LTAR, and finally it calls TARRIF to determine the total cost.

An abbreviated version of the latter function, called GETLTF, derives the link costs but does not find the total file maintenance cost.

NETWORK MODIFICATION FUNCTIONS

The cost of maintaining a network may on occasion be reduced by deleting links, by adding links, or by replacing certain links with others.

The function SNIP removes links from a network whenever this lowers cost or leaves the cost unchanged. SNIP must have the nodes specified at which files are located, and the link list. It proceeds by calculating the link activity, then it calls MINCAP to determine the capacity of each link. It deletes all links that carry no traffic. Then it attempts to delete each of the remaining links from the network, starting with the least active link. At each step, it calls OKCM to check that all nodes remain connected in the network—if not, that case is skipped. After it determines the cost effects of all possible deletions of single links, the one link (if any) is deleted which lowers cost most (or at least leaves it unchanged). Afterwards, it repeats the entire process on the altered network and terminates only when it finds that no additional links can be removed without increasing cost or disconnecting the network.

The function JOIN will add the one link, if any, to a specified node which most reduces cost. After one link is added, it repeats the process to determine if any others may also be added to that node. It follows a scheme similar to SNIP, except that a link will not be added if the cost is unchanged as a result.

JOINALL accomplishes the same process on a network-wide basis, i.e., a link will be added anywhere in the network if it reduces the cost.

REPLACE substitutes new links for old ones whenever this reduces cost. It begins by calling SNIP to delete all links whose removal will not raise costs, then it selectively attempts to delete each of the remaining links and attach a new link instead to one of the nodes that the old link had connected. After trying all such possible replacements and calculating the resulting costs, it chooses the one replacement, if any, which lowers the cost the most. Then, the whole process is repeated on the modified network, and terminates only when no further cost reduction is possible.

MESSAGE-COST-ORIENTED FUNCTIONS

The preceding functions assume an "ARPA-type" design[2,3] in which total costs are based on fixed rental rates for transmission lines for the links according to transmission capacities, plus the cost of maintaining files at specific nodes. However, networks may also be considered in which the costs are calculated according to message *traffic* across links. A family of functions have been written similar to the functions described above, but for which costs are calculated according to the moment-to-moment transmission activity. Those functions are described in this section.

A table, TC, specifies the transmission cost for sending messages over each link. It is assumed that the link capacities are fixed, and are given by the parameter LCAP. A function, FORMLCOST, uses TC and LCAP to create LCOST, the list of message transmission cost rates for the links.

Rather than simply finding a shortest set of links connecting two nodes, in this case it becomes necessary to compare the different total transmission costs for each possible set of links. The function ECSP is provided to do this. ECSP is similar to MCSP, but instead of returning all sets of the minimal number of links connecting two given nodes, it returns only the most economical single set of links.

14	64000		
145	64000		
146	71200		
15	72000		
24	73600		
245	73600		
1456	75200		
134	75200		
1345	75200		
135	79200		
4	80000		
45	80000		
246	80800		
25	81600		
124	81600		
1245	81600		
1346	82400		
156	83200		
2456	84800		
234	84800		
2345	84800		
136	86400		
13456	86400		
46	87200		
13	87200		
235	88800		
1246	88800		
125	89600		
1356	90400		
456	91200		
34	91200		
345	91200		
2346	92000		
256	92800		
12456	92800		
1234	92800		
12345	92800		
16	95200	26	104800
236	96000	123	104800
23456	96000	12356	108000
23	96800	36	110400
1235	96800	3	111200
346	98400	5	112000
2356	100000	1	112000
12346	100000	126	112800
1256	100800	356	114400
3456	102400	56	123200
35	103200	2	125600
1236	104000	12	129600
123456	104000	6	159200

Figure 3

Like MCSP, ECSP requires a recursion control parameter. ESP (which corresponds to MSPAN) calls ECSP, and provides the recursion control parameter implicitly as the node count minus 1.

LINKUACT and LINKQACT correspond to LUACT and LQACT but the links between nodes are chosen by ECSP rather than CSP so as to obtain lowest cost. LQACT calls ROUTE, which operates like PATH except that it chooses the most economic set of links connecting a node to one of several copies of a file. LINKACT corresponds to LACT in the same way.

The function COST corresponds to TARRIF, but it calculates the cost according to the total activity on each link times the component of LCOST corresponding to that link. The function TOTCOST calculates the sum of the costs over all links calculated by COST, and the function ALLCOSTS derives a table, COSTMATRIX, giving the total costs for each file when located at each possible node.

FICO creates a table of costs for all possible combinations of a file at different nodes, shown in Figure 3. The first column specifies the nodes at which the file is located and column 2 supplies the cost. The configuration used to derive this table was a fully connected six-node network.

FORMQM uses LINKQACT to create a table of query costs for each file when located at each node. FORMUM forms a similar table for update costs.

TRIM corresponds to SNIP, and deletes links from a network until the cost no longer drops.

SUPP (for supplement) corresponds to JOINALL, and adds links to a network until the cost no longer drops.

DIAGRAMMING FUNCTIONS

Functions are provided to diagram network configurations developed by the preceding functions. The diagram is prepared as a matrix of blank characters, with nodes indicated by parenthesized numbers, and links indicated by a line composed of a number or other character. Figures 1 and 2 are examples of such diagrams.

DEPICT creates such a matrix when given the dimensions and the positions in the matrix at which the nodes are to be located.

BLAZE draws a line between any two elements in the matrix, when given the location of the elements and the special character with which the line is to be drawn.

CONNECT draws a line, using BLAZE, between two nodes, looking up the locations of each node in the picture matrix to do so.

CONNECTALL diagrams an entire network configuration when given its link list as an argument. Each link is drawn using the least significant digit of the link number. Figures 1 and 2 were produced by CONNECTALL.

A function called SETUP is used to initialize the link list, connection matrix and node count parameter for various preassigned network configurations. The statement "SETUP 5" for example initializes the prespecified network configuration number five.

MEMO-PAD FUNCTIONS

Provisions were made for a memorandum-keeping system which may be of interest to other APL users, inasmuch as it is not restricted just to the network study.

The "niladic" function NOTE causes the terminal to accept a line of character input in which the user may write a comment or note for future references. The note is then marked with the current calendar date and filed on the top of a stack of notes (called NOTES). The function MEMO displays all previous notes with their dates of entry, and indexes each with an identification number. Any note may be deleted by the SCRATCH function. The user simply follows the word SCRATCH by the indices of notes to be deleted.

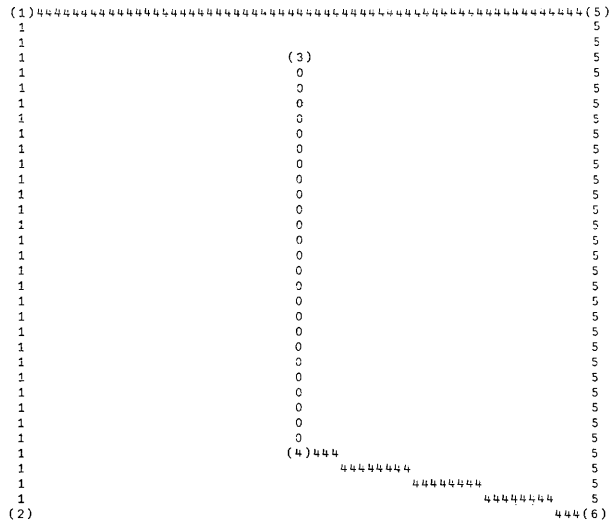


Figure 4a

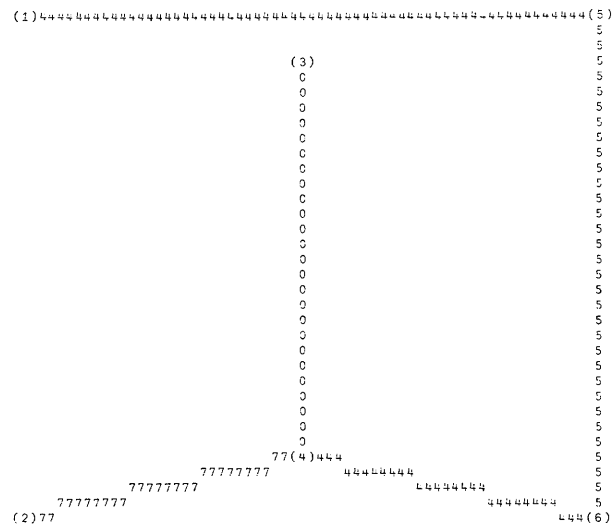


Figure 4b

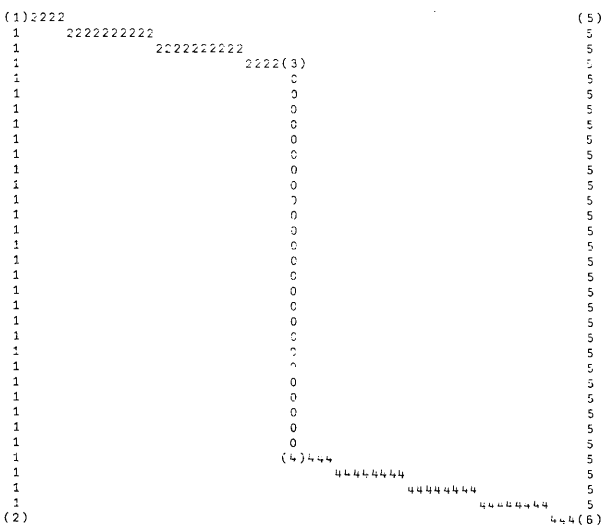


Figure 4c

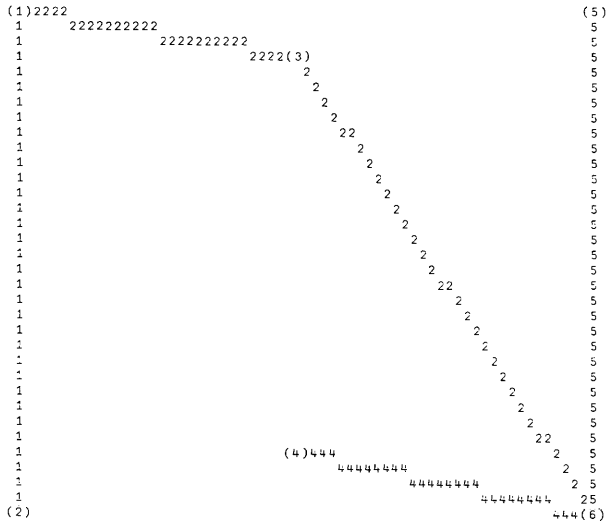


Figure 4d

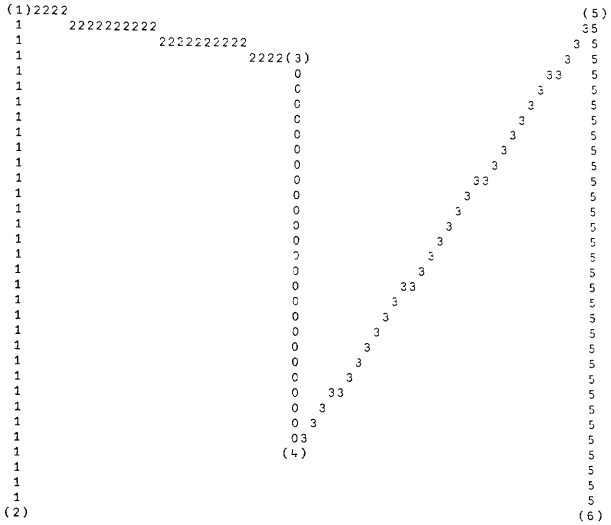


Figure 4e

These functions have proven useful for keeping records of current activities, ideas, and problems from one work session to the next.

EXAMPLES OF EXPERIMENTS WITH THE SYSTEM

In this section, three sample experiments will be discussed. The records of the terminal session are included in an Appendix to show how the experiments were conducted.

In the first run, we start with configuration five, by issuing the command

SETUP 5.

The system responds by typing "CASE 5 IS NOW SET UP." This establishes the link list and connection matrix of the configuration shown in Figure 4a.

We then issue the statement

0 REPLACE 3

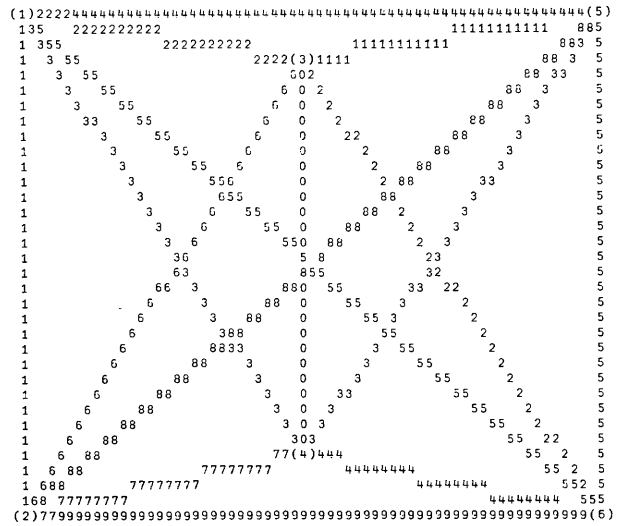


Figure 5a

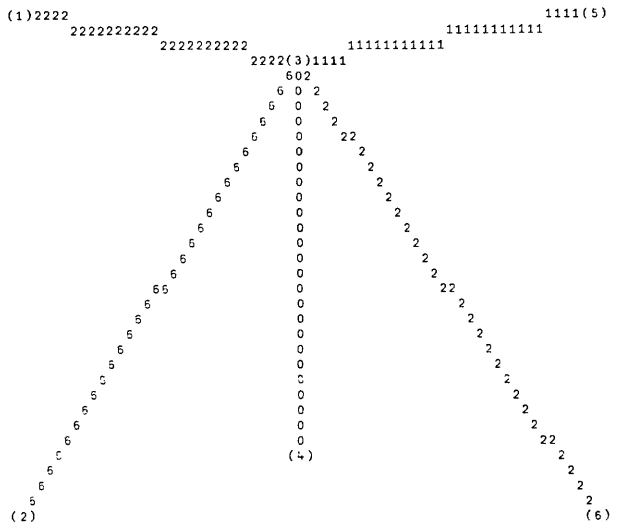


Figure 5b

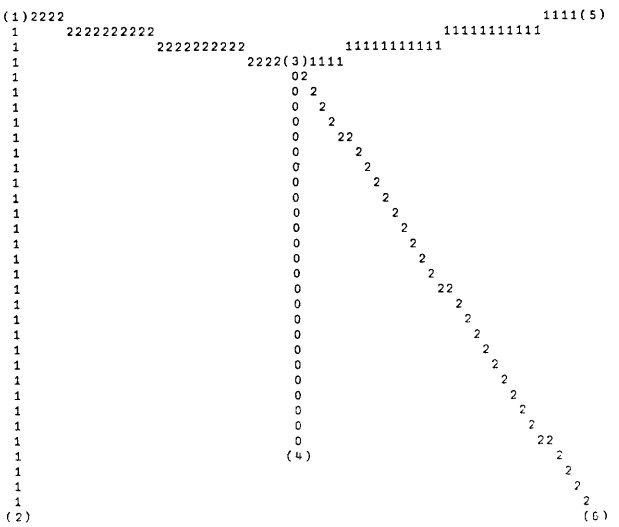


Figure 5c

which signifies that a file is located at the 3rd node, and that alternative links are to be tested in the network to find configurations which are less costly.

In the listing of the terminal session, a partial trace is made of REPLACE and of SNIP, called by REPLACE, to show the cost of each configuration as it is calculated Each time a configuration is discovered which has a lower cost than any of the preceding cases, its link list is displayed.

Thus, we start with the configuration shown in Figure 4a. SNIP finds it cannot delete any links. The cost of this configuration is 2700. REPLACE then attempts to change a link, but the resulting cost is 2750, and because it is more expensive than the original case, it is discarded. REPLACE then tries a second alteration, the cost of which is found to be 2550, which is less than the original cost. The link list of this configuration is displayed, and the revised network is shown in Figure 4b. The reader can note that link 1 has been replaced by link 7.

Following this, three more modifications are examined and the first two are found more expensive than the last acceptable change. The third, however, costs 1550 and thus is noted, and is shown in Figure 4c. Here we see that link 4 in the original configuration has been replaced by link 2.

A large number of additional modifications are examined, but only when it arrives at the configuration shown in Figure 4d is a more economic case found. Finally, REPLACE discovers that the configuration in Figure 4e is lower in cost than any others preceding it, costing 1300. Several additional cases are examined, following this, but none is lower than this last configuration, which is then displayed at the conclusion of execution.

Next, in a second run, we attempt to add links to the final configuration derived in the first run to see if a still cheaper case could be discovered. JOINALL was called with the file specified at node 3, but each new link it attempted to add failed to decrease the cost below 1300. Thus, no change was made.

In a third run, a fully connected network was specified, as shown in Figure 5a. REPLACE was called, again with the file at node 3. REPLACE called SNIP, which removed ten links to produce the configuration shown in Figure 5b, having a cost of 1500. REPLACE then modified the network to produce the structure in Figure 5c, having a cost of 1350. Thus, we see that sometimes an arbitrary configuration such as that in the first run may be a better starting point for REPLACE then a fully connected network, since the first run resulted in a more economic configuration than did the third.

REFERENCES

1. *APL/360-OS and APL/360-DOS, User's Manual*, IBM Corporation, White Plains, New York, Form Number SH20-0906.
2. Roberts, L. G., Wessler, B. D., "Computer Network Development to Achieve Resource Sharing," *Proceedings of AFIPS SJCC* 1970, pp. 543-549.
3. Frank, H., Frisch, I. T., and Chou, W., "Topological Considerations in the Design of the ARPA Computer Network," *ibid*, pp. 581.

First Run

APPENDIX

```
      SETUP 5
CASE 5 IS NOW SET UP

      0 REPLACE 3
SNIP[2]
   1   1   2
   4   1   5
  10   3   4
  14   4   6
  15   5   6
SNIP[4] 2700
REPLACE[3]
   1   1   2
   4   1   5
  10   3   4
  14   4   6
  15   5   6
REPLACE[18] 2750
REPLACE[18] 2550
REPLACE[22]
   4   1   5
   7   2   4
  10   3   4
  14   4   6
  15   5   6
REPLACE[18] 2750
REPLACE[18] 2850
REPLACE[18] 1550
REPLACE[22]
   1   1   2
   2   1   3
  10   3   4
  14   4   6
  15   5   6
REPLACE[18] 1800
REPLACE[18] 2800
REPLACE[18] 2700
REPLACE[18] 2400
REPLACE[18] 3250
REPLACE[18] 2400
REPLACE[18] 2400
REPLACE[18] 2600
REPLACE[18] 2950
REPLACE[18] 2100
REPLACE[18] 2300
REPLACE[18] 2000
REPLACE[18] 2000
REPLACE[18] 3150
REPLACE[18] 3250
REPLACE[18] 1700
REPLACE[18] 1750
REPLACE[18] 2050
REPLACE[18] 1700
REPLACE[18] 1800
REPLACE[18] 1900
REPLACE[18] 2000
REPLACE[18] 1500
REPLACE[18] 1550
REPLACE[18] 2000
REPLACE[18] 2100
REPLACE[18] 1650
REPLACE[18] 1550
REPLACE[18] 1500
REPLACE[18] 1450
REPLACE[18] 1700
REPLACE[18] 1800
REPLACE[18] 1500
REPLACE[18] 1850
REPLACE[18] 2550
REPLACE[18] 1650
REPLACE[18] 1450
REPLACE[18] 1550
REPLACE[18] 1550
REPLACE[18] 1650
REPLACE[18] 1550
REPLACE[18] 1650
REPLACE[18] 1350
REPLACE[18] 1400
REPLACE[18] 2150
REPLACE[18] 2250
REPLACE[18] 1500
REPLACE[18] 1400
   1   1   2
   2   1   3
  10   3   4
  11   3   5
  12   3   6
REPLACE[18] 2050
REPLACE[18] 1800
REPLACE[18] 2700
REPLACE[18] 2800
REPLACE[18] 1850
REPLACE[18] 1600
REPLACE[18] 1500
REPLACE[22]
   1   1   2
   2   1   3
  12   3   6
  14   4   6
  15   5   6
REPLACE[18] 2300
REPLACE[18] 2650
REPLACE[18] 1300
REPLACE[22]
   1   1   2
   2   1   3
  10   3   4
  13   4   5
  15   5   6
REPLACE[18] 2400
REPLACE[18] 2500
REPLACE[18] 1400
REPLACE[18] 1700
REPLACE[18] 1800
REPLACE[18] 1500
REPLACE[18] 1350
REPLACE[18] 1450
REPLACE[18] 1500
REPLACE[18] 1700
REPLACE[18] 1900
REPLACE[18] 1550
REPLACE[18] 2100
REPLACE[18] 2650
REPLACE[18] 1600
REPLACE[18] 1400
REPLACE[18] 1500
REPLACE[18] 2050
REPLACE[18] 2400
REPLACE[18] 1550
REPLACE[18] 2200
REPLACE[18] 2300
REPLACE[18] 1400
REPLACE[18] 2150
REPLACE[18] 2250
REPLACE[18] 1350
REPLACE[18] 1350
   1   1   2
   2   1   3
  10   3   4
  13   4   5
  15   5   6
```

APPENDIX

Second Run

```
      NL←0 JOINALL 3
JOINALL[3]
   1   1   2
   2   1   3
  10   3   4
  13   4   5
  15   5   6
JOINALL[5] 1300
JOINALL[10] 1300
JOINALL[10] 2200
JOINALL[10] 2150
JOINALL[18] 7
JOINALL[10] 1450
JOINALL[10] 1300
JOINALL[10] 1300
JOINALL[10] 2250
JOINALL[18] 7
JOINALL[10] 1400
JOINALL[10] 1350
JOINALL[18] 7
JOINALL[10] 1350
JOINALL[18] 7
JOINALL[18]

      NL
   1   1   2
   2   1   3
  10   3   4
  13   4   5
  15   5   6
```

Third Run

```
      0 REPLACE 3
SNIP[2]
   1   1   2
   2   1   3
   3   1   4
   4   1   5
   5   1   6
   6   2   3
   7   2   4
   8   2   5
   9   2   6
  10   3   4
  11   3   5
  12   3   6
  13   4   5
  14   4   6
  15   5   6
SNIP[4] 1500
REPLACE[3]
   2   1   3
   6   2   3
  10   3   4
  11   3   5
  12   3   6
REPLACE[18] 1650
REPLACE[18] 1650
REPLACE[18] 1900
REPLACE[18] 2000
REPLACE[18] 1350
REPLACE[22]
   1   1   2
   2   1   3
  10   3   4
  11   3   5
  12   3   6
REPLACE[18] 1450
REPLACE[18] 1700
REPLACE[18] 1800
REPLACE[18] 1800
REPLACE[18] 1900
```

A high-level language for use with multi-computer networks

by HARVEY Z. KRILOFF

University of Illinois
Chicago, Illinois

INTRODUCTION

Two basic trends can be observed in the modern evolution of computer systems. They are the development of computer systems dedicated to a single task or user (minicomputers) where the sophistication of large computer systems is being applied to smaller units, and the trend of very large systems that locate the user remotely from the computer and share resources between more and more locations. It is to the latter case that this paper is directed. This trend reaches its culmination in the design of distributed computer systems, where many individual computer components are located remotely from each other, and they are used to jointly perform computer operations in the solution of a single problem. Systems such as these are being developed in increasing numbers, although they are yet only a small fraction of the total number of computer systems. Examples of such systems range from those that are National and International in scope (the United States' APRANET,[1] Canada's CANUNET,[2] the Soviet Union's ASUS system[3] and the European Computer Network Project[4]), the statewide systems (the North Carolina Educational Computer System[5] and the MERIT Computer Network[6]), to single site systems (The Lawrence Radiation Laboratory Network[7] and Data Ring Oriented Computer Networks[8]). These systems and others have been designed to solve problems in the areas of research, education, governmental planning, airline reservations and commercial time-sharing. Taken together they demonstrate a capability for computer utilization that places more usable computer power in the user's control than he has ever had before. The challenge is to make effective use of this new tool.

Development of this new mode of computer usage has followed the same set of priorities that has prevented effective utilization of previous systems. A large body of information has been collected on the hardware technology of network systems,[9] but little effort has been expended on the development of software systems that allow the average user to make effective use of the network. A systematic examination of the requirements and a design for a language that uses the full facilities of a number of computer networks is needed.

NETWORK LANGUAGE REQUIREMENTS

After the language design had begun, it developed that the concept of what a computer network was, and how it was to be used, was not well understood. An effort was therefore begun to classify computer networks and their operations. This classification scheme indicated that a language for computer networks would have to be constantly changing because networks evolve from one form to another. It is this dynamic behavior that makes the design of a single language for different types of networks a high priority requirement.

Types of computer networks

A classification scheme was developed based on the resource availability within the computer network and the network's ability to recover from component failure. The scheme consists of six (6) different modes of network operation with decreasing dependability and cost for the later modes.

1. *Multiple Job Threads*: This consists of a system where all components are duplicated and calculations are compared at critical points in the job. If a component should fail, no time would be lost. Example: NASA Space Flight Monitoring
2. *Multiple Logic Threads*: This is identical with the previous mode except peripherals on the separate systems can be shared.
3. *Multiple Status Threads*: In this mode only one set of components performs each task. However, other systems maintain records of system status at various points in the job. If a component should fail all work since the last status check must be re-executed. Example: Remote Checkpoint-Restart
4. *Single Job Thread*: In this mode one computer system controls the sequencing of operations that may be performed on other systems. If a system failure occurs in the Master computer, the job is aborted.
5. *Load Sharing*: In this mode each job is performed using only a single computer system. Jobs are transferred to the system with the lightest load, if it has all

necessary resources. If the system fails you may lose all record of the job. Example: HASP to HASP job transfer.

6. *Star*: In this mode only one computer system is available to the user. It is the normal time-sharing mode with a single computer. If the system is down you can't select another computer.

All presently operating networks fall within one of the above modes of operation, and additional modes can be developed for future networks. Operation within the first two and last two modes require no additional language features that are not needed on a single (non-network) computer system. Thus, the language to be described will be directed toward utilizing the capabilities of networks operating in modes 3 and 4. In order to evaluate its usefulness, implementation will be performed on both the ARPANET and MERIT Networks. Both networks can operate in modes 3 and 4, and possess unique features that make implementation of a single network language a valid test of language transferability.

Computer network operations

In designing the language we recognized three (3) types of operations that the network user would require.

1. *Data Base Access*: Four operations are necessary for this type of usage. The user can copy the entire data base from one computer system to another, or he could access specific elements in the data base, or he could update the data base, or he could inquire about the status of the data base. Decisions on whether to copy the data base or access it element by element depend on data transfer speeds and the amount of data needed, therefore they must be determined for each job.
2. *Subtask Execution*: Tasks may be started in one of four different modes. In the Stimulus-Response mode a task is started in another machine while the Master computer waits for a task completion message. In the Status Check mode the subtask executes to completion while the Master task performs other work. The subtask will wait for a request from the Master to transmit the result. The third mode is an interactive version of status checking where the status check may reveal a request for more data. The final mode allows the subtask to interrupt the master task when execution is completed. Most networks do not have the facilities to execute in mode four, however all other modes are possible. The novice will find mode one easiest, but greater efficiency is possible with modes two and three.
3. *Network Configuration*: This type of usage is for the experienced network user. It provides access to the network at a lower level, so that special features of a specific network may be used. Programs written using these type commands may not be transferable to other networks. These type commands allow direct connection to a computer system, submission of a RJE job, reservation of a system resource, network status checking, select network options, access to system documentation, and establishment of default options.

The network language must allow the user to perform all these operations by the easiest method. Since the average user will know very little about the network, the system must be supplied with default parameters that will make decisions that the user does not direct. These defaults can be fitted to the network configuration, the individual user or a class of problems.

User operations must be expressible in a compressed form. Operations that the user performs very often should be expressed by a single command. This will prevent programming errors and it will allow for optimization of command protocol. As new operations are required they should be able to be added to the language without affecting already defined commands.

Additional language constraints

Three of the six constraints used to design the network language have already been described. The list of features that were considered necessary for a usable language are:

1. All purely systems requirements should be invisible to the user.
2. The language should be easily modified to adapt to changes in the network configuration.
3. The language should provide easy access to all features of the network at a number of degrees of sophistication.
4. The language should provide a method for obtaining on-line documentation about the available resources.
5. The fact that the system is very flexible should not greatly increase the system's overhead.
6. The language syntax is easy to use, is available for use in a non-network configuration, and it will not require extensive modification to transfer the language to another network.

These requirements are primarily dictated by user needs, rather than those required to operate the hardware/software system. It is the hope that the end result would be a language that the user would use without knowing that he was using a network.

The demand for on-line documentation is particularly important. Most software systems are most effectively used at the location where they were developed. As you get farther from this location, fewer of the special features and options are used because of lack of access to documentation. Since most of the systems to be used will reside on remote computers, it is important that the user

be able to obtain current documentation while he uses the system. That documentation should reside on the same computer as the code used to execute the system.

Options used to determine which computer system you are communicating with should not have to exist in interpretable code. This generality often leads to heavy overhead that defeats the advantages of a computer network. An example of this was the Data Reconfiguration Service created by Rand[10] as a general data converter that could be used when transferring data from one computer to another on the ARPANET. Because it was written to execute interpretly, data conversion can only be performed at low speed. While high-speed and low-overhead were not conditions of their implementation, an operational language should not produce such restricted conditions of usage.

The final condition that the language be easily used and operate on a single computer, led to the investigation of available extendable languages. Using an already developed language has certain advantages, people will not need to learn a new language to use the network, program development can continue even when the network is not operating, and network transparency is heavily dictated by the already designed language syntax. It was a belief that a network language should not be different in structure than any other language that led to the investigation of a high-level language implementation.

THE NETWORK LANGUAGE

An available language was found that met most of the requirements of our network language. The language is called SPEAKEASY[11] and it consists of a statement interpreter and a series of attachable libraries. One set of these libraries consist of linkable modules called "linkules" that are blocks of executable code that can be read off the disk into core and executed under control of the interpreter. This code, that may be written in a high-level language such as FORTRAN, can be used to perform the necessary protocols and other system operations required by the network. Thus, the user would not be required to know anything other than the word that activates the operation of the code. Each user required operation could be given a different word, where additional data could be provided as arguments of the activating word.

Since there are almost no limitations on the number of library members, and each user could be provided with his own attachable library, the language can be easily extended to accommodate new features created by the network. Linkules in SPEAKEASY could be created that communicate with individual computer systems or that perform similar operations in more than one system. Where the latter is implemented, an automatic result would be the creation of a super control language. Since one of the factors that prevent the average user from using more than one computer system is the non-uniformity of the operating systems, the development of a network language will eliminate this problem. Using data stored in other libraries, the network language could supply the needed control syntax to execute a specified task. This operation is not very much different from what the user does when he is supplied control cards by a consultant that he uses until it is outdated by systems changes.

The SPEAKEASY language presently provides the facility to provide on-line documentation about itself based on data in attachable libraries. This would be extended to allow the SPEAKEASY interpreter to read from libraries resident on a remote computer. Therefore, the documentation could be kept up-to-date by the same people responsible for developing the executing code.

Since SPEAKEASY linkules are compiled code, and there may exist seperate modules that are only loaded into core when needed, minimal overhead is provided by adding new operations to the system. This is the same technique used to add operations to the present system, therefore no difference should be detectable between resident and remote linkules.

NETWORK MODULE PROTOCOLS

Where a single computer version of SPEAKEASY has an easy task to determine how a command should be executed, a multi-computer version makes more complex decisions. Therefore it is necessary that there be established a well defined pattern of operations to be performed by each linkule.

Linkule order of operations

Each linkule that establishes communication with a remote (slave) computer system should execute each of the following ten operations, so as to maintain synchronization between the Master task and all remote subtasks. No module will be allowed to leave core until the tenth step is performed.

1. Determine System Resources Needed.
2. Establish the Network Connections.
   a) Select the Computer Systems that will be used.
   b) Establish the System Availability.
   c) Perform the Necessary Connect & Logon Procedures.
3. Allocate the Needed Computer System Resources to the Job.
4. Provide for System Recovery Procedures in the case of System Failure.
5. Determine what Data Translation Features are to be used.
6. Determine whether Data Bases should be moved.
7. Start the main task in the remote computer executing.
8. Initiate and Synchronize any subtasks.

9. Terminate any subtasks.
10. Terminate the remote task and close all related system connections.

In the present single computer version of SPEAKEASY only tasks 3 and 7 are executed, and a more limited form of task 4 is also performed. Where the overhead in opening and closing network connections is greater then the cost of leaving the connections open, the 10th task will not terminate any connection once made, allowing the next linkule to check and find it open.

Information on the systems availability can be obtained by inquiry from the network communications software. Procedures and resource information can be obtained from data stored in local or remote data set, by inquiry from the user, or by inquiry from the system. All allocation procedures will be performed by submitting the appropriate control commands to the standard operating system on the relevant computer.

Since the SPEAKEASY system will have the necessary data about the origination and destination computers for any unit of data being transfered, it can link to a linkule that is designed for that translation only. This linkule could take advantage of any special features or tricks that would speed the translation process, thus reducing overhead during the translation process.

Requirements for network speakeasy

The minimal requirements to execute the Network version of SPEAKEASY is a computer that will support the interpreter. At the present time that means either an IBM 360 or 370 computer operating under either the O/S or MTS operating systems, or a FACOM 60 computer system. The network protocols for only two operations are required. The Master computer must be able to establish a connection to a remote computer and it must be able to initiate execution of a job in the remote computer system.

The remote site should provide either a systems routine or a user created module that will perform the operations requested by the Master computer. This program, called the Network Response Program, is activated whenever a request is made to the remote computer. There may be one very general Network Response Program, or many different ones designed for specific requests. Figure 1 shows the modular structure of the Network Language in both the Master and Slave (Subtask) computing systems.

Because the network features of the language are required to be totally transparent to the user, no examples of network programming are shown. The reader should refer to previously published papers on Speakeasy for examples of programming syntax.

SUMMARY

A network version of the SPEAKEASY system is described that consists of a series of dynamically linked modules that perform the necessary language and system protocols. These protocols are transparent to the user, producing a language that has additional power without additional complexity. The language uses attached libraries to provide the necessary information that will tailor the system to a specific network, and to supply on-line documentation. Since the modules are compiled code, the generality of the system does not produce a large overhead.

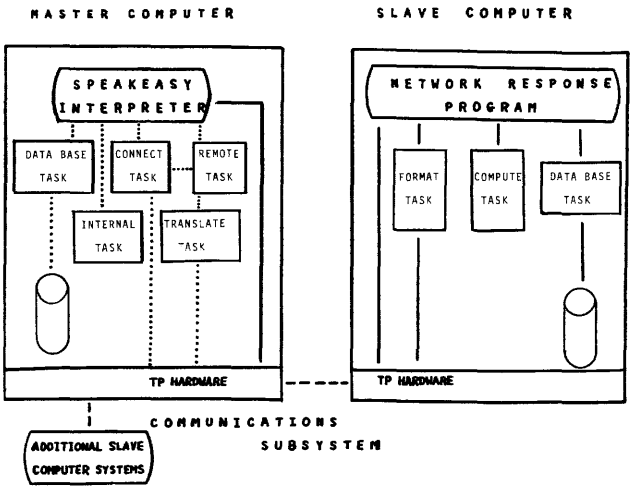


Figure 1—The modular structure of the Network Speakeasy System. Dotted lines indicate dynamic linkages, dashed lines are telecommunications links, and solid lines are standard linkages.

ACKNOWLEDGMENT

The author is indebted to Stanley Cohen, developer of the SPEAKEASY system. His assistance and support made this extension possible.

REFERENCES

1. Roberts, L. G., "Resource Sharing Networks," *Proceedings of IEEE International Conference*, 1969.
2. Demercado, J., Guindon, R., Dasilva, J., Kadoch, M., "The Canadian Universities Computer Network Topological Considerations," *The First International Conference on Computer Communication*, 1972.
3. Titus, J., "Soviet Computing—A Giant Awake," *Datamation*, Vol. 17, No. 24, 1971.
4. Barber, D. L. A., "The European Computer Network Project," *The First International Conference on Computer Communication*, 1972.
5. Denk, J. R., "Curriculm Development of Computer Usage in North Carolina," *Proceedings of a Conference on Computers in the Undergraduate Curricula.*
6. Herzog, B., "Computer Networks," *International Computing Symposium*, ACM, 1972.
7. Mendicino, S. F., "OCTOPUS—The Lawrence Radiation Laboratory Network," *Computer Networks*, Prentice-Hall, Englewood Cliffs, N.J., 1972.

8. Farber, D., "Data Ring Oriented Computer Networks," *Computer Networks*, Prentice-Hall, Englewood Cliffs, N.J., 1972.
9. Frank, H., Kahn, R. E., Kleinrock, L., "Computer Communication Network Design—Experience with Theory and Practice," *Networks*, Vol. 2, 1972.
10. Harslem, E. F., Heafner, J., Wisniewski, T. D., *Data Reconfiguration Service Compiler—Communication Among Heterogeneous Computer Center Using Remote Resource Sharing*, Rand Research Report R-887-ARPA, 1972.
11. Cohen, S., "Speakeasy—RAKUGO," *First USA Japan Computer Conference Proceedings*, 1972.

A resource sharing executive for the ARPANET*

by ROBERT H. THOMAS

Bolt, Beranek and Newman, Inc.
Cambridge, Mass.

INTRODUCTION

The Resource Sharing Executive (RSEXEC) is a distributed, executive-like system that runs on TENEX Host computers in the ARPA computer network. The RSEXEC creates an environment which facilitates the sharing of resources among Hosts on the ARPANET. The large Hosts, by making a small amount of their resources available to small Hosts, can help the smaller Hosts provide services which would otherwise exceed their limited capacity. By sharing resources among themselves the large Hosts can provide a level of service better than any one of them could provide individually. Within the environment provided by the RSEXEC a user need not concern himself directly with network details such as communication protocols nor even be aware that he is dealing with a network.

A few facts about the ARPANET and the TENEX operating system should provide sufficient background for the remainder of this paper. Readers interested in learning more about the network or TENEX are referred to the literature; for the ARPANET References 1,2,3,4; for TENEX. References 5,6,7.

The ARPANET is a nationwide heterogeneous collection of Host computers at geographically separated locations. The Hosts differ from one another in manufacture, size, speed, word length and operating system. Communication between the Host computers is provided by a subnetwork of small, general purpose computers called Interface Message Processors or IMPs which are interconnected by 50 kilobit common carrier lines. The IMPs are programmed to implement a store and forward communication network. As of January 1973 there were 45 Hosts on the ARPANET and 33 IMPs in the subnet.

In terms of numbers, the two most common Hosts in the ARPANET are Terminal IMPs called TIPs[12] and TENEXs.[9] TIPs[8,9] are mini-Hosts designed to provide inexpensive terminal access to other network Hosts. The TIP is implemented as a hardware and software augmentation of the IMP.

TENEX is a time-shared operating system developed by BBN to run on a DEC PDP-10 processor augmented with paging hardware. In comparison to the TIPs, the TENEX Hosts are large. TENEX implements a virtual processor with a large (256K word), paged virtual memory for each user process. In addition, it provides a multi-process job structure with software program interrupt capabilities, an interactive and carefully engineered command language (implemented by the TENEX EXEC) and advanced file handling capabilities.

Development of the RSEXEC was motivated initially by the desire to pool the computing and storage resources of the individual TENEX Hosts on the ARPANET. We observed that the TENEX virtual machine was becoming a popular network resource. Further, we observed that for many users, in particular those whose access to the network is through TIPs or other non-TENEX Hosts, it shouldn't really matter which Host provides the TENEX virtual machine as long as the user is able to do his computing in the manner he has become accustomed*. A number of advantages result from such resource sharing. The user would see TENEX as a much more accessible and reliable resource. Because he would no longer be dependent upon a single Host for his computing he would be able to access a TENEX virtual machine even when one or more of the TENEX Hosts were down. Of course, for him to be able to do so in a useful way, the TENEX file system would have to span across Host boundaries. The individual TENEX Hosts would see advantages also. At present, due to local storage limitations, some sites do not provide all of the TENEX subsystems to their users. For example, one site doesn't support FORTRAN for this reason. Because the subsystems available would, in effect, be the "union" of the subsystems available on all TENEX Hosts, such Hosts would be able to provide access to all TENEX subsystems.

The RSEXEC was conceived of as an experiment to investigate the feasibility of the multi-Host TENEX concept. Our experimentation with an initial version of the RSEXEC was encouraging and, as a result, we planned to develop and maintain the RSEXEC as a TENEX subsystem. The RSEXEC is, by design, an evo-

* This work was supported by the Advanced Projects Research Agency of the Department of Defense under Contract No. DAHC15-71-C-0088.

* This, of course, ignores the problem of differences in the accounting and billing practices of the various TENEX Hosts. Because all of the TENEX Hosts (with the exception of the two at BBN) belong to ARPA we felt that the administrative problems could be overcome if the technical problems preventing resource sharing were solved.

lutionary system; we planned first to implement a system with limited capabilities and then to let it evolve, expanding its capabilities, as we gained experience and came to understand the problems involved.

During the early design and implementation stages it became clear that certain of the capabilities planned for the RSEXEC would be useful to all network users, as well as users of a multi-Host TENEX. The ability of a user to inquire where in the network another user is and then to "link" his own terminal to that of the other user in order to engage in an on-line dialogue is an example of such a capability.

A large class of users with a particular need for such capabilities are those whose access to the network is through mini-Hosts such as the TIP. At present TIP users account for a significant amount of network traffic, approximately 35 percent on an average day.[10] A frequent source of complaints by TIP users is the absence of a sophisticated command language interpreter for TIPs and, as a result, their inability to obtain information about network status, the status of various Hosts, the whereabouts of other users, etc., without first logging into some Host. Furthermore, even after they log into a Host, the information readily available is generally limited to the Host they log into. A command language interpreter of the type desired would require more (core memory) resources than are available in a TIP alone. We felt that with a little help from one or more of the larger Hosts it would be feasible to provide TIP users with a good command language interpreter. (The TIPs were already using the storage resources of one TENEX Host to provide their users with a network news service.[10,11] Further, since a subset of the features already planned for the RSEXEC matched the needs of the TIP users, it was clear that with little additional effort the RSEXEC system could provide TIP users with the command language interpreter they needed. The service TIP users can obtain through the RSEXEC by the use of a small portion of the resources of several network Hosts is superior to that they could obtain either from the TIP itself or from any single Host.

An initial release of the RSEXEC as a TENEX subsystem has been distributed to the ARPANET TENEX Hosts. In addition, the RSEXEC is available to TIP users (as well as other network users) for use as a network command language interpreter, preparatory to logging into a particular Host (of course, if the user chooses to log into TENEX he may continue using the RSEXEC after login). Several non-TENEX Hosts have expressed interest in the RSEXEC system, particularly in the capabilities it supports for inter-Host user-user interaction, and these Hosts are now participating in the RSEXEC experiment.

The current interest in computer networks and their potential for resource sharing suggests that other systems similar to the RSEXEC will be developed. At present there is relatively little in the literature describing such distributing computing systems. This paper is presented to record our experience with one such system; we hope it will be useful to others considering the implementation of such systems.

The remainder of this paper describes the RSEXEC system in more detail: first, in terms of what the RSEXEC user sees, and then, in terms of the implementation.

THE USER'S VIEW OF THE RSEXEC

The RSEXEC enlarges the range of storage and computing resources accessible to a user to include those beyond the boundaries of his local system. It does that by making resources, local and remote, available as part of a single, uniformly accessible pool. The RSEXEC system includes a command language interpreter which extends the effect of user commands to include all TENEX Hosts in the ARPANET (and for certain commands some non-TENEX Hosts), and a monitor call interpreter which, in a similar way, extends the effect of program initiated "system" calls.

To a large degree the RSEXEC relieves the user and his programs of the need to deal directly with (or even be aware that they are dealing with) the ARPANET or remote Hosts. By acting as an intermediary between its user and non-local Hosts the RSEXEC removes the logical distinction between resources that are local and those that are remote. In many contexts references to files and devices* may be made in a site independent manner. For example, although his files may be distributed among several Hosts in the network, a user need not specify where a particular file is stored in order to delete it; rather, he need only supply the file's name to the delete command.

To a first approximation, the user interacts with the RSEXEC in much the same way as he would normally interact with the standard (single Host) TENEX executive program. The RSEXEC command language is syntactically similar to that of the EXEC. The significant difference, of course, is a semantic one; the effect of commands are no longer limited to just a single Host.

Some RSEXEC commands make direct reference to the multi-Host environment. The facilities for inter-Host user-user interaction are representative of these commands. For example, the WHERE and LINK commands can be used to initiate an on-line dialogue with another user:

←WHERE (IS USER) JONES**
JOB 17 TTY6 USC
JOB 5 TTY14 CASE
←LINK (TO TTY) 14 (AT SITE) CASE

* Within TENEX, peripheral devices are accessible to users via the file system; the terms "file" and "device" are frequently used interchangeably in the following.

** "←" is the RSEXEC "ready" character. The words enclosed in parentheses are "noise" words which serve to make the commands more understandable to the user and may be omitted. A novice user can use the character ESC to cause the RSEXEC to prompt him by printing the noise words.

Facilities such as these play an important role in removing the distinction between "local" and "remote" by allowing users of geographically separated Hosts to interact with one another as if they were members of a single user community. The RSEXEC commands directly available to TIP users in a "pre-login state" include those for inter-Host user-user interaction together with ones that provide Host and network status information and network news.

Certain RSEXEC commands are used to define the "configuration" of the multi-Host environment seen by the user. These "meta" commands enable the user to specify the "scope" of his subsequent commands. For example, one such command (described in more detail below) allows him to enlarge or reduce the range of Hosts encompassed by file system commands that follow. Another "meta" command enables him to specify a set of peripheral devices which he may reference in a site independent manner in subsequent commands.

The usefulness of multi-Host systems such as the RSEXEC is, to a large extent, determined by the ease with which a user can manipulate his files. Because the Host used one day may be different from the one used the next, it is *necessary* that a user be able to reference any given file from all Hosts. Furthermore, it is *desirable* that he be able to reference the file in the *same* manner from all Hosts.

The file handling facilities of the RSEXEC were designated to:

1. Make it *possible* to reference any file on any Host by implementing a file name space which spans across Host boundaries.
2. Make it *convenient* to reference frequently used files by supporting "short hand" file naming conventions, such as the ability to specify certain files without site qualification.

The file system capabilities of the RSEXEC are designed to be available to the user at the command language level and to his programs at the monitor call level. An important design criterion was that existing programs be able to run under the RSEXEC without reprogramming.

File access within the RSEXEC system can be best described in terms of the commonly used model which views the files accessible from within a Host as being located at terminal nodes of a tree. Any file can be specified by a *pathname* which describes a path through the tree to the file. The *complete* pathname for a file includes every branch on the path leading from the root node to the file. While, in general, it is necessary to specify a complete pathname to uniquely identify a file, in many situations it is possible to establish contexts within which a *partial* pathname is sufficient to uniquely identify a file. Most operating systems provide such contexts, designed to allow use of partial pathnames for frequently referenced file, for their users.*

It is straightforward to extend the tree structured model for file access within a single Host to file access within the entire network. A new root node is created with branches to each of the root nodes of the access trees for the individual Hosts, and the complete pathname is enlarged to include the Host name. A file access tree for a single Host is shown in Figure 1; Figure 2 shows the file access tree for the network as a collection of single Host trees.

The RSEXEC supports use of complete pathnames that include a Host component thereby making it possible (albeit somewhat tedious) for users to reference a file on any Host. For example, the effect of the command

←APPEND (FILE) [CASE]DSK:<THOMAS>DATA.
NEW (TO FILE) [BBN]DSK:<BOBT>DATA.OLD**

is to modify the file designated ① in Figure 2 by appending to it the file designated ②.

To make it convenient to reference files, the RSEXEC allows a user to establish contexts for partial pathname interpretation. Since these contexts may span across several Hosts, the user has the ability to configure his own "virtual" TENEX which may in reality be realized by the resources of several TENEXs. Two mechanisms are available to do this.

The first of these mechanisms is the *user profile* which is a collection of user specific information and parameters

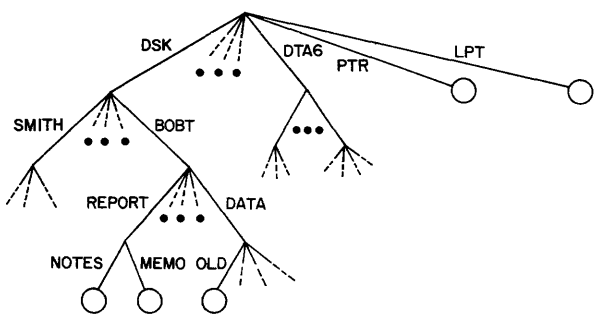


Figure 1—File access tree for a single Host. The circles at the terminal nodes of the tree represent files

* For example, TENEX does it by:
1. Assuming default values for certain components left unspecified in partial pathnames;
2. Providing a reference point for the user within the tree (working directory) and thereafter interpreting partial pathnames as being relative to that point. TENEX sets the reference point for each user at login time and, subject to access control restrictions, allows the user to change it (by "connecting" to another directory).

** The syntax for (single Host) TENEX pathnames includes device, directory, name and extension components. The RSEXEC extends that syntax to include a Host component. The pathname for ② specifies: the CASE Host; the disk ("DSK") device; the directory THOMAS; the name DATA; and the extension NEW.

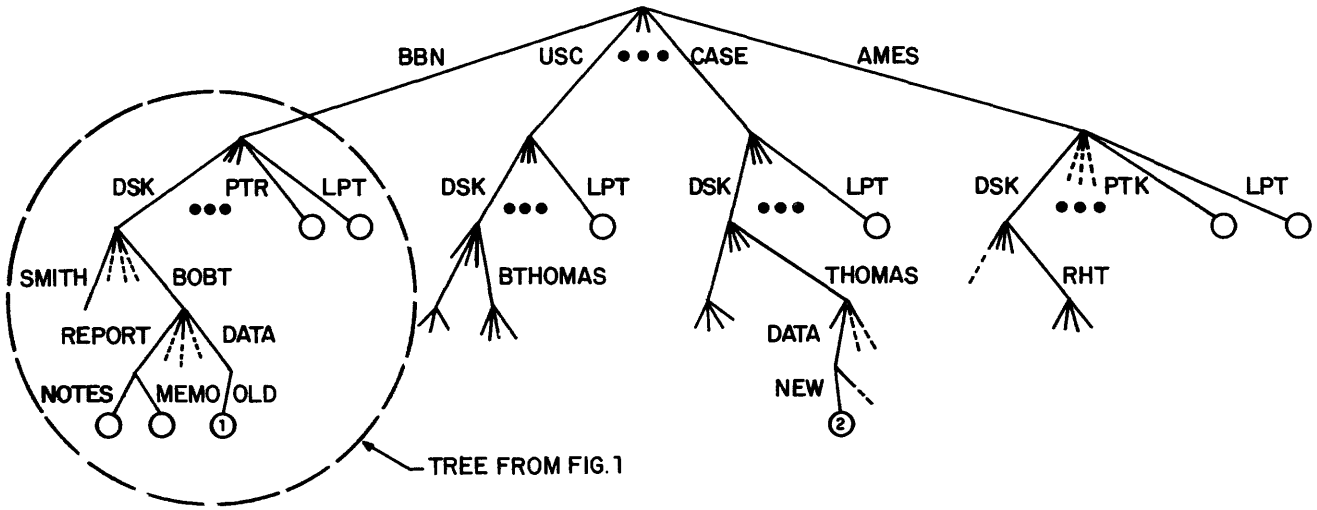


Figure 2—File access tree for a network. The single Host access tree from Figure 1 is part of this tree

maintained by the RSEXEC for each user. Among other things, a user's profile specifies a group of file directories which taken together define a *composite* directory for the user. The "contents" of the composite directory are the union of the "contents" of the file directories specified in the profile. When a pathname without site and directory qualification is used, it is interpreted relative to the user's composite directory. The composite directory serves to define a reference point within the file access tree that is used by the RSEXEC to interpret partial pathnames. That reference point is somewhat unusual in that it spans several Hosts.

One of the ways a user can reconfigure his "virtual" TENEX is by editing his profile. With one of the "meta" commands noted earlier he can add or remove components of his composite directory to control how partial pathnames are interpreted.

An example may help clarify the role of the user profile, the composite directory and profile editing. Assume that the profile for user Thomas contains directories BOBT at BBN, THOMAS at CASE and BTHOMAS at USC (see Figure 2). His composite directory, the reference point for pathname interpretation, spans three Hosts. The command

←APPEND (FILE) DATA.NEW (TO FILE) DATA.OLD

achieves the same effect as the APPEND command in a previous example. To respond the RSEXEC first consults the composite directory to discover the locations of the files, and then acts to append the first file to the second; how it does so is discussed in the next section. If he wanted to change the scope of partial pathnames he uses, user Thomas could delete directory BOBT at BBN from his profile and add directory RHT at AMES to it.

The other mechanism for controlling the interpretation of partial pathnames is *device binding*. A user can instruct the RSEXEC to interpret subsequent use of a particular device name as referring to a device at the Host he specifies. After a device name has been *bound* to a Host in this manner, a partial pathname without site qualification that includes it is interpreted as meaning the named device at the specified Host. Information in the user profile specifies a set of default device bindings for the user. The binding of devices can be changed dynamically during an RSEXEC session. In the context of the previous example the sequence of commands:

←BIND (DEVICE) LPT (TO SITE) BBN
←LIST DATA.NEW
←BIND (DEVICE) LPT (TO SITE) USC
←LIST DATA.NEW

produces two listings of the file DATA.NEW: one on the line printer (device "LPT") at BBN, the other on the printer at USC. As with other RSEXEC features, device binding is available at the program level. For example, a program that reads from magnetic tape will function properly under the RSEXEC when it runs on a Host without a local mag-tape unit, provided the mag-tape device has been bound properly.

The user can take advantage of the distributed nature of the file system to increase the "accessibility" of certain files he considers important by instructing the RSEXEC to maintain *images* of them at several different Hosts. With the exception of certain special purpose files (e.g., the user's "message" file), the RSEXEC treats files with the same pathname relative to a user's composite directory as images of the same multi-image file. The user profile is implemented as a multi-image file with an image maintained at every component directory of the composite directory.*

* The profile is somewhat special in that it is accessible to the user only through the profile editing commands, and is otherwise transparent.

Implementation of the RSEXEC

The RSEXEC implementation is discussed in this section with the focus on approach rather than detail. The result is a simplified but nonetheless accurate sketch of the implementation.

The RSEXEC system is implemented by a collection of programs which run with no special privileges on TENEX Hosts. The advantage of a "user-code" (rather than "monitor-code") implementation is that ordinary user access is all that is required at the various Hosts to develop, debug and use the system. Thus experimentation with the RSEXEC can be conducted with minimal disruption to the TENEX Hosts.

The ability of the RSEXEC to respond properly to users' requests often requires cooperation from one or more remote Hosts. When such cooperation is necessary, the RSEXEC program interacts with RSEXEC "service" programs at the remote Hosts according to a pre-agreed upon set of conventions or protocol. Observing the protocol, the RSEXEC can instruct a service program to perform actions on its behalf to satisfy its user's requests.

Each Host in the RSEXEC system runs the service program as a "demon" process which is prepared to provide service to any remote process that observes protocol. The relation between RSEXEC programs and these demons is shown schematically in Figure 3.

The RSEXEC protocol

The RSEXEC protocol is a set of conventions designed to support the interprocess communication requirements of the RSEXEC system. The needs of the system required that the protocol:

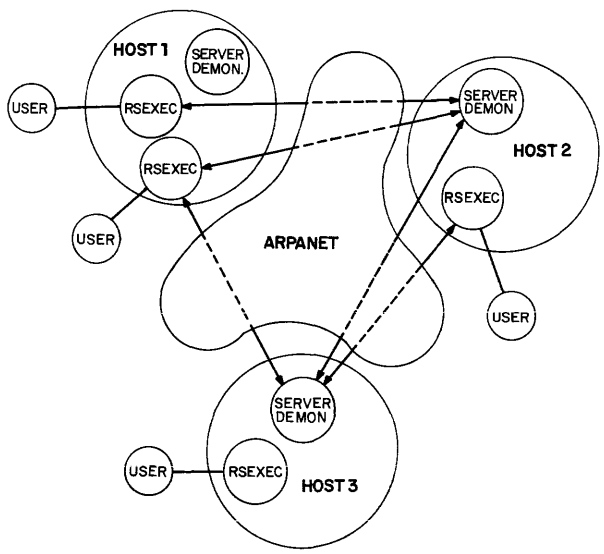


Figure 3—Schematic showing several RSEXEC programs interacting, on behalf of their users, with remote server programs

1. be extensible:
   As noted earlier, the RSEXEC is, by design, an evolutionary system.
2. support many-party as well as two-party interactions:
   Some situations are better handled by single multi-party interactions than by several two-party interactions. Response to an APPEND command when the files and the RSEXEC are all at different Hosts is an example (see below).
3. be convenient for interaction between processes running on dissimilar Hosts while supporting efficient interaction between processes on similar Hosts:
   Many capabilities of the RSEXEC are useful to users of non-TENEX as well as TENEX Hosts. It is important that the protocol not favor TENEX at the expense of other Hosts.

The RSEXEC protocol has two parts:

1. a protocol for initial connection specifies how programs desiring service (users) can connect to programs providing service (servers);
2. a command protocol specifies how the user program talks to the server program to get service after it is connected.

The protocol used for initial connection is the standard ARPANET initial connection protocol (ICP).[12] The communication paths that result from the ICP exchange are used to carry commands and responses between user and server. The protocol supports many-party interaction by providing for the use of auxiliary communication paths, in addition to the command paths. Auxiliary paths can be established at the user's request between server and user or between server and a third party. Communication between processes on dissimilar Hosts usually requires varying degrees of attention to message formatting, code conversion, byte manipulation, etc. The protocol addresses the issue of convenience in the way other standard ARPANET protocols have.[13,14,15] It specifies a default message format designed to be "fair" in the sense that it doesn't favor one type of Host over another by requiring all reformatting be done by one type of Host. It addresses the issue of efficiency by providing a mechanism with which processes on similar Hosts can negotiate a change in format from the default to one better suited for efficient use by their Hosts.

The protocol can perhaps best be explained further by examples that illustrate how the RSEXEC uses it. The following discusses its use in the WHERE, APPEND and LINK commands:

←WHERE (IS USER) JONES

The RSEXEC queries each non-local server program about user Jones. To query a server, it establishes connections with the server; transmits a "request for information about Jones" as specified by the protocol;

and reads the response which indicates whether or not Jones is a known user, and if he is, the status of his active jobs (if any).

←APPEND (FILE) DATA.NEW (TO FILE) DATA.OLD

Recall that the files DATA.NEW and DATA.OLD are at CASE and BBN, respectively; assume that the APPEND request is made to an RSEXEC running at USC. The RSEXEC connects to the servers at CASE and BBN. Next, using the appropriate protocol commands, it instructs each to establish an auxiliary path to the other (see Figure 4). Finally, it instructs the server at CASE to transmit the file DATA.NEW over the auxiliary connection and the server at BBN to append the data it reads from the auxiliary connection to the file DATA.OLD.

←LINK (TO TTY) 14 (AT SITE) CASE

Assume that the user making the request is at USC. After connecting to the CASE server, the RSEXEC uses appropriate protocol commands to establish two auxiliary connections (one "send" and one "receive") with the server. It next instructs the server to "link" its (the server's) end of the auxiliary connections to Terminal 14 at its (the server's) site. Finally, to complete the LINK command the RSEXEC "links" its end of the auxiliary connections to its user's terminal.

The RSEXEC program

A large part of what the RSEXEC program does is to locate the resources necessary to satisfy user requests. It can satisfy some requests directly whereas others may require interaction with one or more remote server programs. For example, an APPEND command may involve

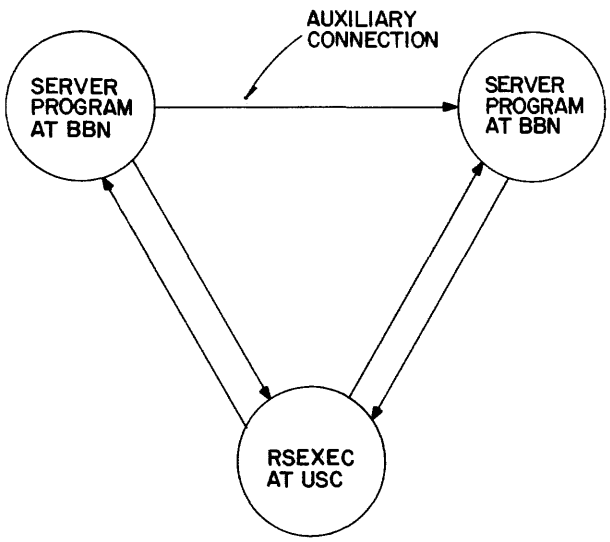


Figure 4—configuration of RSEXEC and two server programs required to satisfy and APPEND command when the two files and the RSEXEC are all on different Hosts. The auxiliary connection is used to transmit the file to be appended from one server to the other

interaction with none, one or two server programs depending upon where the two files are stored.

An issue basic to the RSEXEC implementation concerns handling information necessary to access files: in particular, how much information about non-local files should be maintained locally by the RSEXEC? The advantage of maintaining the information locally is that requests requiring it can be satisfied without incurring the overhead involved in first locating the information and then accessing it through the network. Certain highly interactive activity would be precluded if it required significant interaction with remote server programs. For example, recognition and completion of file names* would be ususable if it required direct interaction with several remote server programs. Of course, it would be impractical to maintain information locally about all files at all TENEX Hosts.

The approach taken by the RSEXEC is to maintain information about the non-local files a user is most likely to reference and to acquire information about others from remote server programs as necessary. It implements this strategy by distinguishing internally four file types:

1. files in the Composite Directory;
2. files resident at the local Host which are not in the Composite Directory;
3. files accessible via a bound device, and;
4. all other files.

Information about files of type 1 and 3 is maintained locally by the RSEXEC. It can acquire information about type 2 files directly from the local TENEX monitor, as necessary. No information about type 4 files is maintained locally; whenever such information is needed it is acquired from the appropriate remote server. File name recognition and completion and the use of partial pathnames is restricted to file types 1, 2 and 3.

The composite directory contains an entry for each file in each of the component directories specified in the user's profile. At the start of each session the RSEXEC constructs the user's composite directory by gathering information from the server programs at the Hosts specified in the user profile. Throughout the session the RSEXEC modifies the composite directory, adding and deleting entries, as necessary. The composite directory contains frequently accessed information (e.g., Host location, size, date of last access, etc.) about the user's files. It represents a source of information that can be accessed without incurring the overhead of going to the remote Host each time it is needed.

* File name recognition and completion is a TENEX feature which allows a user to abbreviate fields of a file pathname. Appearance of ESC in the name causes the portion of the field before the ESC to be looked up, and, if the portion is unambiguous, the system will recognize it and supply the omitted characters and/or fields to complete the file name. If the portion is ambiguous, the system will prompt the user for more characters by ringing the terminal bell. Because of its popularity we felt it important that the RSEXEC support this feature.

The RSEXEC regards the composite directory as an approximation (which is usually accurate) to the state of the user's files. The state of a given file is understood to be maintained by the TENEX monitor at the site where the file resides. The RSEXEC is aware that the outcome of any action it initiates involving a remote file depends upon the file's state as determined by the appropriate remote TENEX monitor, and that the state information in the composite directory may be "out of phase" with the actual state. It is prepared to handle the occasional failure of actions it initiates based on inaccurate information in the composite directory by giving the user an appropriate error message and updating the composite directory. Depending upon the severity of the situation it may choose to change a single entry in the composite directory, reacquire all the information for a component directory, or rebuild the entire composite directory.

The service program for the RSEXEC

Each RSEXEC service program has two primary responsibilities:

1. to act on behalf of non-local users (typically RSEXEC programs), and;
2. to maintain information on the status of the other server programs.

The status information it maintains has an entry for each Host indicating whether the server program at the Host is up and running, the current system load at the Host, etc. Whenever an RSEXEC program needs service from some remote server program it checks the status information maintained by the local server. If the remote server is indicated as up it goes ahead and requests the service; otherwise it does not bother.

A major requirement of the server program implementation is that it be resilient to failure. The server should be able to recover gracefully from common error situations and, more important, it should be able to "localize" the effects of those from which it can't. At any given time, the server may simultaneously be acting on behalf of a number of user programs at different Hosts. A malfunctioning or malicious user program should not be able to force termination of the entire service program. Further, it should not be able to adversely effect the quality of service received by the other users.

To achieve such resiliency the RSEXEC server program is implemented as a hierarchy of loosely connected, cooperating processes (see Figure 5):

1. The RSSER process is at the root of the hierarchy. Its primary duty is to create and maintain the other processes;
2. REQSER processes are created in response to requests for service. There is one for each non-local user being served.

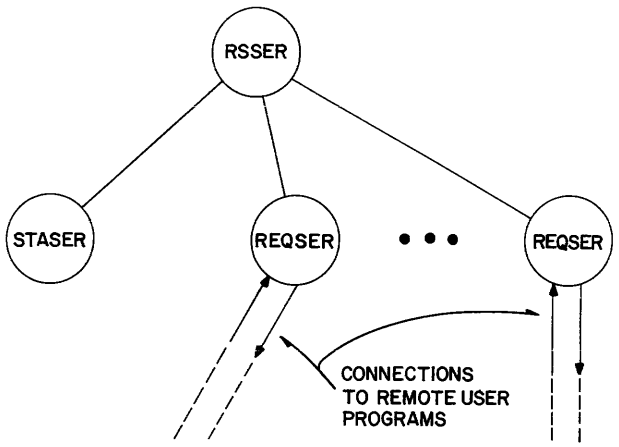


Figure 5—Hierarchical structure of the RSEXEC service program

3. A STASER process maintains status information about the server programs at other sites.

Partitioning the server in this way makes it easy to localize the effect of error situations. For example, occurrence of an unrecoverable error in a REQSER process results in service interruption only to the user being serviced by that process: all other REQSER processes can continue to provide service uninterrupted.

When service is requested by a non-local program, the RSSER process creates a REQSER process to provide it. The REQSER process responds to requests by the non-local program as governed by the protocol. When the non-local program signals that it needs no further service, the REQSER process halts and is terminated by RSSER.

The STASER process maintains an up-to-date record of the status of the server programs at other Hosts by exchanging status information with the STASER processes at the other Hosts. The most straightforward way to keep up-to-date information would be to have each STASER process periodically "broadcast" its own status to the others. Unfortunately, the current, connection-based Host-Host protocol of the ARPANET[16] forces use of a less elegant mechanism. Each STASER process performs its task by:

1. periodically requesting a status report from each of the other processes, and;
2. sending status information to the other processes as requested.

To request a status report from another STASER process, STASER attempts to establish a connection to a "well-known" port maintained in a "listening" state by the other process. If the other process is up and running, the connection attempt succeeds and status information is sent to the requesting process. The reporting process then returns the well-known port to the listening state so that it can respond to requests from other proc-

esses. The requesting process uses the status report to update an appropriate status table entry. If the connection attempt does not succeed within a specified time period, the requesting process records the event as a missed report in an appropriate status table entry.

When the server program at a Host first comes up, the status table is initialized by marking the server programs at the other Hosts as down. After a particular server is marked as down, STASER must collect a number of status reports from it before it can mark the program as up and useful. If, on its way up, the program misses several consecutive reports, its "report count" is zeroed. By requiring a number of status reports from a remote server before marking it as up, STASER is requiring that the remote program has functioned "properly" for a while. As a result, the likelihood that it is in a stable state capable of servicing local RSEXEC programs is increased. STASER is willing to attribute occasionally missed reports as being due to "random" fluctuations in network or Host responses. However, consistent failure of a remote server to report is taken to mean that the program is unusable and results in it being marked as down.

Because up-to-date status information is crucial to the operation of the RSEXEC system it is important that failure of the STASER process be infrequent, and that when a failure does occur it is detected and corrected quickly. STASER itself is programmed to cope with common errors. However error situations can arise from which STASER is incapable of recovering. These situations are usually the result of infrequent and unexpected "network" events such as Host-Host protocol violations and lost or garbled messages. (Error detection and control is performed on messages passed between IMPS to insure that messages are not lost or garbled within the IMP subnet; however, there is currently no error control for messages passing over the Host to IMP interface.) For all practical purposes such situations are irreproducible, making their pathology difficult to understand let alone program for. The approach we have taken is to acknowledge that we don't know how to prevent such situations and to try to minimize their effect. When functioning properly the STASER process "reports in" periodically. If it fails to report as expected, RSSER assumes that it has malfunctioned and restarts it.

Providing the RSEXEC to TIP users

The RSEXEC is available as a network executive program to users whose access to the network is by way of a TIP (or other non-TENEX Host) through a standard service program (TIPSER) that runs on TENEX Hosts.* To use the RSEXEC from a TIP a user instructs the TIP to initiate an initial connection protocol exchange with one of the TIPSER programs. TIPSER responds to the ICP by creating a new process which runs the RSEXEC for the TIP user.

* At present TIPSER is run on a regular basis at only one of the TENEX Hosts; we expect several other Hosts will start running it on a regular basis shortly.

CONCLUDING REMARKS

Experience with the RSEXEC has shown that it is capable of supporting significant resource sharing among the TENEX Hosts in the ARPANET. It does so in a way that provides users access to resources beyond the boundaries of their local system with a convenience not previously experienced within the ARPANET. As the RSEXEC system evolves, the TENEX Hosts will become more tightly coupled and will approach the goal of a multi-Host TENEX. Part of the process of evolution will be to provide direct support for many RSEXEC features at the level of the TENEX monitor.

At present the RSEXEC system is markedly deficient in supporting significant resource sharing among dissimilar Hosts. True, it provides mini-Hosts, such as TIPs, with a mechanism for accessing a small portion of the resources of the TENEX (and some non-TENEX) Hosts, enabling them to provide their users with an executive program that is well beyond their own limited capacity. Beyond that, however, the system does little more than to support inter-Host user-user interaction between Hosts that choose to implement the appropriate subset of the RSEXEC protocol. There are, of course, limitations to how tightly Hosts with fundamentally different operating systems can be coupled. However, it is clear that the RSEXEC has not yet approached those limitations and that there is room for improvement in this area.

The RSEXEC is designed to provide access to the resources within a computer network in a manner that makes the network itself transparent by removing the logical distinction between local and remote. As a result, the user can deal with the network as a single entity rather than a collection of autonomous Hosts. We feel that it will be through systems such as the RSEXEC that users will be able to most effectively exploit the resources of computer networks.

ACKNOWLEDGMENTS

Appreciation is due to W. R. Sutherland whose leadership and enthusiasm made the RSEXEC project possible. P. R. Johnson actively contributed in the implementation of the RSEXEC. The TENEX Group at BBN deserves recognition for constructing an operating system that made the task of implementing the RSEXEC a pleasant one.

REFERENCES

1. Roberts, L. G., Wessler, B. D., "Computer Network Development to Achieve Resource Sharing," *Proc. of AFIPS SJCC*, 1970, Vol. 36, pp. 543-549.
2. Heart, F. E., Kahn, R. E., Ornstein, S. M., Crowther, W. R., Walden, D. C., "The Interface Message Processor for the ARPA Computer Network," *Proc. of AFIPS SJCC*, 1970, Vol. 36.

3. McQuillan, J. M., Crowther, W. R., Cosell, B. P., Walden, D. C., Heart, F. E., "Improvements in the Design and Performance of the ARPA Network," *Proc. of AFIPS FJCC*, 1972, Vol. 41, pp. 741-754.
4. Roberts, L. G., "A Forward Look," *Signal*, Vol. XXV, No. 12, pp. 77-81, August, 1971.
5. Bobrow, D. G., Burchfiel, J. D., Murphy, D. L., Tomlinson, R. S., "TENEX, a Paged Time Sharing System for the PDP-10," *Communications of the ACM*, Vol. 15, No. 3, pp. 135-143, March, 1972.
6. *TENEX JSYS Manual—A Manual of TENEX Monitor Calls*, BBN Computer Science Division, BBN, Cambridge, Mass., November 1971.
7. Murphy, D. L., "Storage Organization and Management in TENEX," *Proc. of AFIPS FJCC*, 1972, Vol. 41, pp. 23-32.
8. Ornstein, S. M., Heart, F. E., Crowther, W. R., Rising, H. K., Russell, S. B., Michel, A., "The Terminal IMP for the ARPA Computer Network," *Proc. of AFIPS SJCC*, 1972, Vol. 40, pp. 243-254.
9. Kahn, R. E., "Terminal Access to the ARPA Computer Network," *Courant Computer Symposium 3—Computer Networks*, Courant Institute, New York, Nov. 1970.
10. Mimno, N. W., Cosell, B. P., Walden, D. C., Butterfield, S. C., Levin, J. B., "Terminal Access to the ARPA Network—Experience and Improvement," *Proc. COMPCON '73*, Seventh Annual IEEE Computer Society International Conference.
11. Walden, D.C., *TIP User's Guide*, BBN Report No. 2183, Sept. 1972. Also available from the Network Information Center at Stanford Research Institute, Menlo Park, California, as Document NIC #10916.
12. Postel, J. B., *Official Initial Connection Protocol*, Available from Network Information Center as Document NIC #7101.
13. Postel, J. B., *TELNET Protocol*, ARPA Network Working Group Request for Comments #358. Available from Network Information Center as Document NIC #9348.
14. Bhushan, A. K., *File Transfer Protocol*, ARPA Network Working Group Request for Comments #358. Available from Network Information Center as Document NIC #10596.
15. Crocker, S. D., Heafner, J. F., Metcalfe, R. M., Postel, J. B., "Function Oriented Protocols for the ARPA Computer Network," *Proc. of AFIPS SJCC*, 1972, Vol. 40, pp. 271-279.
16. McKenzie, A., *Host/Host Protocol for the ARPA Network*. Available from the Network Information Center As Document NIC #8246.

Avoiding simulation in simulating computer communication networks*

by R. VAN SLYKE, W. CHOU, and H. FRANK

Network Analysis Corporation
Glen Cove, New York

INTRODUCTION

Computer communication networks are complex systems often involving human operators, terminals, modems, multiplexers, concentrators, communication channels as well as the computers. Events occur at an extraordinary range of rates. Computer operations take place in microseconds, modem and channel operations take place on the order of milliseconds, while human interaction is on a scale of seconds or minutes, and the mean time between failures of the various equipments may be on the order of days, weeks, or months. Moreover, many systems are being designed and built today which involve thousands of terminals, communication links, and other devices working simultaneously and often asynchronously.

Thus, in addition to the ordinary difficulties of simulation, the particular nature of computer communication networks gives rise to special problems. Blind brute force simulations of systems of this degree of complexity usually result in large, unwieldy, and unverifiable simulation programs only understood by the creator (if by him) and/or statistically insignificant estimates due to unacceptable computational time per sample.

In order to obtain useable results careful attention must be paid to determining the key features of the system to be considered *before* the simulation is designed and approximating, ignoring or otherwise disposing of the unimportant aspects. In this paper, we discuss three approaches we have found useful in doing this. While these ideas may seem obvious especially in retrospect, our experience has been that they have often been overlooked. The first technique takes advantage of situations where the significant events occur infrequently. A common example is in studies involving infrequent data errors or equipment failures. The second idea is the converse of the first and arises from simulations in which the significant events occur most of the time and the rare events are of less importance. The final idea is to make as much use as possible of analytic techniques by hybrid simulations reserving simulation for only those aspects not amenable to analysis. In the next three sections we give many illustrations drawn from practice of these ideas.

* This work was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC 15-73-C-0135.

IMPORTANT RARE EVENTS

In a computer communication environment there coexist various events occurring at widely varying rates. Often the activities of interest are events which occur rarely or are the result of composite events consisting of a relatively infrequent sequence of events. This suggests the possibility of ignoring or grossly simplifying the representation of interim insignificant events. For this the simulator has one big advantage over the observer of a real system in that he can "predict" the future. In a real operational environment there is no way to predict exactly the next occurrence of a randomly occurring event; in a simulated environment often such an event can be predicted since the appropriate random numbers can all be generated in advance of any calculation (if their distribution is not affected by generating the next random number). Once the future occurrence time is known, the simulated clock can be "advanced" to this time. The intervening activities can either be ignored or handled analytically. Extending this idea further the clock in the usual sense can be dispensed with and time can be measured by the occurrence of the interesting events much as in renewal theory.

Example 1—Response time simulation of terminal-concentrator complexes

In performing response time simulations for terminals connected by a multidrop polled line connected to a concentrator or central computer one can often save large amounts of computer time by avoiding detailed simulation of the polling cycle in periods of no traffic. Unless the traffic load is very heavy, large time intervals (relative to the time required for polling) between two successive messages will occur rather frequently. In a real transmission system, the concentrator just polls terminals during this period; if the simulation does the same, large

amounts of computer time may be wasted. In a simulation program, after one message is transmitted, the time the next message is ready to be transmitted can be predicted.

Therefore, the program may advance the clock to the time when the next message is ready and determine which terminal should be polled at that time (ignoring the polling that goes on in the interim).

For each terminal which is not waiting for a reply message from the concentrator, the program can predict the time when it has an inbound message to send to the concentrator by generating as the inter-message time at the terminal, a random number based on input information. For the concentrator, the program can predict in the same way when it has a reply message to deliver. Among these messages the time the first message is ready to send is determined. The time elements involved in polling one terminal is usually deterministic and known. Therefore, if T_1 is the time at the end of the current complete message transaction, and T_2 is the time when the next message begins to wait for transmission, the terminal being polled at T_2 can be determined. Let this terminal be ID. The concentrator did not start to poll ID at exactly T_2 but started at some earlier time T_3. In the simulation program, the clock can be advanced to T_3 and no activities between T_1 and T_2 need be simulated. ID is polled at T_3 and the simulation is resumed and continues until the transaction for the next message is completed.

INSIGNIFICANT RARE EVENTS

In contrast to the situation in the previous section where rare events were of critical importance, often activities resulting from rare events are of little interest in the simulation. These often numerous insignificant rare events can lead to difficulties both in the coding of the simulation and its execution. Firstly, an inordinate amount of coding and debugging time can be spent in representing all the various types of insignificant events. On the other hand, in the operation of the system, the rare events often can occur only when more significant events of much higher frequency also occur. Thus, each time such frequently occurring events are realized, one must check for the occurrence of the rare events which can grossly increase the computer time of the simulation. To reduce these undesirable effects, one may either eliminate the insignificant rare events from the simulation completely, or at least simplify the procedures relevant to the insignificant events so as to reduce the coding efforts and shorten running time.

Example 2—Detecting and recovering from transmission errors[3]

In using a simulation program to predict terminal response time on a polled multidrop line, the transmission of all messages and their associated control sequence is simulated. For each type of transmission, errors may appear in a variety of different forms, such as no acknowledgment to a message transmission, unrecognized message, the loss of one or more crucial characters, and parity errors. For each of them, there may be a different way to react and a different procedure for recovery. In a typical system, there are over forty different combinations of errors and procedures for their treatment. Therefore, it is impractical and undesirable to incorporate every one of the procedures into the program. Even if all the error handling procedures were simulated in the program, there is only limited effect on the response time under normal conditions (i.e., the message transmission error rate and the terminal error rate should be at least better than 10^{-3}) but the program complexity and the computer running time would both be increased by a factor of 10 to 100. With respect to response time, the errors can be easily handled in the program. Whatever the form of error, it must stem from one of the following three sources: a line error during the transmission of the original message; an error caused by a malfunctioning terminal; or a line error during the transmission of an acknowledgment. The error prolongs the response time by introducing the extra delay caused by either the retransmission of the message and its associated supervisory sequences or by a time-out. If the timer is expired and the expected reply or character has not been detected by the concentrator, action is taken to recover from the error. This action may be a retransmission, a request for retransmission, or a termination sequence. The time-out is always longer than the total transmission time of the message and the associated supervisory sequences. Rather than considering the exact form of an error and its associated action, whenever the simulated transmission system detects an error, a delay time equal to the time-out is added to the response time.

With these simplifications introduced into the simulation program, the error handling procedures still require a large percentage of overall computer time. When a leased line is initially installed, the errors are very high mainly due to the maladjustment of the hardware. However, when all the bugs are out, the error rate has a very limited effect on the response time. Therefore, for a slight sacrifice in accuracy the program may be run neglecting errors to save computer time.

Example 3—Network reliability in the design and expansion of the ARPA computer network (SJCC; 1970), (SJCC; 1972)

It was desired to examine the effect on network performance of communication processor and link outages.[10,11] The initial design was based on the deterministic reliability requirement that there exist two node disjoint communication paths between every pair of communication nodes (called Interface Message Processors, IMPs) in the net. A consequence of this is that in order for the network to become disconnected at least two of its elements, communication links or IMPs must fail. Since the

IMPs and communication lines are relatively reliable (availability on the order of .98) disconnections are quite rare so a simulation model appropriate for the analysis of the communication functions of the network would have to run an enormously long time before a single failure could be expected, let alone two failures. To illustrate the ideas involved, let us consider the simplified problem of determining the fraction of the time, $h(p)$, the network is disconnected given that links are inoperable a fraction p of the time. (We assume for purposes of illustration that IMPs don't fail. Depicted in Figure 1 is a 23 IMP, 28 link version of the ARPA network.) An obvious method of determining $h(p)$ would be to generate a random number for each link i of the network; if the number r_i is less than p the link is removed, otherwise it is left in. After this is done for each link the connectivity of the remaining network is determined. This is done many times and the fraction of the times that the resulting network is disconnected provides an estimate of $h(p)$. Unfortunately, a good part of the time $r_i > p$ for each i, occasionally $r_i \leqq p$ would hold for one i, and only rarely would $r_i \leqq p$ for two or more i; thus, many random numbers would be generated to very little purpose. An effective way to sort out the significant samples is by using a Moore-Shannon expansion of $h(p)$. We have

$$h(p)=\sum_{0}^{m} C(k)p^k(1-p)^{m-k}$$

where m is the number of links and $C(k)$ is the number of distinct disconnected subnetworks obtained from the original network by deleting exactly k links. Clearly, $0 \leqq C(k) \leqq \binom{m}{k}$. Thus we have partitioned the set of possible events into those in which 0 links failed, 1 link failed and so on. In practice, it turns out that only a few of these classes of events are significant, the values of $C(k)$ for the remaining classes are trivially obtained. Thus it takes at least $n-1$ links to connect a network with n nodes so that $C(k)=\binom{m}{k}$ for $k=m-n, \cdots, m$. Similarly, $C(0)=C(1)=0$ for the ARPA network because at least 2 links must be removed before the network becomes disconnected. For the network depicted in Figure 1 where $m=28$, $n=23$ the only remaining terms which are not immediately available are $C(2)$, $C(3)$, $C(4)$, $C(5)$, and $C(6)$ (See Table I). $C(2)$ and $C(3)$ can be obtained rather quickly by enumeration and the number of subtrees is obtainable for formula giving $C(6)$, thus leaving only $C(4)$ and $C(5)$ undetermined. *These* can be obtained by sampling; in general, by stratified sampling.

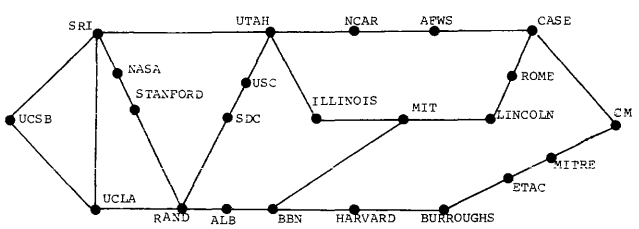


Figure 1—Network for reliability analysis

TABLE I—Exactly Known $C(k)$ for 23 Node 28 Link ARPA Net

Number of links Operative	Number of links Failed	Number of Nets	Number of Failed Nets	Method of Determination
0	28	1	1	a
1	27	28	28	
2	26	378	378	
3	25	3276	3276	
4	24	20475	20475	
5	23	98280	98280	
6	22	376740	376740	
7	21	1184040	1184040	
8	20	3108105	3108105	
9	19	6906900	6906900	
10	18	13123110	13123110	
11	17	21474180	21474180	
12	16	30421755	30421755	
13	15	37442160	37442160	
14	14	40116600	40116600	
15	13	37442160	37442160	
16	12	30421755	30421755	
17	11	21474180	21474180	
18	10	13123110	13123110	
19	9	6906900	6906900	
20	8	3108105	3108105	
21	7	1184040	1184040	a
22	6	376740	349618	b
23	5	98280	?	
24	4	20475	?	
25	3	3276	827	c
26	2	378	30	c
27	1	28	0	d
28	0	1	0	d

Notes: a: not enough links to connect 23 nodes.
b: number of trees calculated by formula.
c: enumerated.
d: less failed links than minimum cutset.

Thus we have not only been able to dispose of frequently occurring but unimportant events corresponding to $C(0)$, and $C(1)$ but also to the rare and unimportant events corresponding to $C(7)$ through $C(28)$.

HYBRID SIMULATIONS

Hybrid simulations refer to models which involve both analytic techniques and simulation techniques. Since analytic techniques are often more accurate and faster than simulation, it is usually worth the effort to model as much of the system as is possible analytically.

A complete computer communication system is in general a composite system of many complicated subsystems, involving the interaction of a variety of computer subsystems, terminal subsystems and functionally independent transmission subsystems. Many of these systems are complicated and not well defined. For example, there is no simple macroscopic characterization of the hardware and software in a computer system or of how the hard-

ware and software interact with each other while dealing with message traffic. It is therefore practically speaking impossible to simulate a whole system without using some analytic representations or employing some approximations. In place of simulation, the functioning of a subsystem can often be represented by an empirical model.

In deriving analytic models, however, simplification is always necessary to make formulation manageable. For example, message flow is often assumed to follow a Poisson pattern, and the message length distribution is sometimes approximated by an exponential distribution. Analytic approaches can give quite acceptable results, if the problem is *properly* modeled.

Example 4—Throughput analysis of distributed networks

In the topological design of distributed computer networks such as the ARPA network a rapid method for analyzing the throughput capacity of a design under consideration is essential. Analyzing the traffic capacity in detail is a formidable task, involving modeling the traffic statistics, queuing at the IMPs, the routing doctrine, error control procedures, and the like for 30 or 40 IMPs and a comparable number of communication links. In order to examine routing methods, Teitelman and Kahn[9] developed a detailed simulation model with realistic traffic routing and metering strategy capable of simulating small versions of the ARPA Network. Since, in the design of the topology, traffic routing is performed for hundreds of possible configurations, such a simulation is impractical so that a deterministic method of traffic analysis was developed[1] requiring orders of magnitude less in computing times and thus allowing its repeated use. The two models were developed independently. Based on a 10 IMP version of the ARPA Network shown in Figure 2, average node-to-node delay time is plotted versus network traffic volume in Figure 3. The curve is from the deterministic analytic model while the x's are the results of simulation. The analytic results are more conservative than the simulation results due to the simplification introduced in the analytic model but the results are strikingly close.[6]

The topological design is chosen with respect to some predicted traffic distribution from the computers in the network. Since such predictions are highly arbitrary, it is

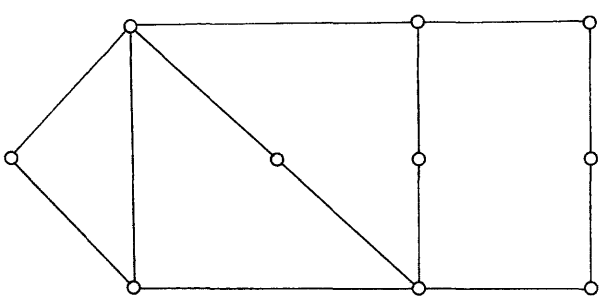


Figure 2—Network for throughput analysis

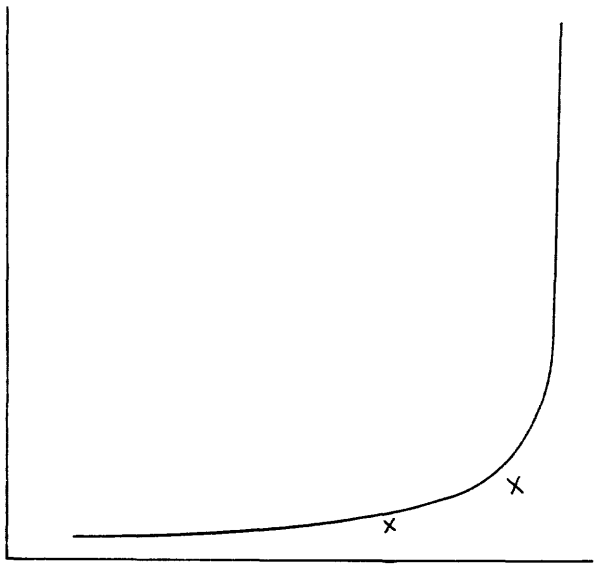


Figure 3—Throughput for network shown in Figure 2

necessary to determine how robust the design is with respect to errors in the prediction of traffic. This was done by choosing the average traffic levels at each computer randomly[4] and using the analytic throughput analyzer to evaluate the network for varying input traffic patterns. Thus, the availability of a quick throughput analyzer allowed a more macroscopic simulation than would have been computationally feasible if the throughput would have had to be calculated by simulation.

Example 5—Time spent at a concentrator[3]

In the study of a general centralized computer communication network, one of the most difficult tasks is to estimate accurately the time span between an inbound message's arrival at the concentrator and the time the reply is ready for transmission back to the terminal which generated the inbound message.

The system we model for this example has several concentrators and one central computer. Several multidrop lines, called regional lines, are connected to a concentrator, called a regional concentrator terminal (RCT). Each RCT is connected to the central computer system (CPS) via a high speed trunk line. After an inbound message reaches the RCT it undergoes the following processes before the reply is transmitted back to the terminal: (1) waiting for processing by the RCT, (2) processing by the RCT, (3) waiting for access to the inbound trunk (from RCT to CPS), (4) transmitting on the inbound trunk, (5) waiting for processing by the CPS, (6) CPS processing the inbound message to obtain the reply, (7) reply waiting for access to the outbound trunk, (8) transmitting on the outbound trunk, (9) waiting for RCT to process, (10) processing by the RCT, (11) waiting for access to the regional line.

In our application, the RCT is under-utilized while the CPS is highly utilized. Items (1), (2), (9) and (10) are times relevant to the RCT and are negligibly small. Items (4) and (8) are transmission times and can be obtained by dividing message length by the line speed. The network control procedure in our model does not allow a second message to be delivered if the reply to the first has not been returned. Therefore, there is no waiting for the reply to access the regional line, and item (11) is zero. A combination of an analytic model and an analytic function is used to obtain item (3), as shown below. An empirical distribution is used for the combination of items (5) and (6). An analytic function is used for determining item (7).

When an inbound message arrives at the RCT, it is processed and queued at the output buffer for transmission to the CPS. The waiting time for access to the inbound trunk line from the RCT to the CPS depends on the traffic load of other regional lines connected to the same RCT. To determine how long the message must wait, the number of regional lines having a message waiting at the RCT for transmission to the CPS must be determined. We assume that the average transaction rate (which is the sum of the transaction rates of all terminals connected to the same RCT) on the trunk connecting the RCT to the CPS is known and that the message arrivals to the trunk have a Poisson distribution and their length has an exponential distribution. Then, the probability of N or more messages is P^N where P is the trunk utilization factor, i.e., the ratio of average total traffic on the inbound trunk to the trunk speed. A random number with this distribution can be generated from one uniformly distributed random number by inverting the cumulative distribution function.

These messages include those waiting at terminals and those at the RCT. A random number for each of these inbound messages is generated to determine which regional line the message is from. The number of regional lines having inbound messages in waiting is thus determined. The network flow control procedure in our model allows no more than one inbound message from each of these regional lines at the RCT. Therefore, the number of non-empty buffers is no more than the number of lines having inbound messages in waiting. Conservatively, the former number is set to be equal to the latter. The waiting for access to the trunk is then equal to the number so obtained multiplied by the time required to transmit one average inbound message from the RCT to the CPS.

The time interval between the time at the end of the transmission of an inbound message to the CPS and the time the reply has returned to the RCT, depends on the CPS occupancy and the trunk utilization. The CPS occupancy is a direct consequence of the transaction rate of the whole system. The trunk utilization is a direct result of the transaction rate input from all the regional lines connected to the RCT. In our model the time waiting for processing at the CPS and the processing time for each message is given as an empirical distribution. With the help of a random number generator, the time spent at CPS can be determined. The time a reply or an outbound message spends in waiting for the trunk is conservatively estimated by using the following analytic formula

$$\text{Probability (waiting time} > t) = Pe^{-(1-P)t/AVS}$$

where p is the trunk utilization factor and AVS is the average time required to transmit an outbound message.[8]

CONCLUSION

Computer communication networks can be quite large and immensely complex with events occurring on a vast time scale. Rarely can all aspects of such a system be simulated at once for any but the most trivially small and simple systems. Therefore, simulations must be designed for relatively restricted purposes and careful attention must be paid to ways and means of simplifying and ignoring factors not directly pertinent to the purposes. Here we have suggested three ways to avoid unnecessary simulation: (1) by not simulating in detail insignificant events which occur at a much higher rate than the significant ones, (2) by ignoring rare events of little practical significance and (3) by using hybrid simulations with extensive use of analytic modeling where applicable. A number of examples drawn from practice illustrate the application of these ideas to computer communication systems.

REFERENCES

1. Chou, W., Frank, H., "Routing Strategies for Computer Network Design," *Proc. of the Symposium on Computer-Communications Networks and Teletraffic*, Polytechnic Institute of Brooklyn, 1972.
2. Frank, H., Chou, W., "Routing in Computer Networks," *Networks*, 1. No.2 pp. 99-112, 1971.
3. Frank H., Chou W., "Response Time/Capacity Analysis of a Computer-Communications Network," *Infotech State of the Art Reports: Network Systems and Software*, Infotech, Maidenhead, Berkshire, England, 1973.
4. Frank, H., Chou, W., "Properties of the ARPA Computer Network," to be published, 1973.
5. Frank, H., Chou, W., Frisch, I. T., "Topological Considerations in the Design of the ARPA Computer Network," *SJCC Conf. Rec. 36*, pp. 581-587 (1970).
6. Frank H., Kahn, R., Kleinrock, L., "Computer Communication Network Design—Experience with Theory and Practice," *Proceedings of the Spring Joint Computer Conf.*, AFIPS Press, 1972.
7. Kershenbaum, A., Van Slyke, R. M., "Recursive Analysis of Network Reliability," *Networks*, Jan. 1973.
8. Saaty, T. L., *Elements of Queueing Theory*, McGraw-Hill, New York 1961.
9. Teitelman, W., Kahn, R. E., "A Network Simulation and Display Program," *Third Princeton Conference on Information Sciences and Systems*, 1961.
10. Van Slyke, R. M., Frank, H., "Network Reliability Analysis—I" *Networks*, 1, No. 3, 1972.
11. Van Slyke, R. M., Frank, H., "Reliability in Computer-Communications Networks," *Proc. of 1971 ACM Winter Simulation Conference.*

An implementation of a data management system on an associative processor

by RICHARD MOULDER

Goodyear Aerospace Corporation
Akron, Ohio

INTRODUCTION

Recent years have witnessed a widespread and intensive effort to develop systems to store, maintain, and rapidly access data bases of remarkably varied size and type. Such systems are variously referred to as Data Base Management Systems (DBMS), Information Retrieval Systems, Management Information Systems and other similar titles. To a large extent the burden of developing such systems has fallen on the computer industry. The problem of providing devices on which data bases can be stored has been reasonably well solved by disc and drum systems developed for the purpose and commercially available at the present time. The dual problem of providing both rapid query and easy update procedures has proved to be more vexing.

In what might be called the conventional approach to DBMS development, sequential processors are employed and large, complex software systems developed to implement requisite data processing functions. The results are often disappointing.

A promising new approach has been provided by the appearance of the Associative Processor (AP). This new computer resource provides a true hardware realization of content addressability, unprecedented I/O capability, and seems ideally suited to data processing operations encountered in data management systems. Many papers have been written about the use of associative processors in information retrieval and in particular about their ability to handle data management problems.[1] To the best of the author's knowledge no actual system has been previously implemented on an associative processor.

This paper will detail the author's experience to date in implementing a data management system on an associative processor. It should be noted that the data management system to be described in the following pages is not intended as a marketable software package. It is research oriented, its design and development being motivated by the desire to demonstrate the applicability of associative processing in data management and to develop a versatile, economical data base management system concept and methodology exploiting the potential of an associative processor and a special head per track disc. The remainder of this paper will describe the hardware configuration of the author's facility, the data storage scheme, the search technique, the user oriented data definition and manipulation languages, and some of the benefits and problems encountered in utilizing an Associative Processor for DBMS.

HARDWARE CONFIGURATION

This section will describe the computer facility employed by the author and available to Goodyear Aerospace Corporation customers. The STARAN* Evaluation and Training Facility (SETF) is a spacious, modern, well equipped facility organized around a four-array STARAN computer. Of particular significance to DBMS efforts is the mating of STARAN to a parallel head per track disc (PHD). The disc is connected for parallel I/O through STARAN's Custom Input/Output Unit (CIOU). Of the 72 parallel channels available on the disc, 64 are tied to STARAN. The switching capability of the CIOU allows time sharing of the 64 disc channels within and between the STARAN arrays. (The SETF STARAN is a 4 array machine, each array containing 256 words of 256 bits each.) The switching provision allows simulations of AP/PHD systems in which the number of PHD channels are selectable in multiples of 64 up to 1024 total parallel channels.

In order to provide for rather general information handling applications and demonstrations, the STARAN is integrated, via hardware and software, with an XDS Sigma 5 general purpose computer, which in turn is integrated with an EAI 7800 analog computer. Both the STARAN and the Sigma 5 are connected to a full complement of peripherals. The Sigma 5 peripherals include a sophisticated, high-speed graphics display unit well suited for real time, hands-on exercising of the AP/PHD DBMS. A sketch of the SETF is given in Figure 1.

The Custom Input/Output Unit (CIOU) is composed of two basic sections. One section provides the communication between the Sigma 5 and the AP. This communication is accomplished by the Direct Memory Access capa-

* TM. Goodyear Aerospace Corporation, Akron, Ohio 44315

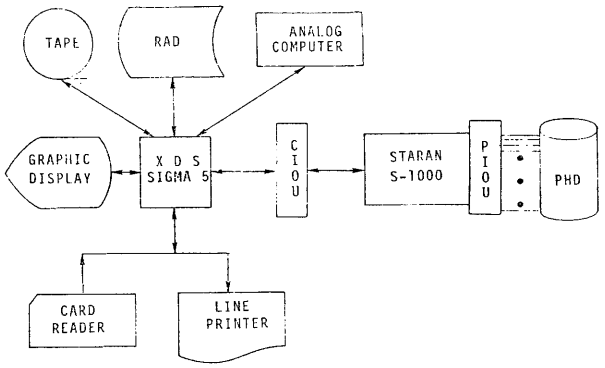


Figure 1—STARAN evaluation and training facility

bility of the Sigma 5 computer. The second section of the CIOU is composed of the Parallel Input/Output Unit (PIOU). This unit interfaces the AP with the PHD. As previously mentioned the PHD is composed of 72 tracks of which 64 tracks are tied to STARAN. The disc is composed of one surface which is sub-divided into 384 sectors consisting of 64 tracks, each track having a bit capacity of 256 bits per sector. The time per revolution for the disc is approximately 39 msec. It should be noted that Goodyear Aerospace Corporation did not manufacture this disc and that there are several manufacturers providing parallel head per track devices. Given this hardware configuration, a data storage scheme was developed.

DATA STORAGE SCHEME

A hierarchical data structure was chosen for our initial effort since it is probably the structure most widely used for defining a data base. In order to utilize the parallel search capabilities of the AP and the parallel communication ability of the AP/PHD system, it was decided to reduce the hierarchical structure to a single level data base. The method used was similar to the one suggested by DeFiore, Stillman, and Berra.[1] In this method each level of the hierarchy is considered a unique record type. Associated with each record type are level codes indicating the parentage of the particular record. The association of level numbers with record type is purely logical and does not imply or require a correspondingly structured data storage scheme.

It should be noted that each record contains a level code for each of the preceding levels in the hierarchical structure. The different records are stored on the PHD in a random fashion. No tables or inverted files are introduced. Only the basic data file is stored on the disc. Since we have an unordered and unstructured data base, it is necessary to search the entire data base in order to respond to a query. The searching of the entire data base presents no significant time delays because of the parallel nature of both the AP and the PHD. This data storage scheme was selected because it provides an easy and efficient means of updating the data base due to the lack of multiplicity of the data. It should be emphasized that this scheme is not necessarily the best approach for all data bases. It appears to be well suited for small to moderately sized data bases having hierarchical structure with few levels. Data bases which are to be queried across a wide spectrum of data elements are especially well suited for this type of organization since all data elements can participate in a search with no increase in storage for inverted lists or other indexing schemes. Since there is no ordering of the data base and no multiplicity of data values, updating can be accomplished very rapidly with a minimum amount of software.

SAMPLE DATA BASE

The data base selected for the AP/PHD DBMS development program is a subset of a larger data base used by the SACCS/DMS and contains approximately 400.000 characters. It is a four-level hierarchical structure composed of command, unit, sortie, and option records.

A tree diagram of the selected data base is shown in Figure 2.

Figure 3 gives an example of an option record.

Definitions for the option record are as follows:

(1) Record Type—indicates the type of record by means of the level number; also indicates the word number of this record. If a logical record requires more than one array word, these words will be stored consecutively with each word being given a word number.
(2) Level Codes—these codes refer to the ancestry of the particular record. They consist of an ordinal number.
(3) Data Item Names—These are elements or fields of one particular record type. Each field has a value. There may not be any two or more fields with the same name.
(4) Working Area—This is that portion of an array word not being used by the data record.

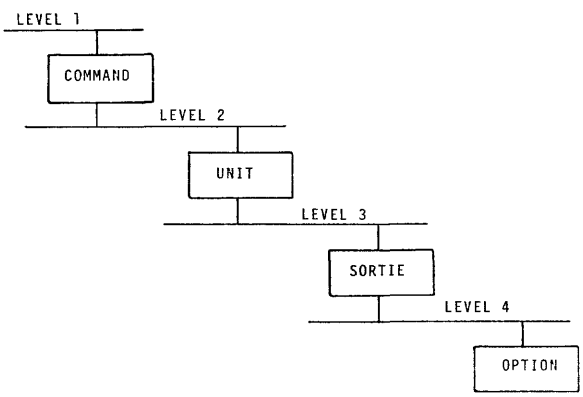


Figure 2—Tree diagram of selected data base

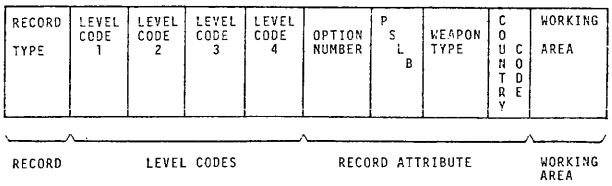


Figure 3—Option record layout

SEARCH TECHNIQUE

In order to search the entire data base, the data will be arranged in sectors consisting of 64 tracks. Each track will be composed of 256 bits. The STARAN Associative Processor employs 256 bit words; each associative array being composed of 256 such words. A PHD sector will therefore contain one fourth of an array load. In our current system we are utilizing the first 64 words of the first array for data storage. With the addition of more tracks on the PHD we could utilize a greater portion of an array with no additional execution time due to the parallel nature of our input. (The AP can of course time share the 64 tracks currently available in order to simulate larger PHD's.) With this technique it is possible to read in a sector, perform the required search, then read in another sector, perform a search, continuing in this fashion until the entire data base is searched.

For our current PHD, the time for a sector to pass under the read/write heads is approximately 100 μsec. Preliminary studies have shown that complex or multi-searches can be performed in this time span. Utilizing this fact it should be possible to read every other sector on one pass of our disc. With two passes the entire data base will have been searched. This assumes that the entire data base is resident on one surface. Additional surfaces would increase this time by a factor equal to the number of surfaces.

Figure 4 shows an associative processor array and the every other sector scheme employed when searching the PHD. It should be emphasized that due to the hierarchical structure of the data base more than one data base search will be required to satisfy a user's request and that the time to search and output from the associative array is variable. In order to minimize the time to search the disc when a 100 μsec is not sufficient time to process an

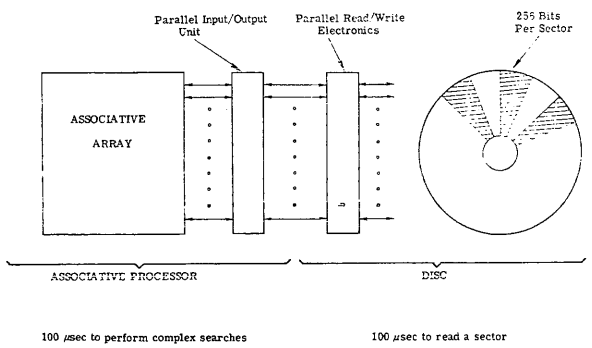


Figure 4—AP/PHD tie-in

array, a bit map of searched sectors is kept. This bit map will allow minimization of rotational delay.

DATA BASE MANAGEMENT SYSTEM

The DBMS software is composed of four modules: the Data Definition, File Create, Interrogation and Update modules. Since our DBMS is intended for research and not a software product package, we have not incorporated all the features that would be found in a generalized DBMS. Our system will provide the means to create, interrogate and update a data base. The following discussion will briefly describe each module.

Data definition module

This module defines a file definition table. This table relates the data structure as the user views it, to a data storage scheme needed by the software routines. Every data base will have only one file definition table. This table will contain the following information.

(1) File name—the name of the particular data base
(2) Record name—the name of each logical record type
(3) Data items—the name of each attribute in a given logical record type
(4) Synonym name—an abbreviated record or attribute name
(5) Level number—a number indicating the level of the record
(6) Continuation number—this number is associated with each attribute and indicates which AP word of a multi-word record contains this attribute
(7) Starting bit position—the bit position in an AP word where a particular attribute starts
(8) Number of bits—the number of bits occupied by an attribute in an AP word
(9) Conversion code—a code indicating the conversion type: EBCDIC or binary

The language used to define the data definition table is called the Data Description Language (DDL). The DDL used in our system has been modeled after IBM's Generalized Information System DDL.

File create module

Once the file definition table has been created, the file create module is run. This module populates a storage medium with the data values of a particular data base.

Interrogation module

This module is the primary means for interrogating the data base. Since our DBMS is intended to demonstrate the applicability of Associative Processors in DBMS, we

have developed a user oriented language. This Data Manipulation Language (DML) is modeled after SDC's TDMS system. It is conversational in nature and requires no programming experience for its utilization. At present there is only one option in the interrogation module implemented in our DBMS. This option is the print option.

The print option is used to query the data base. The user is able to query the data base using any data item(s) as his selection criteria. The form of a query follows:

Print Name(s) Where Conditional Expression(s)

- Name(s)—Any data item name or synonym
- Conditional Expression (CE)—Selection criteria for searches

The form of the CE is:

Data Item Name Relational Operator Value

Relational Operators
EQ—Equal to
NE—Not equal to
LT—Less than
LE—Less than or equal to
GT—Greater than
GE—Greater than or equal to

Logical Operators (LO)—link conditional expressions together:

CE Logical Operator CE Logical operators
- AND—Logical And
- OR—Inclusive Or

At present the output from a query will be in a fixed format. It is anticipated that a formatting module will be added to the system. An example of the print option and its output are shown below.

Query
Print Unit Name, Sortie Number where
Option Number EQ 6 and Weapon Type NE Mark82
and Country Code EQ USA*

Output
Unit Name—303BW
Sortie Number—1876

Update module

This module performs all the updating of the data base. There are four options available to the user. These are change, delete, add, and move. These options are described below.

Change option

The change option performs all edits to the data base. All records satisfying the change selection criteria are updated. The form of the change option follows:

Change Name to Value Where Conditional Expression(s)

- Name—Any data item name or synonym
- Value—Any valid data value associated with the above name; this is the new value
- Conditional Expression—same as the print option.

An example of the change option follows:

Change Unit Name to 308BW Where
Unit Number EQ 1878 And Option Number GE 6*

It should be noted that all records that satisfy the unit number and option number criteria will have their unit names changed to 308BW.

Delete option

This option will delete all records that satisfy the conditional expression. All subordinate records associated with the deleted records will also be deleted. The form and an example of the delete option follows:

Delete Record Name Where Conditional Expression(s)

Record Name—any valid record name (e.g. command, unit, sortie, option)
Conditional Expression(s)—same as print option

Example:
Delete Sortie Where Sortie Number EQ 7781*

In this example all the sortie records with sortie number equal 7781 will be deleted. In addition all option records having a deleted sortie record as its parent will also be deleted.

Add option

This option adds new records to the data base. Before a new record is added to the data base, a search will be initiated to determine if this record already exists. If the record exists a message will be printed on the graphic display unit and the add operation will not occur. The form of the add option follows.

Add Record Name to Descriptor Where Description of Data

- Record Name—any valid record name (i.e. command, unit, sortie, option)
- Descriptor—special form of the conditional expression in which 'EQ' is the only allowed relational

operator; the descriptor describes the parentage of the record being added to the data base.
- Description of Data—This item is used to define the data base; the form of this item is the same as a conditional expression with the requirement that 'EQ' is the only allowed relational operator.

An example of the add option follows:

Add Unit To Name EQ 8AF Where
Unit Name EQ 302BW And Unit Number EQ 1682
And Aircraft Possessed EQ 85*

In this example the command record for the 8AF must be in the data base before this unit record can be added to the data base. In the case of a command record being added to the data base, the descriptor field is omitted from the above form.

Move option

This option allows the user to restructure the data base. The move option will change the level codes of the affected records. The affected records are those satisfying the conditional expression, in the move command, as well as all subordinate records. The move option has the following form:

Move Record Name to Descriptor Where Conditional Expression

- Record Name—any record name
- Descriptor—this item describes the new parentage of the record that is to be moved; the form is the same as the add option
- Conditional Expression—same as the print option

An example of the move option follows:

Move Sortie to Name EQ 12AF and Unit Number EQ 2201
Where Sortie Number EQ 41 and Option Number GT 12*

In this example all sortie records which have a sortie number equal to 41 and an option record with option number greater than 12 will have its parentage (i.e., level codes) changed to the 12AF and 2201 unit. Also included in this restructuring will be all option records which have the changed sortie record as a parent.

Status

Currently the Data Definition and the Create Module are operational and our sample data base has been created and stored on the PHD. A simplified version of the Interrogation Module is running and performing queries using the AP and PHD. In this version the translator for the query is executed on the Sigma 5 and a task list is constructed. This task list is transmitted to the AP where the desired searches are performed and the output of the searches are transmitted to the Sigma 5 via Direct Memory Access. In the simplified version of the Interrogation Module no optimation of the software has been attempted. At the moment the number of disc sectors skipped between reads during a search of the data base is governed by the size of the task list and is not modified by the actual time available to process the task list. No sector bit map has been implemented. It is anticipated that future versions of the Interrogation Module will be optimized and a bit map will be introduced into the system.

The change and delete options of the Update Module are also operational. The search routines currently employed in the Update Module are the same routines found in the Interrogation Module. With a continuing effort, the attainment of our design goals and the completion of our research data base management system should be realized. Further progress reports will be issued as our development efforts are continued.

CONCLUSION

Preliminary results indicate that an AP working in conjunction with a sequential computer affords the best configuration. With this marriage comes the best of two computer worlds, each performing what it is best capable of doing. With the ability to rapidly search the entire data base we have provided the user extreme flexibility in constructuring his search criteria. We have provided this with no additional storage for inverted files and no sacrifice in update time. Associative processing brings to data base management designers and users the ability to query and update data bases in a fast and efficient manner with a minimum amount of software.

With proper programming of AP's, multi-queries can be processed during a single array load, thus greatly increasing the throughput of the system. The future holds great promise for associative processors and we are striving to lead the way. Many questions must be answered such as:

(1) how to use conventional storage devices in combination with PHD's,
(2) what hardware requirements must be provided for a minimum configured AP,
(3) the role of the sequential and AP computers in a hybrid configuration, and
(4) how much software is required to provide a data management capability on AP's.

In the near future these problems and others will have answers. Subsequent reports will elaborate on the performance of the system. Our work to date has shown that associative processors truly provide an attractive alternative to conventional approaches to data retrieval and manipulation.

ACKNOWLEDGMENTS

The help of P. A. Gilmore, E. Lacy, C. Bruno, J. Ernst and others at Goodyear Aerospace is gratefully acknowledged.

REFERENCES

1. DeFiore, C. R., Stillman, N. J., Berra, P. B., "Associative Techniques in the Solution of Data Management Problems" *Proceedings of ACM* pp. 28-36, 1971.

Aircraft conflict detection in an associative processor

by H. R. DOWNS

Systems Control, Inc.
Palo Alto, California

PROBLEM DESCRIPTION

A major problem in Air Traffic Control (ATC) is detecting when two aircraft are on a potential collision course soon enough to take some corrective action. Many algorithms are being developed which may lead to automating the process of conflict detection. However, these algorithms typically require large amounts of computing resource if they are to be performed in real-time. This paper describes some techniques which may be used by an associative processor to perform the conflict detection operation.

Conflict detection

In a terminal ATC environment, each aircraft in the area will be tracked by a computer which contains a state estimate (position and velocity) for each aircraft within the 'field of view' of the radar or beacon. The information available on each aircraft may vary (e.g., elevation may be available for beacon-equipped aircraft) but some form of estimate will be kept for each aircraft under consideration. The conflict detection algorithm typically considers each pair of aircraft and uses their state estimates to determine whether a conflict may occur. The algorithm may perform one or more coarse screening processes to restrict the set of pairs of aircraft considered to those pairs where the aircraft are 'near' each other and then perform a more precise computation to determine whether a conflict is likely.

The algorithm is typically concerned with some rather short interval of time in the future (about one minute) and it must allow for various uncertainties such as the state estimate uncertainty and the possible aircraft maneuvers during this time interval.

As an example of the kind of algorithms being considered for detection of conflicts between a pair of aircraft, a simple algorithm will be described. The relative distance at the initial time, D_R, is computed. Then, the scalar quantity relative speed, S_R, is computed for the initial time. If T is the look-ahead time, then the estimated miss distance MD is

$$MD = D_R - TS_R$$

If MD is less than a specified criterion the aircraft are assumed to be in danger of conflict.

More complicated algorithms are also under study. These involve altitude, turning rates, etc.

Associative processors

An associative processor is a processor which may access its memory by 'content' rather than by address. That is, a 'key' register containing some specific set of bits is compared with a field in each word of memory and when a match occurs, the memory word is 'accessed' (or flagged for later use). This type of memory is typically implemented by having some logic in each memory word which performs a bit-serial comparison of the 'key' with the selected field.[1]

Many associative processors have enough logic associated with each memory word to perform inequality compares (greater or less than) and some arithmetic operations. In others, memory words may be turned on or off; that is, some words may not be active during a particular comparison. This associative processor may be viewed as a parallel-array computer where each word of memory is a processing element with its own memory and the 'key register' is contained in a control unit which decodes instructions and passes them to the processing elements. Some examples of this type of computer are the Sander's OMEN,[2] the Goodyear Aerospace STARAN,[3] and the Texas Instruments SIMDA.

Some additional capabilities which these computers may have are: (1) the ability to compare operands with a different key in each processing element (both compare operands are stored in one 'word') and (2) some form of direct communication between processing elements. Typically, each processing element can simultaneously pass data to its nearest neighbor. More complex permutations of data between processing elements are also possible.

These processors are often referred to as associative array processors, though parallel-array is perhaps more descriptive.

SOLUTION APPROACHES

This section presents several ways to solve the conflict detection problem with the use of an associative processor

and provides some estimate of the *order* of each approach. (The *order* is a measure of how the computation time increases as the number of aircraft increases.)

Straightforward associative processing approach

The data for each aircraft under consideration can be placed in a separate processing element of the associative processor and each aircraft can be simultaneously compared with the remaining aircraft. This technique requires each aircraft to be compared with all other aircraft.

If there are n aircraft under consideration there must be at least n processing elements (or associative memory words). The data on a single aircraft is placed in the key register and compared with the appropriate field in each processing element. Each compare operation in this algorithm is actually fairly complex, involving some arithmetic computations before the comparison can be made.

This approach compares every aircraft with every other aircraft (actually twice) and appears to use the computing resources of the associative array rather inefficiently. In fact, the entire algorithm must be executed n times. Since n comparisons are made each time the algorithm is executed, a total of n^2 comparisons are made. (There are $n(n-1)/2$ unique paris of aircraft to be considered.)

It is more efficient to consider the associative processor for performing some sort of coarse screening procedure, and for all pairs which the AP determines may conflict to be passed to a sequential processor for a more detailed check. This assumes that the sequential processor can execute the complicated algorithm more quickly than a single processing element of the AP and that the AP can significantly reduce the number of pairs under consideration.

An efficient algorithm for a serial processor

One approach which has been suggested for solving this problem is to divide the area under consideration into boxes and to determine whether two aircraft are in the same or neighboring boxes. If they are then that pair of aircraft is checked more carefully for a possible conflict. This process is an efficient method for decreasing the number of pairs of aircraft to be checked in detail for a possible conflict. These boxes help to 'sort out' the aircraft according to their positions in the sky.

The boxes can be described in a horizontal plane and may have dimensions of 1 or 2 miles on a side. An aircraft can be assigned to a box by determining a median position estimate for some time and choosing the box which contains this estimate. If the aircraft turns or changes velocity, then it may not arrive at the expected point at the time indicated but it cannot be very far from this point. For terminal area speeds and turning rates, an aircraft will typically not be more than about one box width away from the expected location.

To check for conflicts, it is necessary to check possible conflicts in the same box and in boxes which are adjacent or near to the given box. If all aircraft in each box are compared with all other aircraft in this box and with all aircraft in boxes to the north, east and northeast, then all possible pairs are checked. By symmetry, it is not necessary to check boxes to the west, south, etc.

The exact size of the boxes and the number of boxes checked for possible conflicts are determined by the aircraft speeds and measurement uncertainties. It is desirable to make them large enough so that only neighbors one or two boxes away need to be checked and yet small enough so that the number of aircraft per box is usually one or zero.

Note, that this scheme does not check the altitude. If aircraft densities increased sufficiently and the altitude were generally available, then it might be desirable to form 3 dimensional boxes, accounting for altitude.

Sort boxes on an associative processor

A similar operation can be performed on an associative processor by assigning each box to a single processing element. If the boxes are numbered consecutively in rows and consecutive boxes are assigned to consecutive processing elements, then adjacent boxes can be compared by translating the contents of all processing elements by an appropriate amount. For a square area containing 32×32 ($=1024$) boxes, the numbering scheme shown in Figure 1 will suffice.

In this numbering scheme, all aircraft in Box 1 are compared with the aircraft in Box 2, Box 33 and Box 34. Similarly, all aircraft in Box 2 are compared with the aircraft in Box 3, Box 34, and Box 35. When these comparisons have been completed, all possible pairs within a box and in neighboring boxes will be detected.

If only one aircraft were in each box, then each aircraft data set would be stored in the appropriate processing element and comparisons with neighboring boxes would be carried out by transferring a copy of each data set along the arrows shown in Figure 2 and comparing the original data set with the copy. When these neighboring boxes have been checked, then all possible conflicting pairs have been discovered.

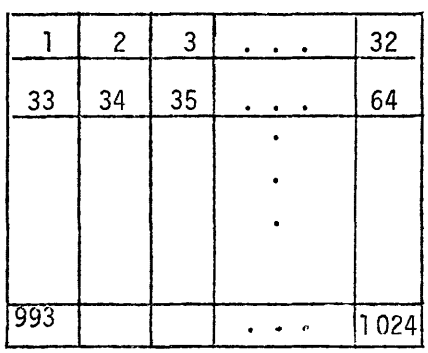


Figure 1—Boxes in 32×32 surveillance area

When there is more than one aircraft in a box, then additional processing is required. The first aircraft which belongs in a box is stored in the proper processing element and a flag (bit) is set indicating this is its proper box. Any additional aircraft are placed in any available processing element. A pointer is set up linking all aircraft belonging in a specific box. In an associative processor, all that is necessary is to store the I.D. of the next aircraft in the chain in a particular processing element. The contents of various fields of a word and the links to the next word are shown in Figure 3.

When the data is stored as indicated above, then every aircraft in box i can be retrieved by starting with processing element i. If the I.D. in Field B is not zero, then the next aircraft data set is located by performing an equality test using Field B from processing element i in the key register and comparing with Field A in all processing elements.

When comparing the aircraft in one box with those in another box, every aircraft in the first box must be compared with every aircraft in the other box. This is done by making a copy of the state estimates for each aircraft pair and placing each pair of state estimates in a single processing element. When the conflict detection algorithm is executed, all the pairs are checked simultaneously. That is, each processing element operates on a pair of state estimates and determines, by a fairly complicated algorithm, whether the aircraft pair represented in this processing element may be in danger of colliding.

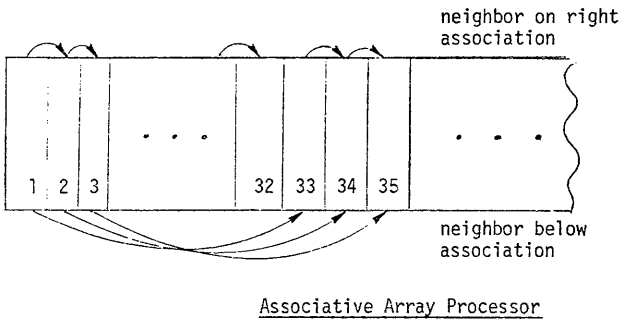


Figure 2—Data transfer for comparison with neighboring boxes

If the memory available in each processing element pair is sufficient, then one pair can be stored in each element. In this case, the array is used very efficiently since many elements can execute the detailed conflict detection algorithm simultaneously. If there are more processing elements than pairs, then the detailed conflict detection algorithm need only be executed once in order to check all pairs for possible conflict.

If there are no processing elements available for storing a potentially conflicting pair, then the algorithm is executed on the pairs obtained so far, non-conflicting pairs are deleted and conflicting aircraft are flagged. The process then continues looking for more conflicting pairs.

The algorithm just described is similar to the one described for sequential computers but takes advantage of the parallel and associative processing capabilities of an associative array computer to speed up the execution of the algorithm. If the associative processor has more elements than there are boxes and than there are pairs to check, then the algorithm requires essentially one pass.

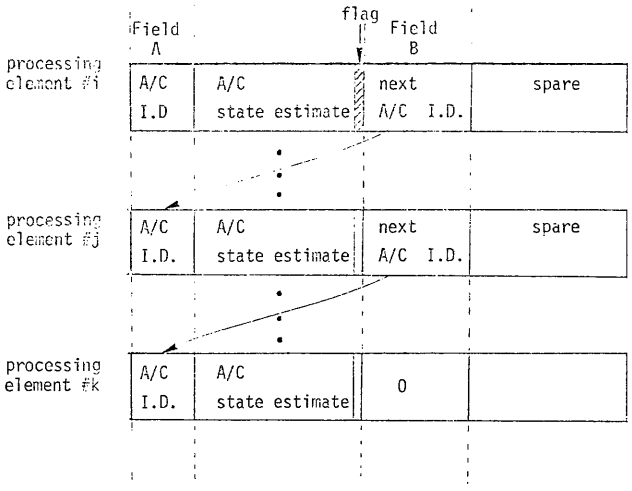


Figure 3—Storage of A/C state estimates for A/C in Box #i

A sliding correlation algorithm

Another approach to restricting the number of pairs of aircraft is possible on an associative array processor. This approach requires that the aircraft be arranged in order of increasing (or decreasing) range in the processing elements. That is, the range from the radar to the aircraft in processing element i is less than the range to the aircraft in processing element $i+1$. A number of techniques are available for performing this sort. They are discussed in the next section.

The technique consists of copying the state estimates and I.D.'s of each of the aircraft and simultaneously passing them to the next adjacent processing element. Each element checks the two aircraft stored there for a possible conflict and flags each of them if a conflict occurs. The copied state estimates are then passed to the next processing element and another conflict check occurs. The process continues until all aircraft pairs in a given processing element are more than r nautical miles apart in range (where r is the maximum separation of two

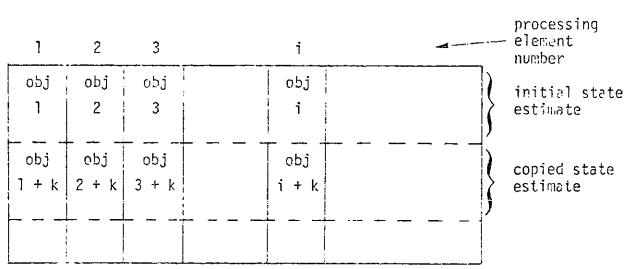


Figure 4—Storage during the kth conflict comparison

aircraft state estimates when a conflict might conceivably occur during the specified time interval). Since the aircraft are range ordered, there is no point in testing any additional pairs. Also, due to the symmetry of the algorithm, it is not necessary to check the other way.

Ordering the state estimates

In the algorithms described earlier, a certain ordering of the aircraft state estimates in the processing elements was required.

The ordering can be performed by checking the arrangement of the data each time a track update is performed. Each aircraft which has moved to a new box has its corresponding state estimate moved to the appropriate processing element at this time. It is assumed that the track update rate is sufficiently high so that state estimates need only move to neighboring boxes. This assumption improves the efficiency with which data is moved since many state estimates can be moved simultaneously. For instance, all objects which moved one box to the east can be simultaneously transferred to their new box. Similarly, for objects moving in other directions, a single transfer is necessary.

The second, third, etc., aircraft in a box must be handled separately. Also, when a box is already occupied, then special handling is required to determine where the new state estimates are stored. Since most boxes contain only one or fewer aircraft, the amount of special handling will be small.

The sorted list described in the previous section may also be done by keeping the list stored and performing minor iterations each time the state estimates are updated. This process is quite straightforward.

A more interesting method is to perform a total sort of the state estimates. This can be done fairly efficiently by using a 'perfect shuffle' permutation of the processing element data. The algorithm is described in Stone[4] and requires $\log_2 n$ steps (where n is the number of aircraft).

SUMMARY

Comparison of techniques

The algorithm in the section "Straightforward Associative Processing Approach" is quite inefficient and requires n passes through the pairwise conflict detection algorithm. This can be quite expensive if n is large and the algorithm is complex.

The algorithm in the section "Sort Boxes on an Associative Processor" greatly reduces the number of executions of the pairwise conflict detection algorithm at the expense of some data management overhead. If the box sizes can be chosen appropriately this is quite efficient. Some studies have shown that a few percent of the pairs need to be checked. That is, the number of potentially conflicting pairs after checking for box matching is $p \cdot n(n-1)/2$ where p is about .05. If a total of four boxes must be checked for conflicts (nearest neighbor boxes only), then the number of times the pairwise conflict detection algorithm must be executed is quite small. There are about $n^2/40$ potentially conflicting pairs and if the number of processing elements is large enough, these can all be checked at one time.

The algorithm described in the preceding section has been estimated to check about 10 percent of the potentially conflicting pairs. Comparing with the section "Straightforward Associative Processing Approach," this requires one-tenth the time or about $n/10$ pairwise conflict detection executions (plus the sort). This is more than the sliding correlation algorithm previously discussed, but the data management is less and fewer processing elements are required. Depending on the execution time of the pairwise conflict detection algorithm one of these approaches could be chosen, the first one if the pairwise algorithm is expensive and the second if the pairwise algorithm is cheap.

Conclusions

This paper has presented some approaches to organizing the conflict detection algorithm for Air Traffic Control on an associative array processor. The relative efficiencies of these approaches were described and some of the implementation problems were explored.

Other techniques of real-time data processing on a parallel-array computer have been described in Reference 5. The problem described here illustrates the fact that developing efficient algorithms for novel computer architectures is difficult and that straightforward techniques are often not efficient, especially when compared with the sophisticated techniques which have been developed for serial computers. The techniques described here may also be applicable to other problems for associative arrays.

REFERENCES

1. Stone, H., "A Logic-in-Memory Computer," *IEEE Transactions on Computers*, January 1970.
2. Higbie, L., "The OMEN Computers: Associative Array Processors," *Proceedings of COMPCON*, 1972, page 287.
3. Rudolph, J. A., "A Production Implementation of an Associative Array Processor—STARAN," 1972, *FJCC Proceedings*, page 229.
4. Stone, H., "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers*, February 1971.
5. Downs, H. R., "Real-Time Algorithms and Data Management on ILLIAC IV," *Proceedings of COMPCON*, 1972.

A data management system utilizing an associative memory*

by CASPER R. DEFIORE**

Rome Air Development Center (ISIS)
Rome, New York

and

P. BRUCE BERRA

Syracuse University
Syracuse, New York

INTRODUCTION

There are a wide variety of data management systems in existence.[1-3,6,7,10-15] These systems vary from those that are fairly general to those that are very specific in their performance characteristics. The former systems tend to have a longer life cycle, while sacrificing some efficiency, whereas the latter are more efficient but tend to become obsolete when requirements are modified.[9]

In addition, current data management systems have generally been implemented on computers with random access memories, that is, the data is stored at specified locations and the processing is address oriented. In contrast, using associative or content addressable memories, information stored at unknown locations is processed on the basis of some knowledge of its content. Since much of the processing of data management problems involve the manipulation of data by content rather than physical location, it appears that associative memories may be useful for the solution of these problems.

In order to demonstrate the feasibility of utilizing a hardware associative memory, a data management system called Information Systems For Associative Memories (IFAM) has been developed and implemented. After presenting a brief description of associative memories, the implementation and capabilities of IFAM are described.

DESCRIPTION OF ASSOCIATIVE MEMORIES

Associative memories differ considerably from random access memories.[16] Random access memories are address oriented and information is stored at known memory addresses. In associative memories, information stored at unknown locations is retrieved on the basis of some knowledge of its content. The information stored at unknown locations can be retrieved on the basis of some knowledge of its content by supplying the contents of any portion of the word.

An associative memory contains a response store associated with every word which is at least one bit wide. Its purpose is to hold the state of events in the memory. The response store, provides an easy way of performing boolean operations such as logical AND and OR between searches. In the case of logical AND for example, in a subsequent search only those words whose response store were set would take part in the search. In addition boolean operations between fields of the same word can be performed by a single search. The applicability of boolean operations is a most important requirement for a data management system since the conditional search for information in a data base requires their use.

The instruction capabilities of associative memories are usually grouped into two categories: search instructions and arithmetic functions. The search instructions allow simultaneous comparison of any number of words in the memory and upon any field within a word. A partial list of search instructions include the following: equality, inequality, maximum, minimum, greater than, greater than or equal, less than, less than or equal, between limits, next higher, and next lower. All of these instructions are extremely useful for data management applications. For example, the extreme determination (maximum/minimum) is useful in the ordered retrieval for report generation. An associative memory can perform mass arithmetic operations, such as adding a constant to specific fields in a file. The type of operations are as follows: add, subtract, multiply, divide, increment field and decrement field.

IFAM IMPLEMENTATION

IFAM is an on-line data management system allowing users to perform data file establishment, maintenance,

* This research partially supported by RADC contract AF 30(602)-70-C-0190, Large Scale Information Systems.

** Present Address, Headquarters, DCA, Code 950, NSB, Washington, D.C.

retrieval and presentation operations. Using this system it is easy for users to define a meaningful data base and rapidly achieve operational capability. Discussed below are the hardware and software aspects of IFAM.

Hardware

IFAM is implemented on an experimental model associative memory (AM) at Rome Air Development Center (RADC). The AM, developed by Goodyear, is a content addressable or parallel search memory with no arithmetic capability. It contains 2048 words where each word is 48 bits in length. The search capability consists of the 11 basic searches described previously, all performed word parallel bit-serial. As an example of the timing, an exact match search on 2048-48 bit words takes about 70 microseconds.

The AM operates in conjunction with the CDC 1604 Computer via the direct memory access channel as shown in Figure 1. Information transfers between the AM and the 1604 are performed one word at a time at about 12 microseconds per word.

The CDC 1604 computer is a second generation computer with 32000-48 bit words and a cycle time of 6.4 microseconds. It has various peripheral and input/output devices, one of which is a Bunker Ramo (BR-85) display console. The display unit is capable of visually presenting a full range of alphabetic, numerical or graphical data. The console allows direct communication between the operator and the computer. The display unit contains a program keyboard, an alphanumeric keyboard, a control keyboard, curser control. and a light gun. The IFAM user performs his tasks on-line via the BR-85 display console.

Software

Both the operational programs and the data descriptions used in IFAM are written in JOVIAL, the Air Force standard language for command and control.

The operational programs manipulate the data in the AM and are utilized in the performance of retrieval and update operations. These programs are written as JOVIAL procedures with generalized input/output parameters which operate on data in the AM. By changing the parameters, the same procedures can be used for many different operations resulting in a more general and flexible data management system.

The data descriptions are used to specify the format of the data such as the name of a domain, its size, type, value, the relation to which it belongs, etc. They are used in conjunction with the operational programs to process information within IFAM. The data descriptions specify the data format to the operational program which then performs the desired task. Providing an independence between these two means the information and the information format can change without affecting the programs that operate on the information and conversely.

In addition, the data and data descriptions are maintained separately from the data. This separation means that multiple descriptions of the same data are permitted and so the same data can be referred to in different ways. This can be useful when referencing data for different applications.

The next section describes the capabilities of IFAM using a personnel data base as a test model.

IFAM CAPABILITIES

Test Model Description

As a test model for IFAM a data base has been implemented containing information about personnel. The data structure for the test model, shown in Figure 2, is as follows:

PERSONNEL (NAME, SOCIAL SECURITY NUMBER, CATEGORY, GRADE, SERVICE DATE, DEGREES)
DEGREES (DEGREE, DATE).

For this structure PERSONNEL is a relation that contains six domains and DEGREES is a relation that contains two domains. The occurrence of the domain degrees

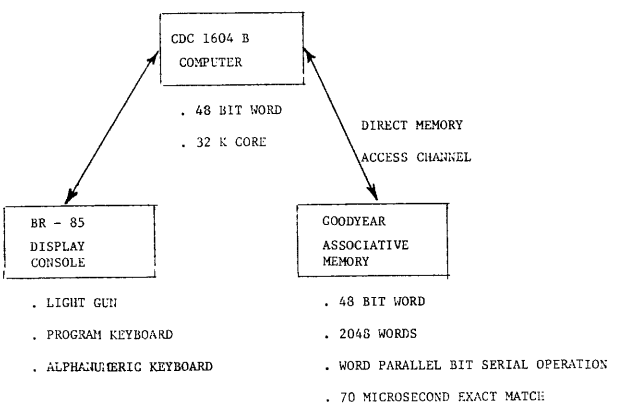


Figure 1—RADC associative memory computer system

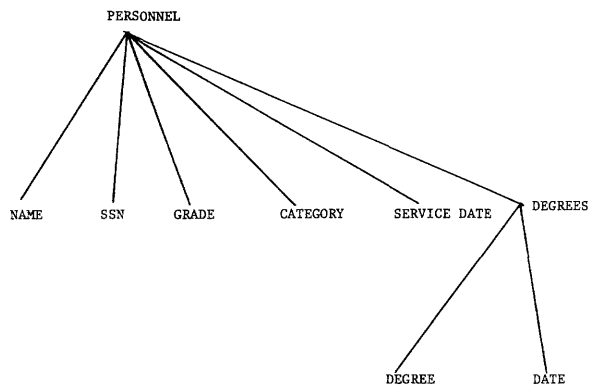


Figure 2—Hierarchical structure for data base

in the PERSONNEL relation is a non-simple domain since it contains two other domains namely degree and date. All other domains are called simple domains since they are single valued. A method and motivation for eliminating the non-simple domain is given in Reference 4. Essentially the procedure eliminates each non-simple domain by inserting a simple domain in its place and inserting this same simple domain in all subordinate levels of the structure. Through the application of this procedure, the data structures now contain only simple domains and therefore are amenable to manipulation on an associative memory.

When this procedure is applied to the above data structure, the result is as follows:

PERSONNEL (NAME, SOCIAL SECURITY NUMBER, CATEGORY, GRADE, SERVICE DATE, α_1)
DEGREES (DEGREE, DATE, α_1),

where α_1 is the added simple domain.

The storage structure showing one n-tuple of the personnel relation and one n-tuple of the degrees relation appears in Figure 3. Five associative memory (AM) words are needed to contain each n-tuple of the personnel relation and two AM words are needed to contain each n-tuple of the degrees relation.

Since the capacity of the Goodyear AM is 2000 words, any combination of n-tuples for the two relations not exceeding 200 words can be in the AM at any one time.

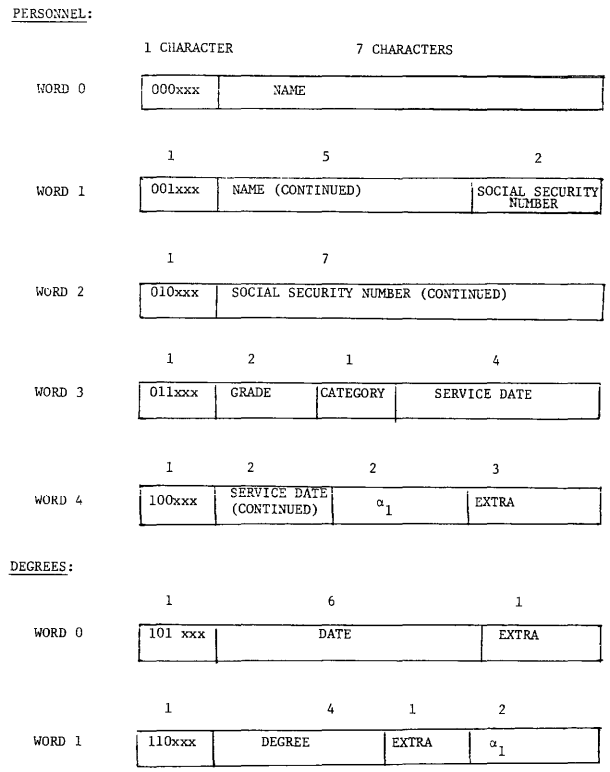


Figure 3—Data base storage structure

That is, at any one time the AM could contain a combination of n-tuples such as the following: 400 personnel and no degrees, 1000 degrees and no personnel, 200 personnel and 500 degrees, etc. Whenever the number of n-tuples exceeds the capacity of the AM, additional loading is required.

The first three bits of each word are the word identification number and are used to determine which group of AM words participate in a search. For example, consider a case where both the personnel and degrees relations are in the AM, and suppose it is required to perform an exact match on the first two digits of social security number (see Figure 3). In this case only word one of each personnel n-tuple participates in the search. In order to insure that this is so, a 001 is placed in the first three bits of the comparand register. This corresponds to words in the personnel relation that contain the first two digits of social security number. The remainder of the comparand contains the search argument, which in this case is the desired social security number. This method assures that only the proper AM words participate in a search. Also, as shown in Figure 3, word 4 character 4 of the PERSONNEL relation and word 1 character 7 of the DEGREES relation contain α_3 domains.

Query and update method

A dialog technique exists within IFAM in which an inquirer interacts with a sequence of displays from the BR-85 to accomplish the desired task. In this way the user does not have to learn a specialized query language thereby making it easier for him to achieve operational capability with a data base.

The first display appears as shown in Figure 4. Assuming Button 1 is selected, the display appears as shown in Figure 5. Observe that although the degrees relation is embedded in the personnel relation, a retrieval can be performed on either relation directly (i.e., one does not have to access the personnel relation in order to get to the degrees relation). Assuming button 2 (NAME) is selected, the next display appears as in Figure 6, in which the 11 basic searches available in the system are shown. Assum-

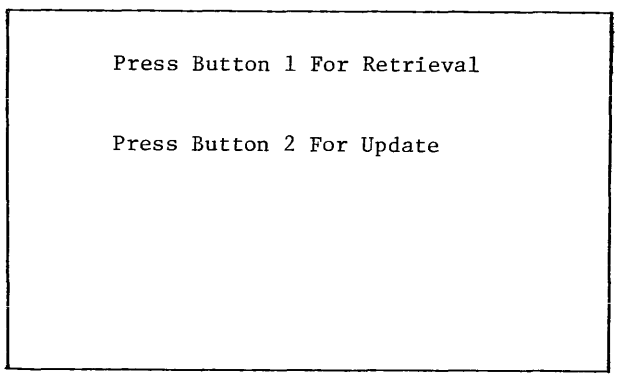


Figure 4—First display

```
(1)   Personnel

      (2)  Name
      (3)  Social Security Number
      (4)  Grade
      (5)  Category
      (6)  Service Date

(10)  Degrees

      (11)  Degree
      (12)  Date

Press button of item you wish to specify
(only one button may be pressed)

Press button 20 to terminate
```

Figure 5—Second display

ing button 2 (EQUAL) is chosen, the next display appears as shown in Figure 7. This display is used to specify the search arguments and the AND or OR operation for conditional searches. Assuming that an "E" is typed in the second position, then all personnel with names which have E as the second letter will be retrieved from the system and the next display appears as shown in Figure 8. This display gives the inquirer the option of terminating his request, having been given the number of responders, or having the results displayed. Assuming it is desired to display the results, button 1 is selected and the next display appears as shown in Figure 9.

In Figure 9 the n-tuples of the personnel and degrees relations satisfying the request are displayed. Observe that each name has an E as the second letter and that all the degrees a person has are displayed (e.g., Mr. Feate has 3 degrees). At the completion of this display the system recycles and begins again with the first display as shown in Figure 4.

```
You have chosen
    (2)  Name

The operators available are:

 1.  Between Limits
 2.  Equal
 3.  Greater than or equal
 4.  Greater than
 5.  Less than or equal
 6.  Less than
 7.  Maximum value
 8.  Minimum value
 9.  Not equal
10.  Next higher
11.  Next lower

Press button of operation you wish to specify

Press button 20 to terminate
```

Figure 6—Third display

```
you have chosen

  Name     Equal

Specify values below if required


(_E_ _ _ _ _ _ _ _ _ _)

(_ _ _ _ _ _ _ _ _ _ _ _)

Input xxx to terminate

Also specify AND, OR operation from previous search

Blanks specify neither (_ _ _)
```

Figure 7—Fourth display

Using IFAM it is easy to perform operations involving the union or intersection of data items. For example, consider ANDing together SSN, Name and Degree such that the 2nd, 4th, 5th, and 9th digits of the SSN are equal to 3, 9, 0, 5 respectively, the second letter of the name is equal to K, and the degree begins with B, as shown in Figure 10. This is performed as a routine request in IFAM in the following way. In one access those records satisfying the specified SSN are found by placing the desired SSN in the comparand register and the fields to be searched in the mask register. Of those records, only the ones in which the second letter of the name is equal to K are found in one memory access using the AND connector between searches. For those records the α_1 values are used to set the corresponding values in the degree file ANDed together with the records in which the degree begins with B. The records in this final set satisfy this complex request. Other systems either have to anticipate such a request and provide the necessary programming and indexing to accomplish it, or would have to perform a sequential search of the data base.

On-line updating is provided in IFAM and operates in conjunction with retrieval. Retrieval is used to select the portion of the data to be altered. Once selected the items are displayed on the BR-85 and any of the domains of a relation including the α s can be changed and put back into the data base. Using IFAM, updates are performed in a straightforward manner with no directories, pointers, addresses or indices to change.

```
There are 4 responders to your query


Do you wish to display them?

If Yes press Button 1

If No press Button 2
```

Figure 8—Fifth display

NAME	SSN	CATEGORY	GRADE	SERVICE DATE	DEGREE	DATE
Berg	031714132	C	13	570725	BS	49
					MS	51
Teaomes	032513165	C	12	621113	BS	55
Feate	425626582	C	11	510820	BS	55
					MS	58
					Ph.D.	62
Reed	057254321	C	10	480530	BS	47

Figure 9—Sixth display

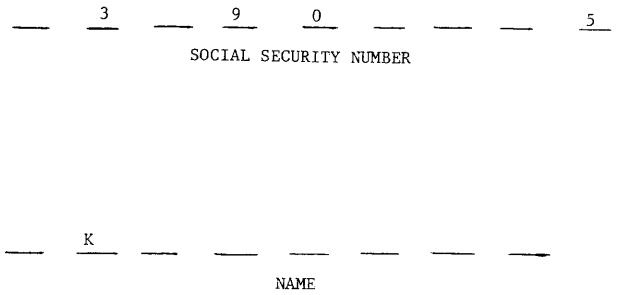


Figure 10—Sample retrieval request

CONCLUSION

This paper has described the utilization of an associative schemata for the solution of data management problems. There are advantages and disadvantages in utilizing associative memories for data management. Among the advantages shown by this implementation are that this method superimposes little additional structure for machine representation and eliminates the need for indexing. From an informational standpoint, an index is a redundant component of data representation. In the associative method, all of the advantages of indexing are present with little data redundancy. A comparison between IFAM and an inverted list data management system given in Reference 4 shows that the inverted list technique requires 3 to 15 times more storage.

Also provided in Reference 4 is a comparison between IFAM and inverted lists in the area of query response time. It is shown that queries using inverted lists take as much as 10 times longer than IFAM. In the inverted list technique, as more items are indexed response time to queries tend to decrease, whereas update time normally increases. This is so because directories and lists must be updated in addition to the actual information. Since IFAM does not require such directories, the overhead is kept at a minimum and updating can be accomplished as rapidly as queries. In addition, the associative approach provides a great deal of flexibility to data management, allowing classes of queries and updates to adapt easily to changing requirements. Thus, this system is not likely to become obsolete.

One of the disadvantages of associative memories is the cost of the hardware. The increased cost is a result of more complex logic compared to conventional memories. This cost is justified in instances such as those described above. On the other hand, if factors such as speed and flexibility are not important, then the less costly current systems are preferable.

REFERENCES

1. Bleier, R., Vorhaus, H., "File Organization in the SDC Time Shared Data Management System (TDMS)", *IFIP Congress*, Edinburg, Scotland, August, 1968.
2. CODASYL System Committee Technical Report, "Feature Analysis of Generalized Data Base Management Systems", *ACM Publication*, May, 1971.
3. Dodd, G., "Elements of Data Management Systems", *Computer Surveys*, June, 1969.
4. DeFiore, C., "An Associative Approach to Data Management", *Ph.D. Dissertation*, Syracuse University, Syracuse, New York, May, 1972.
5. DeFiore, C., Stillman, N., Berra, P. Bruce, "Associative Techniques In The Solution of Data Management Problems", *Proc. of the 1971 ACM National Conference*, 1971.
6. "GE 625/635 Integrated Data Store", *GE Reference Manual*, August, 1965.
7. "Generalized Information Management (GIM), Users Manual", *TRW Doc. #3181-C*, 15 August, 1969.
8. Minker, J., "An overview of Associative or Content Addressable Memory Systems And A KWIC Index to the Literature: 1956-1970", *Computing Reviews*, Vol. 12, No. 10, October, 1971.
9. Minker, J., "Generalized Data Management Systems - Some Perspectives", *University of Maryland*, TR 69-101, December, 1969.
10. Prywes, N., Gray, H., "The Multi-List System for Real Time Storage and Retrieval", *Information Processing*, 1962.
11. Prywes, N., Lanauer, W., et al., "The Multi-List System - Part I, The Associative Memory", *Technical Report I, AD 270-573*, November, 1961.
12. Sable, J., et al., "Design of Reliability Central Data Management System", *RADC TR 65-189*, July, 1965.
13. Sable, J., et al., "Reliability Central Automatic Data Processing System", *RADC TR 66-474*, August, 1966.
14. *SDC TM-RF-104/000/02*, "Management Data Processing Systems", (Users Manual), 20 June, 1969.
15. *SDC TM-RF-10/000/01*, "Management Data Processing System", (On-Line File Manipulation Techniques), 20 June, 1969.
16. Wolinsky, A., "Principles and Applications of Associative Memories", *TRW Report 5322.01-27*, 31 January, 1969.

Associative processor applications to real-time data management

by RICHARD R. LINDE, ROY GATES, and TE-FU PENG

System Development Corporation
Santa Monica, California

INTRODUCTION

This paper describes a research study concerning the potential of associative processing as a solution to the problem of real-time data management.[1] The desired outcome of the research was an evaluation of the comparative advantages of associative processing over conventional sequential processing as applied to general real-time Data Management System (DMS) problems. The specific DMS application framework within which the study was carried out was that of the data management functions of the U.S. Air Force Tactical Air Control Center (TACC).

The primary feature used to select an associative processor (AP) configuration was "processing efficiency," by which is meant the number of successful computations that can be performed in a given time for a certain cost. In DMS applications, processing efficiency is influenced by large-capacity storage at low cost per bit, the record format, search speed, and the ability to search under logical conditions and to combine search results in Boolean fashion. The primary technique used for comparing an AP's performance with that of a sequential processor was to arrive at a set of *actual* execution rates for the class of real-time data management problems. These execution rates, coupled with the assumption that parallel computers cost four or five times more than their equivalent sequential computer counterparts, will yield an estimate of an AP's cost-effectiveness for the DMS problem. Obviously, the more data that can be processed in parallel, the greater the processing efficiency; and DMS applications, by their nature, have a high degree of parallelism and, hence, they dictated the type of parallel processor used for the comparisons. As a result, we dealt with machines that fit into the general category of the associative processors.[2] These are machines that execute single instruction streams on multiple data paths; or, viewed another way, they are machines that can execute a program on a (large) set of data in parallel. The classical array processors, such as the ILLIAC-IV and PEPE machines, are characterized by a relatively small number of sophisticated processing elements (e.g., a network of mini-computers), each containing several thousand bits of storage.[3] In order to obtain a high processing efficiency for data management applications, we have described a machine that has many more processing elements than the classical array processors and whose processing elements are smaller (256 bits) and, hence, provide more distributed logic. For cost reasons, our machine cannot be fully parallel, having logic at every bit; for efficiency reasons, however, we selected a byte-serial, external-byte-logic machine design as opposed to the bit-serial processors that are widely described in the literature. For example, three cycles are required for this type of machine to perform an 8-bit add, whereas 24 cycles are required for the bit-serial, external-logic machines, yet the byte-serial logic cost is very little more than that of the equivalent bit-serial logic.

Past studies have shown that data management systems implemented on an AP can become I/O-bound quite quickly due to the AP's microsecond search operations relative to slow, conventional, I/O-channel transfer rates.[4] Hence, for our associative memory (AM), we have hypothesized a large, random-access data memory for swapping files to and from the AMs at a rate of 1.6 billion bytes/sec. Other studies have considered the use of associative logic within high-capacity storage devices (logic-per-track systems) such as fixed-head-per-track disc devices.[2,5]

Thus, for reasons of efficiency, at a reasonable cost, we have described a byte-serial, word-parallel machine, called the Associative Processor Computer System (APCS), that consists of two associative processing units, each having 2048 256-bit parallel-processing elements. This machine is linked to a conventional processor that is comparable to an IBM 370/145 computer. In order to compare this machine with a sequential machine, we have programmed several real-time problems for both machines and compared the resultant performance data. The set of problems for the study were derived in part from an analysis of the TACC testbed developed at Hanscom Field, Mass. The TACC is the focal point of all air activity within the Tactical Air Control System (TACS). The TACS is a lightweight, mobile surveillance and detection system, assigned to a combat area at the disposal of Air Force Commanders for making real-time air support and air defense oriented decisions.

The TACC is divided into two divisions: Current Plans and Current Operations. Current Plans is responsible for developing a 24-hour fragmentary (FRAG) order that defines the air activities within the combat area during the next 24-hour period, and Current Operations monitors those activities in real time. Some of the Current Operations functions were programmed for the IBM 1800 computer at the testbed, and an analysis was made of a scenario, relating to a testbed demonstration, for determining those functions critical to Current Operations and, therefore, important candidates for our study.

Also, a study was made of several data management systems (SDC's DS/2, CDMS, and CONVERSE, and MITRE'S AESOP-B) to arrive at a set of associative processor DMS primitives.[6,7,8,9] Those primitives that appeared to be most important to DMS functions were evaluated. We used the primitives required by update and retrieval applications for writing APCS programs, and compared the APCS performance results against conventional programs, written for data bases stored physically in random-access fashion.

A relational data-structure model provided us with a mathematical means of describing and algorithmically manipulating data for an associative processor. Because relational data structures are not encumbered with the physical properties found in network or graph-oriented models, they provide independence between programs and data. The TACC testbed data structure provided us with examples for describing and logically manipulating relational data on the APCS in a DMS context.

The study results enabled us to make recommendations for further studies directed toward arriving at cost-effective parallel-machine solutions for real-time DMS problems. APCS-like machine descriptions will continue to evolve for the next several years. We do not recommend that the APCS or the various future versions (which, like the APCS, will have to be fixed for comparison purposes even though their structures may not be optimally cost-effective) be considered a cost-effective solution to the DMS problem. Instead, we hope that they will serve as starting points for the machine evolution studies that will have to precede any attempt to describe the "ultimate" solution for data management.

ASSOCIATIVE PROCESSOR COMPUTER SYSTEM

A complete associative processor computer system (APCS) for a real-time data management application is shown in Figure 1. It is not a so-called "hybrid" system, with an interconnection of serial and associative processors; rather, it is a totally integrated associative computer. The central processor functions as an active resource that controls the overall system and performs information transformation. The memory system acts as a passive resource for the storage of information. Numerous data paths are provided between the central processor and various memory and external units for efficient information movement.

The central processor is an integration of five units: two associative processing units (APUs), a sequential processing unit (SPU), an input/output channel (IOC), and a microprogrammed control memory. The APUs provide a powerful parallel processing resource that makes the system significantly different from a conventional one. From a conventional system-architectural point of view, the APUs can be viewed as a processing resource that has a large block of long-word local registers (in this case, 4K × 256 bits) with data manipulation logic attached to each word; an operation is performed on all registers simultaneously, through content addressing. Actually, the central processor has three different processors—I/O, sequential, and parallel—and all three can operate simultaneously. The three processors fetch instructions from the program memory, interpret the microprograms from control memory, and manipulate the data from data memory.

The memory system consists of three units: primary memory, control memory, and secondary storage. The data management functions and the data base of the data management system are independent of each other; for this reason, the primary memory is subdivided into program memory and data memory for faster information movement and for easy control and protection. This separation is another special feature of the system. The program memory is assumed to be large enough to store executive routines, data management functions, and user programs for the three processing units; consequently, program swap operations are minimized. The data memory can store the data base directory and several current active data files.

Finally, two additional resources, terminals and displays and other peripherals, are added to make up a complete real-time system. The other peripherals include tape units and unit-record devices such as line printers, card readers, and punches.

The reading and writing of AP words are controlled by tag vectors. The word logic is external to each AM word for actual data operations and control. The word logic

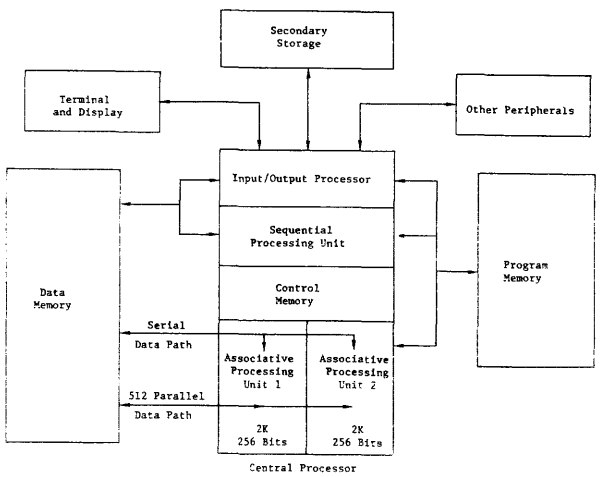


Figure 1—Associative processor computer system (APCS)

consists of two main portions: byte operation logic and tag vectors. The byte operation logic includes byte adders, comparison logic, and data path control logic. There are three types of tag vectors: hardware tag vectors, tag vector buffers, and tag vector slices. The hardware tag vectors consist of four hardware control tag columns: word tag vector (TVW), control tag vector (TVC), step tag vector (TVS), and response tag vector (TVR). The TVW indicates whether a corresponding AM word is occupied; it is set as the AM is loading and reset when it is unloading. The TVC controls the actual loading and unloading of each AM word in a way analogous to that in which mask bits control the byte-slice selection, except that it operates horizontally instead of vertically. The setting of TVS indicates the first AP word of a long record (one containing more than one AP word). The use of the TVR is the same as the familiar one, where a corresponding tag bit is set if the search field of an AP word matches the key. In addition, most tag-manipulation logic (such as set, reset, count, load, store, and Boolean) is built around the TVR. The status and bit count of the TVR vector can be displayed in a 16-bit tag vector display register (TVDR) to indicate the status of various tag vectors for program interpretation. These four hardware tag vectors are mainly controlled by hardware, which sets and resets them during the actual instruction execution. In addition, they can also be controlled by software for tag vector initiation or result interpretation. The contents of various hardware tag vectors can be stored temporarily in the eight tag vector buffers if necessary. This is particularly useful for simultaneous operations on multiple related data files or for complex search operations that are used for multiple Boolean operations on the results of previous tag-vector settings. The use of the five tag vector slices is similar to that of the tag vector buffers except that they can be stored in the DM along with the corresponding AP words and loaded back again later.

A RELATIONAL DATA MODEL FOR THE TACC TESTBED

Let us now consider a data structure for the APCS in terms of the TACC testbed DMS.

Set relations provide an analytical means for viewing logical data structures on an associative processor.[10,11,12] Given sets $(S_1S_2, \cdots, S_n)$. If R is a relation on these sets, then it is a subset of the Cartesian Produce $S_1 \times S_2 \cdots S_n$, and an n-ary relation on these sets. The values of elements in this relation can be expressed in matrix form where each j^{th} column of the matrix is called the j^{th} domain of R and each row is an n-tuple of R. Domains may be simple or non-simple. A simple domain is one in which the values are atomic (single valued), whereas non-simple domains are multivalued (i.e., they contain other relations). As an example of set relations applied to associative processing, consider the example of the TACC testbed.

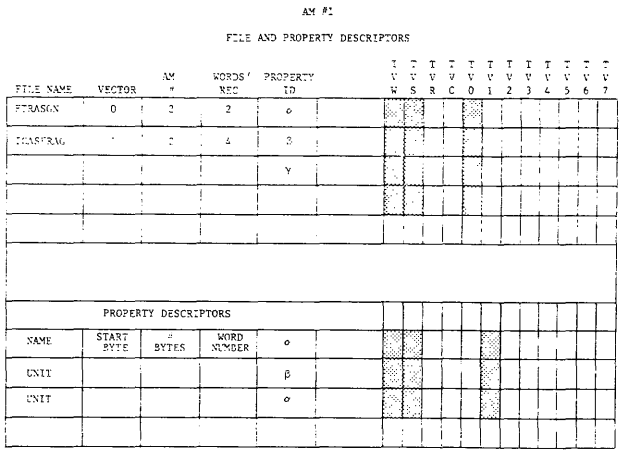


Figure 2—TACC tested storage structure on the APCS

The testbed defines a "property" as one item of information (e.g., one aircraft type for a designated base). An "object" is a collection of properties all relating to the same base, unit, etc. Objects are the basic collections of data within a file (e.g., an air base file would have one object for each designated base, and each object would contain all properties for one base). A complex property is a collection of properties (such as data and time) that are closely related and that can be accessed either individually or as a group (date/time).

The data base files are structured to show property number, property name, complex property indicator, property type, EBCDIC field size, range/values, and property descriptions. They are self-described and contain both the data (the property values) and the descriptions of the data. The descriptive information constitutes the control area of the file, and the data values constitute the object area of the file. The control area contains an index of all objects in the file (the object roll), and an index and description of all properties in the file (the property roll). The object roll contains the name of every object in the file, along with a relative physical pointer to the fixed-length-record number containing its data; the property roll defines the order of property-value data for each object and points to the location of the data within the object record.

Figure 2 illustrates how AM #1 might be used to logically represent a testbed file control area. It contains two relations:

$$R_1(a_{11}, a_{12}, a_{13}, a_{14}, a_{15})$$

and

$$R_2(a_{11}, a_{12}, a_{13}, a_{14}, a_{15})$$

which represent an index of all the files in the testbed (R_1) and a list of all the property descriptors associated with the file (R_2). The first domain (file name) of relation R_1 is the primary key; that is, it defines a unique set of elements (or n-tuples) for each row of R_1.

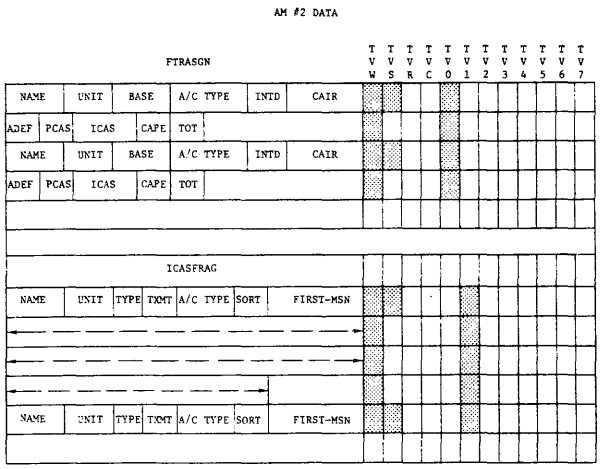


Figure 3—TACC testbed property values on the APCS

The domain labeled "Vector" contains the address of the tag vector that delineates the property values for the file; they are contained in AM #2. The domains labeled "AM #" and "Words/Record" contain the AM number for the property values and the number of physical AM words per record. The domain labeled "Property ID" contains a marking value that identifies the property descriptors in relation R_2 for each file name of primary key in relation R_1. Tag vectors TV0 and TV1 mark those words in AM #1 containing relations R_1 and R_2, respectively; this is necessary because two domains of the respective relations may be of the same size and content, thus simultaneously satisfying search criteria. The programmer must AND the appropriate vector, either TV0 or TV1, with TVR after such a search to obtain the relevant set of N-tuples.

Since the APCS uses a zero-address instruction set, parameter lists must be defined in control memory for each APCS instruction to be executed. Obviously, control-memory parameter lists are needed to define search criteria for relations R_1 and R_2 and table definitions (not shown) for creating these lists before the descriptors contained in AM #1 can be used to define another set of lists for accessing the property values in AM #2 (see Figure 3).

Figure 3 illustrates the relations that describe the Fighter Assignment File (FTRASGN) and the Immediate Close Air Support Frag Order File (ICASFRAG) property values. They are represented by n-ary relations containing a set of simple domains, 1 through n, which contain the actual data values for the elements: first mission number, last mission number, aircraft/type, etc. The domain labeled UNIT is the primary key for the relation and corresponds to the unit number contained in the testbed object roll.

If the domain representing aircraft type contained more than one aircraft and a set of simple domains describing each aircraft, then a non-simple domain would occur in a primary relation. That is, there might be F104 and F105 fighter aircraft and parameters relating to their flight characteristics (range, speed, etc.) occurring within the primary relation. To normalize this relation, the primary key or a unique marker is inserted in the non-simple domain, the non-simple domain is removed from the primary relation, and the sequence is repeated on the secondary relation, tertiary relation, etc., until all non-simple domains have been removed. DeFiore[10] has described and illustrated this technique on an AP and calls the data structure containing only simple domains the Associative Normal Form (ANF).

SEARCH AND RETRIEVAL COMPARISONS

A search and retrieval comparison was performed using the APCS and a sequential computer, the IBM 370/145, as the comparative processors. Two problems involving fact and conditional search and retrieval were coded. A random-access storage structure was chosen for the conventional case; the conventional dictionaries were unordered. The data structure for the APCS can be shown by the relation

$$R_1(a_{11}, a_{12}, a_{13}, a_{14}, a_{15})$$

$$R_1{}^1(a_1{}^1{}_6, a_1{}^1{}_7, a_1{}^1{}_8, a_1{}^1{}_9, a_1{}^1{}_{10}, a_1{}^1{}_{11}).$$

It is an 11-tuple requiring two words per record. Branching or updating more than one word per record presents no particular problem.[10,13,14] Table I contains data (normalized to the 370/145) from these measures. The APCS was between 32 and 110 times faster than the sequential processor, depending upon the algorithm and the number of keys searched. However, system overhead time to process the I/O request and to transfer the data from disc to data memory (no overlap assumed) was charged to the sequential and associative processors. Therefore, after 4000 keys, the APCS times tended to approximate those of the sequential processor since an IBM 2314 device was required for swapping data into the DM.

UPDATE COMPARISONS

In the conventional case, the measurements involved ordered and unordered dictionaries; in the associative case, no dictionary was used. For both cases, we used fixed-length records, 64 bytes/record, and assumed that update consisted of adding records to the file, deleting records from it, and changing fields within each record. As in retrieval, the personnel data base was used for the measurements. It consists of a set of simple domains shown by the general relation $R_1(a_{11}, a_{12}, \cdots, a_{1n})$.

Updating the data base involved a series of searches on the domain a_{1n} such that a subset of a_{1n} was produced: the list of length $|r_{1n}|$.

The set of n-tuples in the relation R can be changed using the APCS Store Data Register command in $N|r_{1n}|$ insertions, where N in the APCS measurements

TABLE I—Retrieval Measurements (Data Normalized to 370/145)

	FACT RETRIEVAL			CONDITIONAL SEARCH AND RETRIEVAL		
	APCS T1	370/145 T2	RATIO T2/T1	APCS	370/145	RATIO T2/T1
1000	.341 ms	10.888 ms	31.8	.695 ms	53.135 ms	76.5
2000	.401	18.345	45.6	1.098	105.928	96.4
4000	.613	36.617	60.0	1.926	211.810	110.0
8000	913.1	1,011.853	1.11	915.35	1,335.624	1.45

- APCS EASIER TO PROGRAM, ALTHOUGH IT TOOK MORE CODE
- APCS REQUIRED 15% LESS STORAGE

consists of three domains: department number, division, and employee ID. In each case, a batch update technique was used, and one-half the data records were modified: 20 percent changed, 15 percent added, 15 percent deleted. Since no dictionary was needed in the associative case, there was a 15 percent savings for data storage for the APCS. The system calls and tables used in the Search and Retrieval calculation were used in the Update measures, as well.

Assuming that the list r_{1n} and the Update list are both in sorted order, data values could be inserted into the list r_{1n} using the parallel transfer capabilities of the APCS in m times the basic cycle time, where m is the number of bytes in the set of n-tuples to be changed.

Table II shows the normalized (to the 370/145) timings for the APCS and a conventional computer (370/145) for the Update measurement. The measurements for the conventional case involved ordered and unordered dictionaries.

The search algorithm for the conventional case was applied to an ordered dictionary, where a binary search technique was used involving $(\lceil \log_2 \nu(x) \rceil) - 1$ average passes; $\nu(x)$ denotes the length of the dictionary. For the unordered case, it was assumed that a linear search technique found the appropriate dictionary key after a search of one-half the dictionary entries.

Since the Search and Retrieval measurements were severely limited by conventional I/O techniques, the Update measures were projected for mass storage devices (semi-conductor memory) holding four million bytes of data. Hypothetically, such devices could also be bubble and LSI memories or fixed-head-per-track discs; in either case, the added memories are characterized by the parallel-by-word-to-AM transfer capability.

Since $N/2$ records were updated over an N-record data base, the conventional times increased as a function of $N^2/2$, whereas the APCS times increased as a function of N. Even though a four-megabyte semiconductor memory would probably not be cost effective for conventional processing, it was hypothesized as the storage media for the conventional processor so as to normalize the influence of conventional, channel-type I/O in the calculations. Even so, the APCS showed a performance improvement of 3.4 orders of magnitude over conventional processing for unordered files consisting of 64,000 records.

SEARCH AND RETRIEVAL WITH RESPECT TO HIERARCHICAL STRUCTURES

A comparison was made between the associative and sequential technologies using hierarchically structured data. Consider a personnel data base containing employee information, wherein each employee may have repeating groups associated with his job and salary histories. The non-normalized relations for this data base are:

employee (man #, name, birthdate, social security number, degree, title, job history)
job history (jobdate, company, title, salary history)
salary history (salary date, salary, percentile)

The normalized form is:

employee (man #, name, birthdate, social security number, degree, title)

TABLE II—Execution Times—Update—APCS vs. Conventional Times (Normalized to 370/145)

NUMBER OF RECORDS	APCS (T1)	CONVENTIONAL (T2) DICTIONARIES		RATIOS (T2/T1) DICTIONARIES	
		UNORDERED	ORDERED	UNORDERED	ORDERED
1000	.046 sec.	2.874	.719 sec.	62.4	15.6
2000	.108	11.386	2.912	105.0	26.9
4000	.216	45.326	10.263	210.0	43.5
8000	.433	180.861	38.091	416.0	87.9
16000	.865	722.422	146.440	825.0	169.0
64000	2.916	11,548.198	2,271.915	3,875.0	770.0

job history (man #, jobdate, company, title)
salary history (man #, jobdate, salary date, salary, percentile)

and the Associative Normal Form is:

$$R_1(a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}, \alpha_1)$$
$$R_2(a_{21}, a_{22}, a_{23}, \alpha_1, \alpha_2)$$
$$R_3(a_{31}, a_{32}, a_{33}, \alpha_1, \alpha_2)$$

The total number of searches, n_1, required for the relation R_1 is the sum of the number of possible search arguments for each domain in R_1. Similarly, the number of searches for R_2 and R_3 can be represented by n_2 and n_3. The number of words, or processing elements (PEs), satisfying a search on R_1 is $|k_1|$, and for R_2, $|k_2|$. Since the α_1 values from the PEs satisfying the search conditions for R_1 are used as arguments for searching R_2, the total number of searches possible for R_2 is equal to $n_2+|k_1|$; analogously, the total number for R_3 is equal to $n_3+|k_2|$. Therefore, the total number of searches for one APCS data load required for the three relations is:

$$T=n_1+n_2+n_3+|k_1|+|k_2|.$$

The first comparison dealt with a search and retrieval problem for a personnel data base consisting of the three relations R_1, R_2, and R_3. These relations required 50,000 records, each contained within two PEs. The problem dealt with the query:

PRINT OUT THE MAN #, SALARY DATE, AND SALARY FOR THOSE EMPLOYEES LQ 40, WITH MS DEGREES, WHO ARE ENGINEERS, AND WHO HAVE WORKED AT HUGHES.

In the associative case, the total number of searches required was:

$$T_1=(2+|k_1|+|k_2|)N$$

where N = number of APCS data loads. (Note: the multi-conditional search on R_1 can be made with one search instruction.)

For the total data base, it was assumed that varying numbers of records satisfied the R_1 search, i.e.,

(EMPLOYEES LQ 40, DEGREE EQ MS, TITLE EQ ENGINEER);

and that a variable number of the records marked after the α_1 search satisfied the R_2 search for 'HUGHES'.

For the conventional case, it was assumed that a random-access storage structure existed and that an unordered dictionary was used for calculating record addresses if needed. An index in R_1 was used to access the records belonging to R_2, and an index in R_3 was used to obtain the values to be printed. The total number of Search Compare instructions required was a function of the length of R_1 (l_1), the number (C) of comparison criteria, and the length ($|P_1|$) of the list formed after a search of R_1 times the average number of items, $\bar{a}$, in the simple domain a_{21}. This is represented by:

$$T_2=Cl_1+|p_1|\bar{a}$$

Holding the term $C\,l_1$ constant in T_2 and examining the two equations T_1 and T_2 it can be seen that, for this particular query, as the number of responding PEs increases, the associative performance improvement ratio will decrease; that is, as the number of α_1 and α_2 searches increases, as a result of an increase in the number of PEs responding to primary and secondary relation multi-conditional searches, the performance improvement ratio for the APCS decreases. This is illustrated by Table III. However, as the number of records increases ($C\,l_1 \rightarrow \infty$), the performance ratio increases for the APCS when holding the number of responding PEs constant over the measures.

Based on the search and retrieval problems that have been investigated, it can be concluded that with respect to conventional random-access storage structures, the performance improvement for the APCS is affected by the amount of parallelism inherent within a search, the number of multi-field searches spread over a set of domains, and the degree of hierarchical intra-structure manipulation, with respect to α_1 and α_2 searches, invoked by the query.

A set of queries that tends to produce the following effects will improve the APCS performance ratio:

- Queries invoking an increase in the number of PEs participating in a series of searches.
- Queries requiring an increasing number of multi-field searches.
- Queries requiring increasing volumes of data (assuming that high-speed parallel I/O devices are available for loading the APCS).

The APCS performance ratio decreases as the queries tend to produce:

- A decreasing number of PEs invoked by a series of searches.

TABLE III—Hierarchical Search and Retrieval (50,000 Records) (Normalized to 370/145)

α_1 AND α_2 SEARCHES	APCS TIME (T1)	IBM 370/145 TIME (T2)	RATIO T2/T1
11	6.167 MS	901.951 MS	150
110	14.785	910.51	62
600	50.87	952.6	18
1100	81.87	989.8	12
2300	150.07	1077.3	7

- An increasing number of α_1 and α_2 searches with respect to hierarchy.
- An increase in the use of slow, conventional channel I/O techniques.

TACC APPLICATION SYSTEM AND ASSOCIATIVE PROCESSING ANALYSIS

After having looked at specific DMS subfunctions on an AP, it is of interest to see their importance applied to the processing activities of a real-time system and to evaluate an AP's impact from a total system viewpoint.

The desired outcome of this study is to have a capability to evaluate the impact of associative processing on the future data automation efforts of the Tactical Air Control System (TACS). A portion of the data management system processing operations were defined, for the purpose of this study, in the context of the on-going Advanced Tactical Command and Control Capabilities (ATCCC) Studies. The results presented below were derived from an analysis of a scenario for the TACC Current Operations Division that was under investigation on a testbed facility at Hanscom Field, Massachusetts. The testbed consisted of functional software for the Current Operations and a data management system operating on an IBM 1800/PDP-8 distributed system.[15]

A real-time associative processor data management system (AP/DMS) study model was developed with equivalence to the TACC testbed system in mind. A practical comparison was made between the two systems in order to determine which system better meets TACC data management requirements. The AP/DMS design also serves as a basis for projections of the future effects of sophisticated application of associative processing technology on hardware, data structures, and future data management systems. The following describes the AP/DMS and the comparison methodology and results. The comparison measurements are normalized to bring the hardware, data structures, and data management systems to equivalent levels.

The testbed system (running on the IBM 1800 computer) and the AP/DMS (running on the APCS) were compared first with both using IBM 2311 discs as peripheral storage, then with both using IBM 2305 fixed-head-per-track discs. In the first case, the number of instruction executions for both systems was multiplied by the average instruction time for the IBM 1800; in the second, the multiplier was the average instruction for the APCS.

AP/DMS

The AP/DMS was required to provide a data management environment for the comparison of associative processing techniques with the sequential techniques used in conventional data management systems. It was aimed at the development of cost-effectiveness and performance ratios and as a background for the analysis of advanced associative processors.

This general statement of requirements was used to derive a set of general capability and performance requirements for the system. The equivalence between the testbed system and the AP/DMS was the overriding requirement. The testbed data base and scenario were kept in the same form to achieve the desired results. The user language was tailored to match both the testbed system and the AP/DMS. The notion of normalizing computer instruction times implies a similarity in the software and the data flow. The I/O requirements were accorded separate but equal status to provide an ability to make both simple and complex changes to the system configuration.

System organization

The concern for system equivalence pervades all phases of system design—starting with the system's logical organization and physical configuration. An AP/DMS system organization aimed at meeting the requirements discussed above was established. The basic organizational concept was that the data processing capability should be concentrated in the associative processor. Such a system organization offers better and faster response to user queries and provides a capability for system growth and change and favorable conditions for system design and implementation.

The APCS computer performs the following functions:

1. Generation, storage, and maintenance of the system data base.
2. Storage of general-purpose and application programs.
3. Dynamic definition of data base organization.
4. Execution of message processing programs and, in support of those programs, retrieval from and manipulation of data base files.
5. Message reformatting—specifically, conversion from data base format to display format.
6. Immediate response to message and data errors, reducing the processing time for inconsistent users queries.

Measurements

The AP/DMS was coded in JOVIAL-like procedures using the APCS instruction set. An instruction path derived from the testbed scenario was given to a counting program, and a comprehensive set of statistics was generated for the AP/DMS. Similar measurements were made from several SDC testbed documents to derive a set of testbed statistics.

Table IV presents the test-run results. The total time for the TACC testbed was 29.9 seconds; the TACC AP/DMS time was 11.8 seconds. These times were normal-

TABLE IV—Comparison Normalized to IBM 1800-2311 Base

	TESTBED	TACC AP/DMS		
	SP ONLY	AP+SP	AP	SP
AVG. TIME PER OPERATION	4810	4810	4810	4810
NO. OF OPERATIONS	2,161,700	994,300	217,000	777,300
OPERATION TIME	10,397,777	4,782,583	1,043,770	3,738,813
I/O OPERATION TIME	19,589,770	7,063,731	8,731	7,055,000
TOTAL TIME	29,987,547	11,846,314	1,052,501	10,793,813
RATIO	TESTBED: AP/DMS	2.5:1		

ized to the IBM 1800/2311 operational times. Table V was normalized to APCS/2305 operational times. In the latter case, the performance ratio was approximately 3:1; that is, the AP performed the equivalent task three times faster than the conventional processor.

Statistics showed that the APCS spent 80 percent of its time in parameter definition and passing functions, whereas the testbed computer spent 3.8 percent of its time with such functions. The APCS defined its instruction operands from data definition parameters located in an AM-resident dictionary, and sequences of code passed these parameters from JOVIAL procedure to procedure.

The associative processor does use more parameters than the sequential processor, but not 20 times as many; statistics showed that 73.7 percent of the AP/DMS time was spent in dictionary operation. It can be concluded that there were too many calls to the dictionary, causing too many parameters to be used and passed. Another factor in the AP/DMS performance is the use of the JOVIAL procedure call technique. This technique uses elaborate register and address saving and restoring code. This code is unnecessary, particularly for one AP procedure calling another AP procedure.

When we reduce the parameter passing and dictionary calls by 90 percent and disregard I/O times, the performance ratio is approximately 30:1—however, we can improve this ratio even more.

The TACC data base used by the AP/DMS was biased in favor of sequential computing. The size of the data base was 400KB, consisting of 50 files containing 1850 domains which involved an average of 68 processing elements. Cost-effective parallel computation techniques favor the reverse—68 domains spread over 1850 processing elements. The TACC data base can be reduced to 500 domains, which would have the net effect of involving more parallel processing elements and reducing the time spent in parameter setup, producing a 60:1 CPU gain over sequential processing for the TACC problems.

CONCLUSIONS AND RECOMMENDATIONS

In order to investigate real-time DMS functions on an associative processor, a hypothetical Associative Processor Computer System (APCS) was described. The description was carried to the instruction level in order to have an associative machine on which to code Air Force Tactical Air Control Center (TACC) and update and retrieval problems for comparison with corresponding conventional (sequential) codings.

A TACC testbed scenario describing an operational demonstration of the TACC Current Operation functions was analyzed and from this analysis it was concluded that of the DMS functions most able to use an associative processor, the great majority of tasks in an operational situation fell in the three data management areas of retrieval, update, and search, with 60 percent of the total processing tasks being associated with data search and retrieval.

For this reason, it was decided to investigate Search and Retrieval and Update subfunctions on the APCS. This investigation involved coding Search and Retrieval and Update problems for the APCS and for a conventional computer, an IBM 370/145. From an analysis of various conventional physical data organizations, the random-access storage structure was chosen for comparison with the APCS because of its similarity to testbed data organizations, because of its widespread use in other systems,[16] and because of other studies relating to list organizations.[10] Hierarchical and non-hierarchical record structures were investigated.

APCS performance improvements, normalized to the IBM 370/145 computer, varied from 32 to 110 times faster for Search and Retrieval and from 15 to 210 times faster for update; this assumes that a half-million-byte mass storage device (semi-conductor memory) with a parallel I/O bandwidth of 1.6 billion bytes/second exists for loading the associative memory. By increasing the size of this device, more cost-effective performance ratios were achieved for the two DMS measures; other mass storage devices which may be more cost-effective for associative memories are bubble memories, fixed head-per-track discs,[17] and LSI memories.

TABLE V—Comparison Normalized to APCS-2305 Base

	TESTBED	TACC AP/DMS		
	SP ONLY	AP+SP	AP	SP
AVG. TIME PER OPERATION	291	291	291	291
NO. OF OPERATIONS	2,161,700	994,300	217,000	777,300
OPERATION TIME	829,055	289,341	63,147	226,194
I/O OPERATION TIME	1,756,270	587,351	351	587,000
TOTAL TIME	2,585,325	876,692	63,498	813,194
RATIO	TESTBED: AP/DMS	3.0:1		

From the testbed scenario analysis, three critical real-time functions were selected to provide us with a real Air Force problem to investigate. We obtained program listings for these functions and converted them to analogous APCS code. The JOVIAL language, augmented with APCS instructions, was used for coding the functions. Due to the data structuring and coding tehniques used, a 3:1 improvement, normalized to the testbed IBM 1800 computer, was initially shown for the APCS. This improvement ratio was due to the fact that almost a literal translation was made for converting testbed data structures to the APCS. Also, the JOVIAL techniques used for coding the APCS required twenty-five times as much overhead for passing parameters between subroutines as did the testbed code. By restructuring the testbed data in order to gain more parallelism and by reducing the parameter passing overhead to that of the testbed, we concluded that a 60:1 improvement could be gained for the APCS, and we would expect to obtain that ratio upon repeating the measures. This assumes that a sufficient quantity of mass storage exists for swapping data into the AMS with a bandwidth of 1.6 billion bytes/sec. Based on our study, recommendations are made for future associative processor research studies relating to high-order languages, firmware implementation techniques, generalized tag operations, LSI design and AP cell implementation, and memory organization and data movement.

ACKNOWLEDGMENT

This work was sponsored by Air Force Systems Command, Rome Air Development Center, Griffiss AFB, New York under Contract No. F30602-72-C-0112.

REFERENCES

1. Linde, R. R., Gates, L. R., Peng, T., *Application of Associative Processing to Real-Time Data Management Functions*, Air Force Systems Command, Rome Air Development Center, Griffiss AFB, New York, 1972.
2. Minker, J., *A Bibliography of Associative or Content-Addressable Memory Systems—1956-1971*, Auerbach Corporation, Philadelphia, 1971.
3. *Session 1—Parallel Processing Systems*, 1972 Wescon Technical Papers, Los Angeles Council, IEEE.
4. Dugan, J. A., Green, R. S., Minker, J., Shindle, W. E., "A Study of the Utility of Associative Memory Processors," *Proc. ACM National Conference*, 1966.
5. Parhami, B., "A Highly Parallel Computing System for Information Retrieval," *AFIPS Conf. Proc.*, Vol. 41, Part II, pp. 681-690, 1972.
6. *DS/2 Data Management System Technical Description*, System Development Corporation, Santa Monica, California.
7. *CDMS Data Management System Technical Description*, System Development Corporation, Santa Monica, California.
8. Kellogg, C. H., "A Natural-Language Compiler for On-Line Data Management," *AFIPS Conf. Proc.*, Vol. 33, Part I, 1968.
9. Hazle, M., *AESOP-B Data Management System*, MITRE Corporation, MTR-851, 1970.
10. DeFiore, C. R., *An Associative Approach to Data Management*, PhD Thesis, Syracuse University, 1972.
11. Childs, D. L., "Description of Set-Theoretic Data Storage," *IFIPS Cong. Proc.*, 1968.
12. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, June, 1970.
13. DeFiore, C. R., Stillman, N. J., Berra, P. B., "Associative Techniques in the Solution of Data Management Problems," *Proc. ACM National Conf.*, 1971.
14. Rudolph, J. A., "A Production Implementation of an Associative Array Processor—STARAN," *AFIPS Conf. Proc.*, Vol. 41, Part I, 1972, pp. 229-242.
15. *Definition of General Purpose and Tactical Air Control Center Functional Software*, System Development Corporation, TM-LX-346/200/00, Lexington, Mass., 1971.
16. Dodd, G., "Elements of Data Management Systems," *Computer Surveys*, June 1969.
17. Barnes, G. H., et al., "The ILLIAC IV Computer, *IEEE Trans. Computers*, Vol. C-17 pp. 746-751, August 1958.

A computer graphics assisted system for management

by ROHI CHAUHAN

Tektronix, Incorporated
Beaverton, Oregon

INTRODUCTION

Everyone agrees that a "picture is worth a thousand words". However, it is not yet obvious that it would make good business sense for managers to make liberal use of Computer Graphing in almost all phases of business decision-making. This paper asserts that it is convenient and practical where something worthwhile can be gained by study of variation of such important parameters as demand, inventory level, or sales, with time or with respect to another independent variable. This assertion is predicated on the assurance that:

A. Graphs can be obtained easily as soon as they are needed and in the form they are needed.
B. Graphs are very easy to obtain and modify, and
C. Graphing is cost effective.

Some examples of activity areas where business data graphing is known to be definitely profitable are corporate planning, purchasing, resource allocation, production scheduling, and investment portfolio analysis. However, today, only in a very few management decision-making processes can it be said that computer data graphing is being used as a daily routine. Reasons are primarily that the three conditions mentioned above have not so far been met to users' satisfaction.

The need for easy graphing with desired flexibility dictates use of computer assisted graphing on display terminals providing both graphic output as well as graphic input and hard copy capabilities. Management systems involving such Computer Display Terminals have been, until recently, quite expensive,[1,2] and difficult to justify for common business applications. Also, the choice of computer display terminal suppliers and their product lines were very limited. The software packages that would really make the job of a "business programmer" easy were practically non-existent. As such, the application of Display Terminal Graphics in business decision-making has remained limited to a very few places, like IBM,[3] and there too apparently on somewhat of an experimental basis. A significant number of industries have been using plotter type graphics,[4] but only sparingly because of inconvenience, lack of flexibility, and expense.

However, after introduction of a less than $4,000 Computer Display Terminal by Tektronix, Inc., in October 1971, the use of high speed interactive graphing terminals in several phases of business planning and control activities has now become an economically practical reality. Several companies are known to have implemented, rather quickly, some simple but elegant and profitable graphics assisted systems. Two companies that have publicly talked about their applications are Uniroyal[5] and International Utilities.[6]

This paper will suggest how it is possible to configure simple low cost Decision Support Systems, via description of a system called GAMA-1 for **G**raphics **A**ssisted **M**anagement **A**pplications. This system is being used at Tektronix Inc., Beaverton, Oregon, for corporate planning and production scheduling purposes. The discussion is focused on characteristics of such business systems, software architecture, simplicity of design, and ease of its usage, all of which, of course, is with reference to the GAMA-1.

SYSTEMS CONFIGURATION

The GAMA-1 system is an imaginative attempt at exploiting capabilities furnished by modern computers in timesharing environment. Tektronix 4010 type computer display terminals, Tektronix 4610 type hardcopy units for the 4010's, knowledge of quantitative methods that are frequently used in decision-making process, simple data organization concepts, and FORTRAN are utilized. Figure 1 shows a multi-terminal GAMA-1 system's configuration. The computer system presently being used is a PDP-10.

FUNCTIONAL REPRESENTATION

As depicted in Figure 2, the GAMA-1 system can be functionally used in three ways, i.e.,

1. Graphing of user data files (complete with specified annotation), movement of the produced pictures as desired to fit suitably on the screen, and finally, making hard copies for a report.
2. Manipulation of user data, e.g., adding and subtracting of two series, updating, smoothing, etc., and then performing the task described above, in step 1.

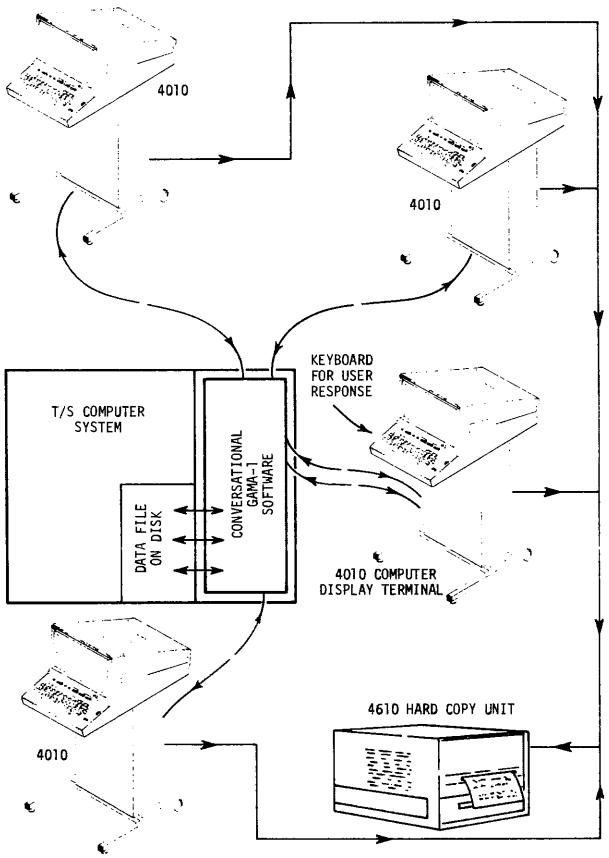


Figure 1—GAMA-1 system's configuration

3. Statistical analysis of the user data, systematic making of forecasts, and generation of reports as in step 1.

For data manipulation, analysis, and forecasting, several useful techniques such as adaptive exponential smoothing, census method X-II, pairing, and regression analysis are furnished, together with the simple but commonly used methods of moving averages, forecasting by graphing and inspection. Also, hooks have been designed, into GAMA-1, so that programs of other desirable techniques can be integrated into the system.

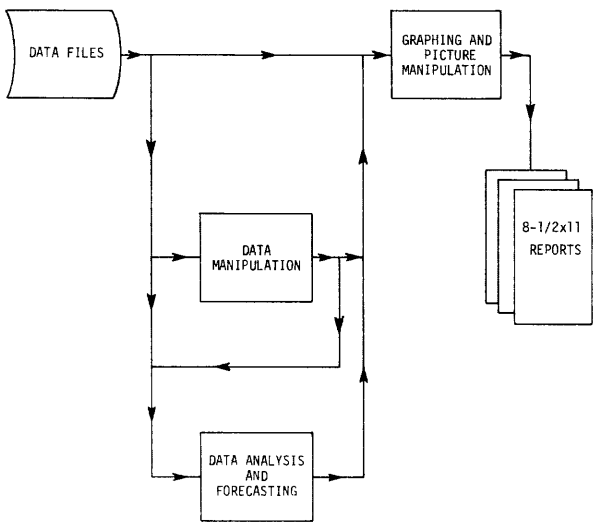


Figure 2—GAMA-1 functional representation

SOFTWARE ATTRIBUTES

The GAMA-1 software has been designed to provide the following most desired features into the system.

1. An ease of use as reflected by the self-guiding, conversational nature of the system. A conversational program is normally thought of as one where the user responds to the queries from the system, one at a time. However, this standard can be surpassed by making all possible choices open to the user known to him at all times. This has been accomplished in the GAMA-1 software by extensive display of menus of the available choices and use of the graphic cross hair for selection of the chosen menu item.
2. Ability to combine usage of several data manipulation, analysis, and forecasting techniques as desired, i.e., adaptability to varying user requirements.
3. Liberal use of graphing throughout the system for easy comprehension of results of various GAMA-1 operations.
4. Easy interfaceability with user's own programs furnishing capability of further growth of the system.
5. No need for the GAMA-1 users to do any applications programming.
6. Mechanism for saving, if so desired, the results that are generated during a GAMA-1 session in the format in which they can be directly fed back as data into the subsequent GAMA-1 sessions.
7. Extensive report generation capabilities allowing the user easily to compose his report pages consisting of graphs of original data as well as the results of the GAMA-1 analysis programs saved before. Complete control over the size and positioning of the graphs, form of the axes, grid super-imposition, alphanumeric annotations (both vertical and horizontal), and movement of the report element on a page is provided. Also, any page of a saved report may be retrieved, modified, and saved again. This feature can be exploited for storing the frequently used preformatted report pages, retrieving them later as needed, filling the blanks (with graphs of new data or annotations) for making up a new report.

SYSTEMS ARCHITECTURE

The GAMA-1 is a file oriented system designed to function in one of the five GAMA modes at any one time, i.e.,

1. Option Select Mode (OSM):
2. Data Manipulation Mode (DMM),
3. Analysis and Forecasting Mode (AFM),
4. Report Generation Mode (RGM), or
5. Help Mode

As illustrated in Figure 3, OSM is the central interface mode which must be entered before any other mode can be invoked. A typical GAMA-1 session will mostly consist of operations in one or more of the three work modes, i.e., the DMM, AFM, and RGM. The system's activity in any of these three work modes is performed by a set of applications programs whose purpose is related to the nomenclature of the modes.

During a normal conversational session the system uses three types of disk files, namely,

GAMA FILE,
DATA FILE, and
REPORT FILE,

whose simplified functional relationships are shown in Figure 3.

Gama file

The GAMA FILE is the central work file used for conducting information flow between various modules of the GAMA-1 software. It is read by the programs in DMM, AFM, and RGM, and it can be modified by the program in DMM and AFM. It can have several segments in it, each of them being a binary representation of the original data or the data resulting from operations on a GAMA FILE SEGMENT by one of the applications programs. Use of this technique saves information storage costs and adds to run time efficiency.

Figure 4 describes the general design philosophy of the GAMA FILE. It is simple. A complete display normally consists of more than one display element, which can be added to, removed from, or moved on the display screen, but may not be subdivided. The display elements may be either "Graph" or an "Annotation", as defined below.

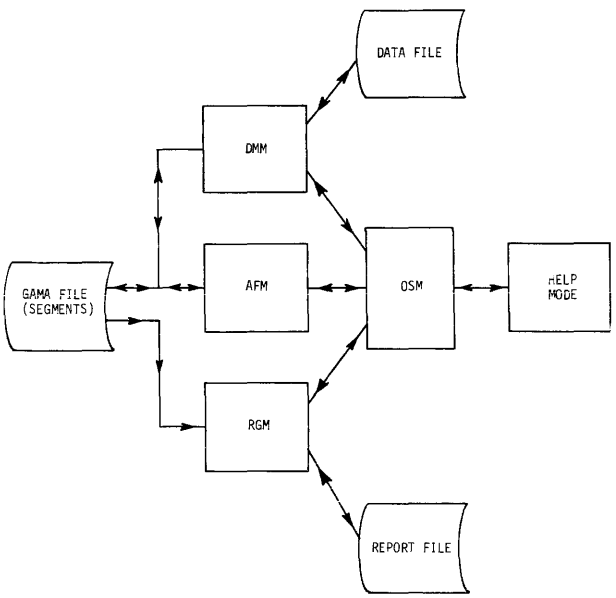


Figure 3—GAMA-1 file relationships

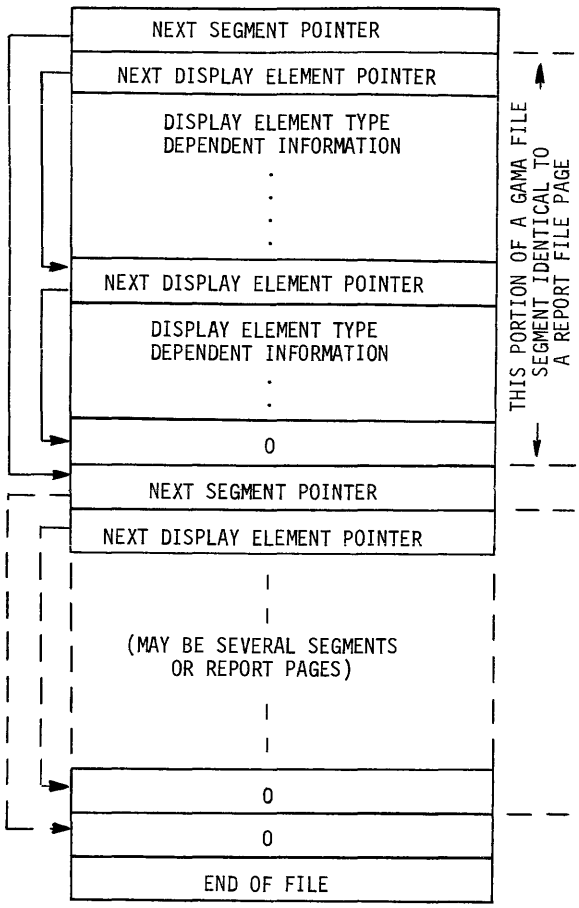


Figure 4—GAMA FILE layout

Graph—Binary representation of data, both numeric (to be graphed) and alphanumeric, that must be treated as one unit. For example, a GAMA FILE segment generated from a DATA FILE (Figure 5) would contain only one element which would be of type "GRAPH".

Annotation—A display element created by the GAMA-1 programs during DMM, AFM, or RGM under user control. Annotations do not have names. Annotations cannot exist outside of RGM unless they are associated with a graph. Annotations may be either graphic or alphanumeric.

Alphanumeric Annotation—It is a string of alphanumeric characters, alternately referred to as labels. Multiple lines are permitted, in both horizontal and vertical configurations.

Graphic Annotation—Lines or points that have been added to a "Graph", by graphic cursor input.

Data file

The DATA FILES, to be created from raw data by using computer system's text editor, contain data to be analyzed by GAMA-1 in character strings. The data is

classified under four categories, which must follow after declaration of the headers by the same name. Figure 5 shows an example of the DATA FILE.

```
/TITLE   "XYZ COMPANY"; "ABC DIVISION";
         "CONSOLIDATED SALES"
/TYPE*   MONTHLY FROM 6706 to 6805
/YUNITS  "THOUSANDS $"
/XUNITS  "YEARS"
/DATA
255  179  87  140  82  53  31
 76   98  29   80  16 100
```

Figure 5—A DATA FILE using one year data

It is to noted that:

- A character string following a "/", until a blank space is encountered, constitutes the HEADER. Once a header has been declared, it is associated with the data following it until a new header is declared or an end of file is encountered.
- All information is entered in free format separated by blanks. The entry of each line is terminated by a carriage return.
- The actual numbers representing data must start on the next line following the header /DATA. Also, the data must be the last entry.
- Length and order of /TITLE, /TYPE, /XUNITS, and/YUNITS type information is arbitrary inasmuch as it occurs before /DATA. Any of these types may also be omitted.
- A ";" in the TITLE type information indicates the beginning of a new line.

The DATA FILES can be generated either by using a text editor, from raw data, or from a GAMA FILE segment by using the INTERPRT command in DMM. A DATA FILE is read only by using the CREATE command in DMM for generation of a GAMMA FILE segment; it is the GAMA FILE segment which is read or written by all other applications programs. After a GAMA FILE segment has been made, the corresponding DATA FILE can be deleted because it can always be re-created by selection of the INTERPRT command in the DMM when required.

* Possible types are DAILY, WEEKLY, MONTHLY, PERIODICAL, QUARTERLY, YEARLY, and PAIRS. In case of PAIRS the format will be /TYPE PAIR nnn, where nnn is the number of pairs; then, in the DATA all the X components of the pairs are listed first followed by the Y components.

Report file

In the Report Generation Mode (RGM), a report page is first composed in a one dimensional real array, called Display Data Array (DDA), and later saved as a page in the REPORT FILE. A report page, i.e., DDA of the RGM, consists of all information that is on display on the screen, with exception of the GAMA-1 system prompts. While in the RGM the DDA always contains all information that is necessary to reproduce the current display. When a display is saved the DDA is written into the REPORT FILE on disk with the specified page number.

The DDA may be filled up gradually, via usage of RGM commands, from information in the GAMA FILE segments or it can be loaded from an existing page in the REPORT FILE.

It is to be noted that the information in the DDA is organized in form of one or more linked display elements. Each of these elements can be manipulated (i.e., deleted, moved, etc.) as one unit by the RGM commands. Also, the other types of display elements have been linked via pointers to allow one refresh* routine to scan through and redraw the complete display in DMM, AFM or RGM. This feature has been made possible by choice of identical structures of the display elements in both GAMA FILE segments and the display data array.

GAMA-1 USAGE CONSIDERATIONS

Because of the system's self-guiding, responsive, and conversational nature, a GAMA-1 session is naturally very creative and interesting. Unlike most systems, a user of GAMA-1 is not required to guess and key in the answers at every stage. All alternatives available to a user are displayed before him as a menu on the right hand margin of the display terminal screen, as shown in Figure 6. A selection of any menu item can be readily made by positioning just the horizontal line of the graphic cross-hair cursor over the desired item and touching a key. Quite often, selection of one menu item results in the display of a subsidiary menu comprising the alternatives that are available in relationship to the previously selected activity. For example, with reference to Figure 6, once the RGM's DISPLAY command in menu I is chosen, the menu II appears. A convenient mechanism for transfer of control from one set of activities to another is built into the system. For example, input of a "$" character in response to any input request, anywhere, results in cancellation of the going activity and the main menu of that mode is activated.

After a while in a GAMA-1 session, a display often gets cluttered because of systems prompts and menus. For such situations, a clean, clutter-free, and current display (i.e., showing effect of MOVE and DELETE commands and addition of any new display elements) can be

* Selection of the REFRESH command clears the entire screen and displays a fresh page. It is often used to redisplay the same information after erasing the screen, free of the unnecessary clutter.

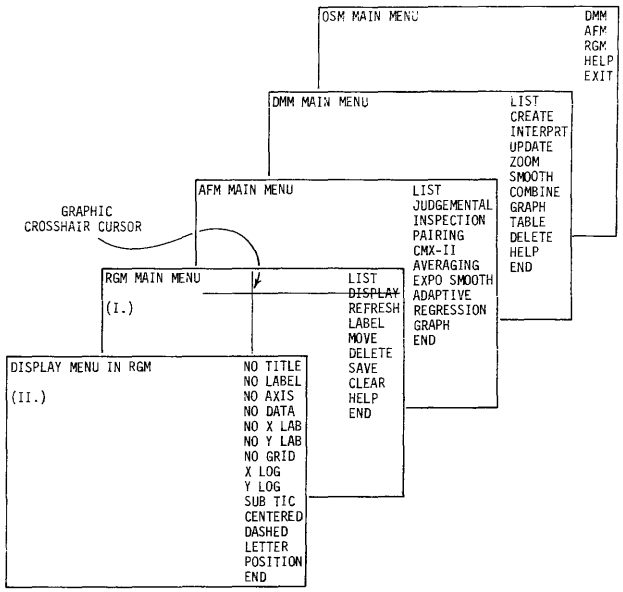


Figure 6—Some of the GAMA-1 menus

obtained by either selection of the REFRESH command or typing an "*" followed by a carriage return.

Default options

Inasmuch as an extensive usage of the menu mechanism furnishes complete control over selection of data manipulation, analysis, and forecasting techniques, and picture (i.e., graph and annotations) manipulation (Figure 7), certain default options are also available to the users. For example, referring again to Figure 6, when menu II is active, selection of END draws the picture with the alternatives that have been chosen in menu II and returns control to the main RGM menu, i.e., menu I. If no alternatives, in menu II, are chosen before END is activated, a graph using the default options is produced. As is evident from Figure 6, a GRAPH command for a quick look at graphic representations of data, without the bother of choosing any display options whatsoever, is also provided in the DMM and AFM so that application of right techniques can be facilitated.

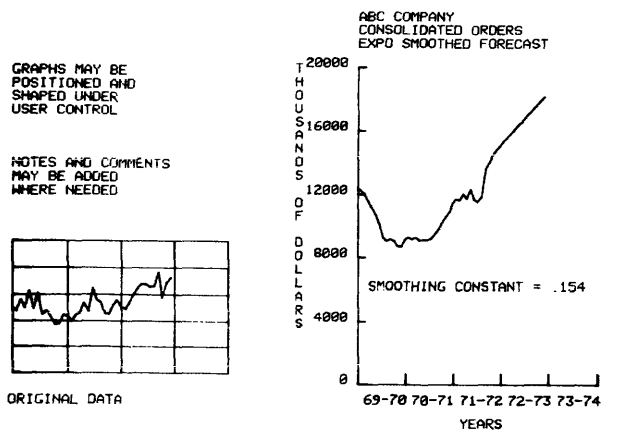


Figure 7—Some picture positioning options—A true hard copy of the display

Control file option

It may be desirable to speed up a GAMA-1 session by cutting down on the bulk of conversation with the system. Such a facility is implemented via a control file option. If this option is selected, completion of a normal GAMA-1 session results in saving of a control file which can later be executed any number of times to reproduce the same GAMA-1 session with different sets of data. The system, when run as governed by a control file, asks only a minimum number of necessary questions, such as identification of the new data, etc.

This option will be particularly appreciated by managers and those users who do not have a need to get into the "nitty gritty" of the GAMA-1 system. They can have a control file for a session tailored to their requirements by an analyst and execute it, with their data, for results with just about no questions asked.

HELP TO USERS

While in the Option Select Mode (OSM), users have a choice of selecting the HELP Mode for a reasonably detailed description of the GAMA-1 system. In this mode, an attempt is made to advise users of the system's various capabilities and to furnish adequate guidance for their usage in a conversational style.

A limited (one page only) amount of information pertaining to a particular work mode (e.g., DMM, AFM, or RGM) is also available if the HELP command is selected out of the main menu in the respective mode.

THE GAMA-1 RESULTS

Results generated during activities in DMM and AFM can be saved into the GAMA FILE as new segments or as replacement of the existing segments. Each of the final report pages can be saved, with respective page numbers, into the REPORT FILE.

Figures 8 and 9 are reproductions of true hard copies of three pages, for example, of a report generated in the RGM. Tabular outputs are produced separately by each applications program.

SUMMARY

At last, the long awaited usage of Computer Graphics in business decision-making is here. It is now also cost-effective. Looking ahead, it appears certain that after five years, if not sooner, we will be able to say that the largest numbers of applications of computer graphing terminals are in business, not in computer aided design as today.

Systems such as GAMA-1 are on the threshold of emergence all over. They are very versatile in being potentially applicable to a variety of business environments, sepa-

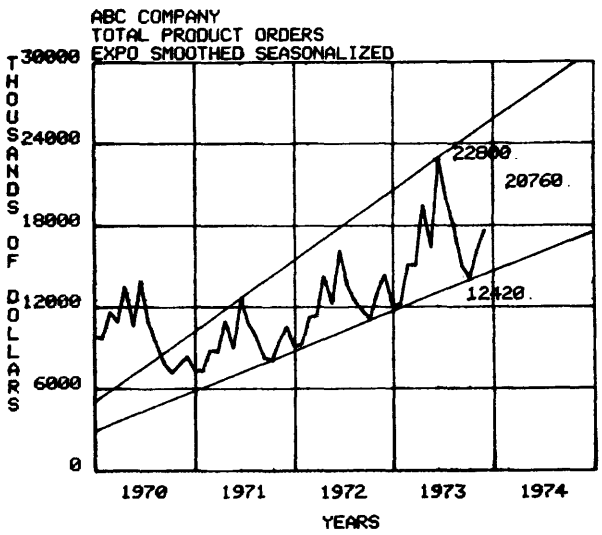


Figure 8—Method of inspection in AFM

rately as well as a functional module of complex Corporate Information Systems.

Future developments will be in the area taking advantage of distributed computing in business decision support systems, exploiting minis or intelligent terminals for processing of pictures and small applications functions, locally, leaving the large number crunching jobs to the big computers in remote locations. The possibility of simple turnkey systems using a large mini is also very promising.

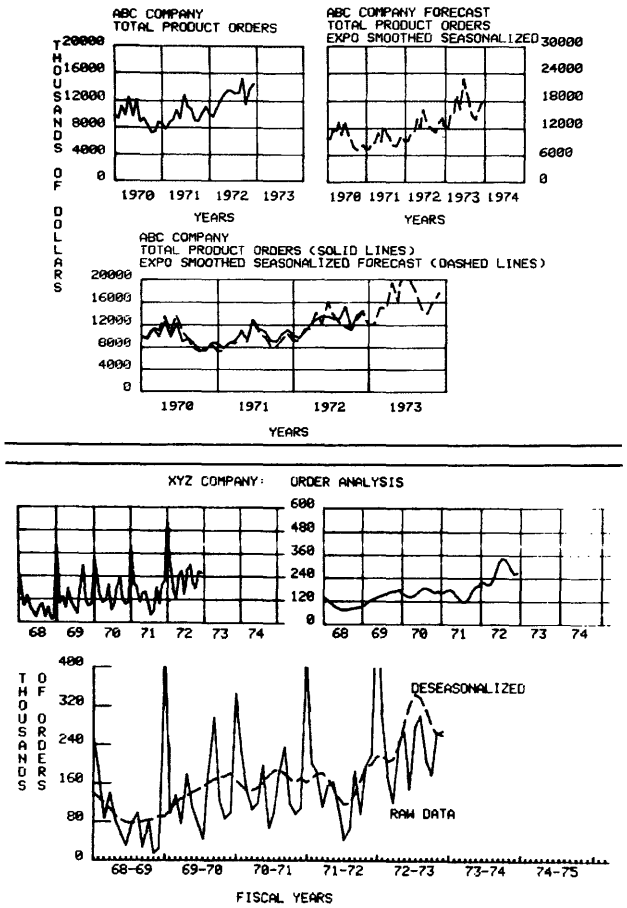


Figure 9—Two report pages

ACKNOWLEDGMENTS

This paper would be incomplete without due acknowledgments to Roz Wasilk, my colleague, for her help by participation in discussions, in working out the details in various file designs, and in programming several modules of the GAMA-1 software. The author also wishes to express his sincere gratitude for the foresight demonstrated by Peter G. Cook and Larry Mayhew in accepting my ideas and furnishing necessary encouragement and support without which the project would never have gotten off the ground. Finally, my thanks to Judy Rule who has been a great help by enthusiastically making sure that the typing and artwork were completed in time, at a very short notice, and to a number of other individuals who have contributed to the success of GAMA-1.

REFERENCES

1. Machover, C., "Computer Graphics Terminals—A Backward Look", *AFIPS Conference Proceedings*, Vol. 40, 1972, AFIPS Press, Montvale, N.J., pp. 439-446.
2. Hagan, T. G., Stotz, R. H., "The Future of Computer Graphics," *AFIPS Conference Proceedings*, Vol. 40, 1972, AFIPS Press, Montvale, N.J., pp. 447-452.
3. Miller, I. M., "Economic Art Speeds Business Decision-making", *Computer Decisions*, Vol. 4, No. 7, July 1972, Hayden Publishing Co., Rochelle Park, N.J., pp. 18-21.
4. Shostack, K., Eddy, C., "Management by Computer Graphics," *Harvard Business Review*, November-December, 1971, pp. 52-63.
5. Friedman, H., *Scheduling with Time-Shared Graphics*, Joint ORSA/TIMS/AIIE Meeting, Atlantic City, N.J., 1972. Unpublished.
6. Curto, C. J., *Analytic Systems for Corporate Planning—The State-of-the-Art*, Joint ORSA/TIMS/AIIE Meeting, Atlantic City, N.J., 1972. Unpublished.

On the use of generalized executive system software

by WILLIAM GORMAN

Computer Sciences Corporation
Silver Spring, Maryland

INTRODUCTION

The characteristic of third generation computing systems that most distinguishes them from previous ones is that they are designed to perform multiprogramming. The purpose of multiprogramming is cost-effective utilization of computer hardware, which is achieved by reducing the CPU time otherwise lost waiting for completion of I/O or operator action. An operating system is necessary to achieve multiprogramming: to schedule jobs, allocate resources, and perform services such as I/O.

Since these systems must be very generalized in order to accommodate the vast spectrum of potential application program requirements, they require some specific information from the user if they are to perform effectively. To supply the needed information and intelligently (efficiently) use the system, then, the user must have some understanding of the operating system's function as related to his particular needs.

A third generation computing system has so much generality that to use or understand it one must wade through stacks of manuals that seem neither clear nor convenient. Problems plague users, who get caught in a juggling act with executive system control language. The unwary become hopelessly involved, generating endless control card changes, with attendant debugging problems and loss of valuable personnel time. Other users have gotten almost mystic about their job control language (JCL), taking an "it works don't touch it" attitude. With indirection such as this, even competent organizations can become very inefficient, using far more hardware, software, and human resources than are actually needed for the work at hand.

Before we surrender and send out an appeal for the "save money" salesmen, let's examine the purposes of executive system software and determine if the application of a little horse-sense doesn't go a long way toward solving our dilemma. Randall[1] notes in his excellent paper on operating systems that the quite spectacular improvements that are almost always made by tuning services "are more an indication of the lamentable state of the original system, and the lack of understanding of the installation staff, than of any great conceptual sophistication in the tools and techniques that these companies use." Clearly, the key to effective use is *understanding* of the operating system and of its interface between a job and the hardware available to perform that job.

It is the purpose of this paper to suggest ways to effectively use a third generation operating system. Most of the examples used will be from the most generalized and complicated of them all—IBM OS/360. Our examination of operating systems will begin with the typical hardware resources of a computing plant and the OS response to those resources. A brief overview of the user tools supplied by the operating system will then be presented, followed by discussions on bugs and debugging and other problems of performance.

Our conclusion will cover the most valuable and cantankerous resource of all—human. Lack of space prevents a complete tutorial, but it is the author's hope that many questions and ideas will be raised in the reader's mind. Perhaps a thoughtful user may see ways to regain effective use of his computing facility, or as Herb Bright says, "Learn to beat OS to its knees."

HARDWARE RESOURCES

CPU time

The first and most valuable resource we shall examine is that of computer time itself. Emerson has said, "Economy is not in saving lumps of coal but in using the time whilst it burns." So it is with computer time. Most large computer shops run their computers 24 hours a day, yet typically their central processing units are doing useful work for far too small a percentage of that time.

Cantrell and Ellison[3] note that "The second by second performance of a multiprogrammed system is always limited by the speed of the processor or an I/O channel or by a path through several of these devices used in series.... If some limiting resource is not saturated, there must be a performance limiting critical path through some series of resources whose total utilization adds up to 100%." To achieve the theortical potential of a computing system, we must manipulate it so as to increase the percentage of resource use. Analysis of the bottlenecks that cause idle time generally reveals that resource needs of companion runs are in conflicting demands in such a manner as to gain greater use of the CPU.

There are three states of CPU time: wait, system, and active. System wait time is time when the CPU is idle. System time is that time spent in supervisory routines, I/O and other interrupt processing, and error handling—most of which is considered overhead. Active time is the time spent executing problem programs. Any reduction of system or wait time makes more time available for problems, thus contributing to greater efficiency.

I/O channel time

The channel handles the transfer of information between main storage and the I/O devices and provides for concurrency of I/O and CPU operation with only a minimum of interference to the CPU. Whenever I/O activity overloads a CPU, idle time can result because the CPU might be forced to wait for completion of I/O activity in order to have data to process. Such cases might be an indication of poor job mix.

Problems also result from the frequency and duration of I/O activity. When data is moved in many small bursts, competition for channels and devices can markedly slow the progress of the operating system.

Main storage

Main storage, or memory, is probably the most expensive and limiting resource in a computing system, besides the CPU itself. Many programs use huge amounts of memory—often more than is available. Since the consumption of memory by programmers seems, like Parkinson's Law, to rise with availability, it is doubtful that expansion of memory will alone solve the average main storage problem. Expansion of memory without a corresponding increase in the average number of jobs residing in memory at execution time is a dubious proposition.

Certain portions of the operating system must reside permanently in main memory in order to execute; but the basic system is too large, with many portions too infrequently used, to make it all resident. Memory not used by the system then serves as a pool of storage from which the system assigns a partition or region to each job step as it is initiated.

One memory scheme used by the 360 breaks memory up into fixed-length parts or partitions, and the user program is allocated the smallest available partition that will accommodate it. Another 360 scheme has the system allocate (de-allocate) memory at execution time in the specific amounts requested. This method is more complicated, with more overhead, but it permits a greater variation in the number and size of jobs being executed.

The most common abuse of memory that I have observed is over-allocation, or more simply the request for greater amounts of memory than are used. Fragmentation, a particularly frustrating problem, results from the requirement that memory be allocated to user jobs in single continuous chunks. As jobs of varying size are given memory, the memory assignments are at first contiguous to one another. When a job finishes, the space it occupied is freed and can be assigned to another job or jobs. However, if subsequent jobs require less than the full amount vacated, small pieces or fragments of unused memory occur and must wait until jobs contiguous to them are ended and can be combined back into usable size. As a result, when we wish to execute programs with large storage needs, the operator often must intervene and delay the initiation of other jobs until enough jobs terminate to create the necessary space. Thus, our CPU can become partially idle by virtue of our need to assemble memory into a single contiguous piece large enough to start our job.

Direct-access storage[4]

Direct-access storage is that medium (drum, disk, or data cell) where data can be stored and retrieved without human intervention. Modern computing demands could not be met without direct-access storage, and operating systems could never reach their full potential without it.

The operating system uses direct-access to store system load modules and routines for use upon demand. Control information about jobs waiting to be processed, jobs in process, and job output waiting to be printed or punched is stored on direct-access devices by the operating system. The system also provides facilities whereby user programs have access to temporary storage to hold intermediate data.

Magnetic tapes

Data can be recorded on magnetic tape in so many different forms that we frequently sacrifice efficiency through lack of understanding. We often encounter difficulty with I/O errors not because of bad tapes, but rather due to incorrect identification to the operating system of recording format and such trivial things as record size. Further errors can develop from contradictions between our program's description and the JCL description of the same data.

We generally inform the operating system of the recording format, etc., through JCL parameters. The system provides many services in the handling of tapes, one of the more important ones being the ability to identify data sets on tape by comparing JCL parameters with labels written as separate files in front of the data being identified. In my diagnostic work, I have identified more I/O errors as due to bad JCL and wrong tape mounts than as legitimate I/O errors. Due to the perishable nature of tape, provision for backup must also be made.

Unit-record devices

Printers, card readers, and punches all fit into this category. The operating system rarely reads or writes user data directly from user programs to these units. Normally, data input from a card reader or output to a punch or

printer is stored as an intermediate file on direct-access devices, so that the system can schedule the use of these relatively slow devices independently of the programs using them. High volume and slow speed can occasionally cause system degradation.

SOFTWARE TOOLS

Many of the tools of the operating system are independently developed segments or modules collected into libraries for use by the system and the user. Additional libraries are created to contain installation-developed routines, programs, and utilities.

Utilities

Supplied with our operating system are numerous service programs or utilies for performing frequently used operations such as sorting, copying, editing, or manipulating programs and data. Among the services supplied are programs to update and list source files, print or punch all or selected parts of data sets, and compare sets of data.

Generally these programs are easy to use once learned, are control card driven, and have "... the priceless ingredient of really good software, abrasion against challenging users."[2] They are generally stable from operating system release to release.

User-written utilities

This brings us to the subject of user-written utilities and programs. A search that I personally made at one installation uncovered over 20 user-written utilities, all occupying valuable disk space and all performing the same function—updating program source files. The only reason for this that I could discover was that the user was unable or unwilling to understand the utility already available. Despite what I termed waste, the writers, to a man, thought their approach sensible.

Many useful utilities have been user-written, and often copies can be secured from the writers at no charge or low charge.

Programming languages and subroutine libraries

Programming languages have been developed to reduce the time, training, expense, and manpower required to design and code efficient problem programs. Essentially they translate human-readable code into machine-readable instructions, thus speeding up the programming process. With these language translators come subroutine libraries of pretested code to handle functions such as deriving the square root of a number of editing sterling currency values.

It is beyond the scope of this paper to discuss language selection, but one note seems in order. When only one or two programmers in an installation are using a complicated higher-level language, those users can encounter serious debugging problems for which no help is available from fellow programmers, due to a lack of expertise in that language.

Input/output control systems

The IOCS portion of the system automatically synchronizes I/O operations with the programs requesting them, provides built-in automatic error handling, and is further extended by system schemes to handle queues of I/O requests from many totally unrelated programs. The system also permits the user to change his output medium with only a simple change to JCL. Users need to write I/O code at the device level only when introducing unique, special-purpose hardware to a system.

Linkers and loaders

The 360 linkage editor combines program segments that were compiled or assembled separately into a single program ready to be loaded. We can therefore make changes without recompiling an entire program. The linkage editor also permits us to create a program too large for available hardware by breaking it into segments that can be executed and then overlaid by other segments yet to be executed.

The loader handles minor linkage tasks and physically loads into main storage the programs we wish to execute.

JCL AS A GLUE

Operating system job control languages (JCL) have been established to enable us to bypass the operator and define precisely to the system the work we wish to perform. JCL reminds me of glue: used properly, it's effective; used poorly, it's a mess.

I look upon the differences between the major operating systems as trade-offs between simplicity and flexibility. UNIVAC and most of the others have opted for simplicity, while IBM has stressed flexibility. For example, UNIVAC uses a very simple control language with single-letter keys that identify the limited range of options permitted via control card. IBM, on the other hand, allows extreme flexibility with literally dozens of changes permitted at the control card level—a not very simple situation.

I consider 360 JCL to be another language—quite flexible, but unfortunately a little too complicated for the average user. To execute a job on the 360 we need three basic JCL cards: a job card to identify our job and mark its beginning, an execute card to identify the specific program we wish to execute, and data definition (DD) cards to define our data sets and the I/O facilities needed to handle them.

When we supply information about a data set via JCL rather than program code, it becomes easier to change

parameters such as block size, type of I/O device used, etc., than it would be with other control languages, simply because no recompilation is required as is frequently so with the other approaches. However, due to the complexity of the process we can unknowingly make mistakes. For example, to create printed output under 360 OS we need only code SYSOUT=A on the DD card describing the data set. Since printed output is usually stored on intermediate disk files, a block size is needed; but unless block size is specified, the output may end up unblocked. Also we might not be able to estimate the volume of printed output that would be generated before our job fails for lack of space allocated to handle the printed output.

Numerous and extremely troublesome problems are generated when our use of JCL is uninformed or haphazard. The large number of JCL parameters required to properly execute a job introduces error possibilities due to sheer volume and an inability to remember every detail required by a large process. Even proficient JCL users may require several trial runs to iron out bugs, while uninformed users frequently give up and instead borrow JCL that allegedly works, even though that JCL may not really match their needs. It therefore becomes imperative that we devise ways to help users assume their proper responsibilities and get away from JCL as much as possible.

IBM assists by making provisions for cataloged libraries of JCL called procedures. To invoke a procedure, a user need supply only a job card and an execute card for each procedure we wish to execute. Within a procedure, necessary details can be coded in symbolic form, with the procedure equating our symbols to proper JCL values. Any value so defined can be changed merely by indicating the symbol and its new value on the execute card invoking the procedure. We can also add or override DD cards and DD card parameters by supplying additional cards containing values to be changed or added.

BUGS AND DEBUGGING

Diagnosing bugs

Diagnosing bugs in user programs requires a clear understanding of the relationship between system services and problem programs.

Bugs call to mind complicated dumps and endless traces, I/O errors that aren't I/O errors at all, and other frustrating experiences. Certain higher-level languages include debugging facilities, trace facilities, and other diagnostic capabilities that can further complicate the diagnostic process whenever they are unable to properly field and identify an error.

Our problem, then, in debugging is the rapid reduction of bugs to their simplest terms, so that proper corrections can be easily and simply made. Here we see the need for informed diagnosticians.

Worthwhile procedures for debugging

Often there is more than one path to a problem solution. We should avoid the trial-and-error, pick-and-choose methods because they are expensive and generally unproductive.

Here is a quick overview of my formula for diagnosing abnormal terminations ("ABEND"s).

First, examine the operating system's reported cause for the ABEND. Try to get a clear understanding of why the operating system thought an error occurred. If any point is not clear, consult the appropriate reference manuals. Research the ABEND description until it is understood.

Before progressing, ask these questions: Can I in general identify the instructions subject to this error? Can I recognize invalid address values that would cause this error? If either answer is yes, proceed; if no, dig some more.

Next, examine the instruction address register portion of the program status word (PSW) which reveals the address of the next instruction to be executed. Check the preceding instruction to see if it was the one that failed. If this process does not locate the failing instruction, perhaps the PSW address was set as the result of a branch.

Check each register at entry to ABEND. Do they look valid or do they look like data or instructions? Are a reasonable percentage of them addresses within our region of memory?

If register conventions are observed, tracing backwards from the error point might reveal where a program went awry. The beauty of higher-level languages is that they consistently follow some sort of register use convention. Once these are learned, debugging becomes simpler.

The process just described continues point by point backwards from the failure to the last properly executed code, attempting to relate the progress of the machine instructions back to the original language statements. If this process fails, attempt to start your search at the last known good instruction executed and work forward.

The same kind of process is followed with I/O errors and errors in general: first identifying the exact nature of the error which the system believes to have occurred; next identifying via register conventions pertinent items such as the data set in error—going as far back into the machine level code as necessary to isolate the error type.

I have a whole string of questions that I ask myself when debugging and it seems that I'm forced to dream up new ones constantly. Let me sum up my approach with four statements:

- Get a clear understanding of the nature of the error.
- Ask yourself questions that bring you backwards from failure point to the execution of valid code.
- If this yields nothing, try to approach the error forward, working from the last known valid execution of code.

• If none of these approaches gets results, try re-creating the error in a small case. Perhaps you'll find that the first step—undertanding the error—was not really completed.

PERFORMANCE PROBLEMS

The improvement of system performance and the elimination of bottlenecks has attracted wide attention of late, perhaps because economy and good business practice dictate that it be so. Unfortunately, no cookbook approach yet exists, and it remains up to us to discover one for ourselves. The tools are legion,[5,6] and are sometimes quite expensive and difficult to interpret. The tools include accounting data, failure statistics, operator shift reports, various types of specially developed system interrogation reports, simulations, and hardware and software monitors. The availability of tools and the level of sophistication of systems personnel may dictate whether these tools are handled in-house or contracted out on a consulting basis.

Our first step is to outline each system resource to determine its fit into the overall system scheme and how our use may affect that fit. A manager might begin this process with a sort of time and motion study, eliminating as many handling problems associated with introduction of jobs to the computer as possible and smoothing the work flow between users, schedulers, messengers, operators, etc., and the computing system itself. The worst bottleneck might, in fact, be the one that prevents a tape or a card deck from being where needed when needed. Assuming that these things have been accomplished, we then poll the operators and the users for their impressions of how well the system serves their needs, at which time we might be forced to reexamine our work procedures.

Our next step properly belongs to systems programmers. Their study should concentrate on those aspects of the system that consume the greatest asset of all—time. There are many techniques and packages available that measure system activity, and these should be put to work. Since the system contains the most heavily used code, our systems programmer places the most active system modules in main memory, intermediate-activity modules in the fastest available secondary storage hardware such as drums, and lower-activity modules in larger-volume, lower-speed (and lower traffic-density) hardware, perhaps large disks or tape. The direct-access addresses of the most frequently fetched routines ought to be resident to eliminate searches for them, and where possible these data sets should reside on the devices with the fastest access speeds. System data sets on direct-access devices with movable arms should have the most active of these routines stored closest to their directories, so that seek arm travel is kept to a minimum. Educated trial and error is necessary before reasonable balance in these areas can be achieved.

Next we study each of the online storage facilities and their use. Questions as to adequacy, reliability, method of backup, and recovery ought to be asked. Criteria for the allocation of direct-access space should be established based upon criticality of use, volume and frequency of use, and cost-effectiveness when compared with available alternatives.

Follow this with the determination and elimination of unused facilities which should be identifiable through the tools previously mentioned.

After our system examination comes an examination of user programs and processes. In this part of our improvement cycle we first look at production programs, starting with the heaviest users of computer time and resources. Many times we find them in an unfinished state, with improvements possible through the elimination of unnecessary steps. An important item to observe is the use of unblocked records or the operation of production programs from source rather than from object or executable load modules. Production programs that require the movement of data from card to disk or tape to disk preparatory to use should be avoided when a simple change to JCL makes this same data available directly to the program without the intervening steps. What is required is that an experienced, knowledgeable, and objective eye be brought to bear upon production programs and other user processes.

Production programs should be examined to determine where the most CPU time is spent in executing and, consequently, what code could be improved to yield the best results.

With OS 360, users can reduce wait time, system time, channel time, and device time by assembling records into blocks set as close as possible to addressable size. This reduces the number of times the I/O routines are invoked, as well as the number of channel and device requests and the seek time expended. With blocking, fewer I/O operations are needed and our programs spend less time in a nonexecutable state waiting on I/O completion. We gain additionally because blocking permits greater density on the storage devices. Many users are unaware of the fact that gaps exist between every record written on a track of a direct-access device. For example, the track capacity on an IBM 2314 disk is 7294 characters. If 80-byte card images are written as one block of 7280 characters, only one write is required to store 91 records on a track; yet if these records are written unblocked, only 40 records will fit on a track because of the inter-record gaps, and 40 writes are invoked to fill that track.

Using large block sizes and multiple buffers introduces additional costs in terms of increased memory required for program execution. Obviously, we should balance these somewhat conflicting demands.

A frustrating problem encountered by users of some systems is that of proper allocation of direct-access space. Since printed or punched output is temporarily stored on

direct-access, such a job's output volume needs to be known so that the user can request the necessary space through his JCL. Jobs can abnormally terminate if insufficient space is allocated to data sets being created or updated. Over-allocation is wasteful and reduces space available for other jobs, as well as permitting excessive output to be created without detection. The space required should if possible be obtained in one contiguous chunk, as less CPU time and access time are used than if data is recorded in several pieces scattered across a unit.

Another problem is locating files or even individual records within files. The system provides catalogs to point to the unit upon which a specific data set resides, but improper use or nonuse of these catalogs or of suitable substitutes can prevent a job from executing, due to an inability to identify where the data set resides. The use of a catalog introduces the problem of searching the catalogs for the individual data set's identity; and if these catalogs are excessively long, useful time (both CPU and I/O) can be lost, since every request for a data set not specifically identified as to unit results in a search of the system catalogs.

Because of the changing nature of user requirements, a data set occupying permanently allocated space might occupy that space long after it is no longer used, simply because we are unaware of the fact. Techniques exist to monitor space, but users can easily cheat them.

Proper estimation and allocation of direct-access space needs is a must, as is the release of unused or temporary space as soon as its usefulness has ceased. At nearly any installation one can find unused data sets needlessly tying up valuable space and sometimes forcing the system to fragment space requests due to the volume of space so wasted.

Proper tape handling is mainly a user problem. Blocking should be employed for efficiency's sake. Data should be blocked to the largest sizes possible, consistent with memory availability, to reduce the amount of tape required to contain a data set and the average I/O transfer time consumed per record. Use of the highest densities provides for faster data transfer and surprisingly greater accuracy because of the built-in error recovery available with high density techniques.

To protect tapes from inadvertent destruction the use of standard-label tapes is encouraged as a site-imposed standard. This permits the operating system to verify that the tape mounted by the operators is the one requested by the user's program.

When processing multiple volume files, two tape drives should be allocated, if available, to permit a program to continue processing rather than wait for the mounting of subsequent tapes when earlier tapes are completed. Further, users should free tape drives not being used in subsequent steps.

Systems programmers usually provide for blocking of card input and certain output types through default values and control card procedure libraries. Users ought not to unblock this I/O.

Careful reduction of the volume of printed data to that actually needed by the ultimate recipient serves both the user and the system by reducing output volume. A typical high speed printer can consume some 35 tons of paper a year, and I can't even estimate the average consumption of cards for a punch unit. Perhaps, like me, you flinch when you observe the waste of paper at a typical computer site. To further reduce the waste of paper, users are well advised to go to microfilm where these facilities exist, particularly for dictionary type output.

It is amazing how many users are still dependent on punched cards in this generation. Processing large volumes of cards requires many extra I/O operations and machine cycles that could be avoided by having these data sets on tape or disk. True, to update a card deck one only needs to physically change the cards involved; but the use of proper update procedures with tape or disk is a far more efficient and accurate use of computer and human time.

This brings us to the subject of software tool usage. The most frequent complaints associated with vendor-supplied utilities are that they are difficult to use and that the documentation is unclear. This is probably due to their generality. Another difficulty is finding out what is available.

To answer the difficulty-of-use problem, we might point out that it is simpler and more productive to try to understand the utilities available than to write and debug new ones. A small installation cannot afford the luxury of user-developed utilities when existing ones will do the job. Often it is better to search for what one is sure must exist than to create it. Still another avenue to investigate would be whether other installations might have the required service routine developed by their users.

Since available utilities are usually more or less unknown to installation users, consideration might be given to assigning a programmer the responsibility of determining the scope of the utilities available and how to use them. This information could then be passed on to fellow programmers. In fact, a good training course would justify its costs by eliminating unnecessary programming and enabling installations programmers to select and use utilities that perform trivial tasks quickly. With vendor-supplied utilities, the first use or two might appear difficult, but with use comes facility.

The use of programming languages ought not be an ego trip. Programmers should be forced to practice "egoless programming"[7] and to follow recognized practices. Efficiency dictates that programs be written in modular form and that they be straightforward, well documented, and without cute programming gimmicks or tricks, lest the next release of the operating system render the program nonexecutable. Programmers themselves should realize that they cannot escape later responsibility for the problems of "tricky" programs, as long as they work for the same employer.

Our evaluation process then continues with research into how new programs and problem program systems are

tested and developed. It is the writer's experience that only rarely has much consideration been given to the efficiency of program development and testing. Frequently the heaviest consumers of computer resources are programmers with trial-and-error methods of debugging and complex or "clever" coding. Again, understanding the system can yield us an insight into relative inefficiencies of program development and how they might be overcome.

Once we have attempted everything within our means, we might then consider outside performance improvement or consulting services.

A final suggestion on resource consumption is a point regarding cost allocation. If a resource does not cost the user, he or she is not likely to try hard to conserve it. Proper allocation of cost in relation to resource use is both a form of control and an attempt to influence users to adjust their demands to the level most beneficial to the system. In short, users ought not to be given a free ride.

A CASE FOR SPECIALISTS

Merging the many parts of a third generation computing system into an effective problem-solving instrument requires that we inform the system of a myriad of details. Efficiency and economy in their most basic forms dictate that we simplify our work as much as possible. Some way *must* be devised to supply to the system with the detail it needs without typing up some 40 percent (as has actually happened—I cannot bear to cite a reference) of programmer time with JCL and associated trivia. What we are striving for is a lessening of the applications programmer's need to know operating system details.

As already noted, the first step is to have knowledgeable system programmers supply as many efficient procedures and other aids as possible and to have them generate a responsive system. Also, those same systems people can generate libraries of control language to cover all the regular production runs (and as many of the development and other auxiliary processes as are practical after sufficient study).

I propose, however, that we go one step further in this area.

Chief programmer team

One exciting concept to emerge recently has been that of the Chief Programmer Team.[8] Significantly increased programmer productivity and decreased system integration difficulties have been demonstrated by the creation of a functional team of specialists, led by a chief programmer applying known techniques into a unified methodology. Managers of programming teams would do well to study this concept. Properly managed, this process has the programmer developing programs full-time, instead of programming part-time and debugging JCL part-time.

Programmer assistance[9]

Our examination of the use of third generation systems has continually pointed out the need for including sufficiently knowledgeable people in the process of system use. A focal point for users needing assistance should be created. This is usually done anyway informally, as users search out fellow programmers and systems people who might have answers to their problems. I propose that experienced, knowledgeable, and systems oriented people be organized into a team to answer user questions and to provide diagnostic assistance. This same group could aid in developing standards, optimizing program code, and teaching courses tailored to user needs. My own experience in a programmer assistance center has shown that such services greatly increases the productivity of a DP installation's personnel.

Diagnostic services

A cadre of good diagnostic programmers should be used to assist programmers who are unable to isolate their program bugs or who need assistance with utilities, JCL, or any other aspect of the operating system. Such a group should keep a catalog of the problems encountered for handy future reference and as a way for determining personnel training needs or system enhancements. The group could aid in locating and correcting software errors by creating small kernels or test programs designed solely to re-create the error. Through the use of such kernels, various error solutions could be tested without disturbing the users main program or process.

Program optimization service

These same personnel might also be charged with research into and development of simple and efficient programming techniques. These techniques could then be implemented in the optimization of the most heavily used programs or systems. Once we identify our largest program consumers of computer time and their most heavily used routines, we can find it cost-effective to thoroughly go over such code, replacing the routines identified with more efficient ones.

Known programming inefficiencies and their correction might be identified by an occasional thorough review of the computer-generated output. For example, the entire output of a weekend might be reviewed in search of poor use of I/O processing. Runs with poor system usage might then be earmarked, and the programmers responsible, notified and given suggestions for possible improvements. The most frequently encountered poor practices might then be subjects of a tutorial bulletin or training session for programmers.

Conventions and standards

"Standards" is a dirty word to many people, but when large numbers of programming personnel are found to be

employing poor practices, our systems group would be charged with developing optimum alternatives. Effective streamlining of frequently used facilities could be accomplished through the publication of tested techniques and standards or conventions. With standards for guidance, the user has a yardstick to determine if his utilization of system resources meets a minimum acceptable level. Prior to the design of new problem program systems, these standards would play an important role in ensuring that optimum use of available facilities was achieved.

"Structured Programming"[10,11] and other developing techniques for making the programming practice manageable should be researched and used as the basis for developing usable installation standards.

Instructors

The accumulation of expertise within a single group would be almost wasteful if this knowledge were not disseminated among the users of the system. A logical extension to our assistance group, then, would be the assignment of instructors who would conduct tutorial seminars and develop tailored training courses and user manuals.

SUMMARY

I have attempted in this paper to give you my views on coping with a highly generalized operating system. I have found that complexity is the main problem facing users of third generation systems. My formula suggests that we insulate the general user/programmer from OS detail to the maximum extent possible, and that we provide the user community with a technically competent consultative group to assist in fielding problems with which the user need not be concerned.

ACKNOWLEDGMENTS

I wish to extend my sincere thanks and appreciation to Messrs. Evmenios Damon and Chesley Looney of the National Aeronautics and Space Administration, Goddard Space Flight Center; Carl Pfeiffer and Ian Bennett of Computer Sciences Corporation; Dave Morgan of Boole and Babbage, Inc.; and John Coffey of the National Bureau of Standards. In particular, I wish to thank Herb Bright of Computation Planning, Inc., for his many hours of discussion and words of encouragement.

REFERENCES

1. Randall, B., "Operating Systems: The Problems of Performance and Reliability," *IFIPS 71*, vol. 1, 281-290.
2. Bright, H. S., "A Philco multiprocessing system," *Proc. FJCC'64 Part II*, AFIPS Vol. 26 pp. 97-141, sec. 14 par. 1, Spartan Books, Washington 1965.
3. Cantrell, H. N., Ellison, A. L., "Multiprogramming System Performance Measurement and Analysis," *AFIPS Conference Proceedings SJCC, 1968*, pp. 213-221.
4. Gorman, W., *Survey and Recommendations for the Use of Direct-Access Storage on the M&DO IBM System/360 Model 95 Computer*, Computer Sciences Corporation Document No. 9101-08800-01TM, (Jan. 1973), NASA Contract No. NAS 5-11790.
5. Lucas, H. C., Jr., *"Performance Evaluation and Monitoring,"* ACM Computing Surveys, 3,3 (Sept. 71), pp. 79-91.
6. Sedgewick, R., Stone, R., McDonald, J. W., "SPY: A Program to Monitor OS/360," *AFIPS Conference Proceedings FICC, 1970*, pp. 119-128.
7. Weinberg, G. M., *The Psychology of Computer Programming*. New York: Van Nostrand Reinhold, 1971.
8. Baker, F. T., "Chief Programmer Team Management of Production Programming," *IBM Systems Journal*, 11,1 (1972), pp. 56-73. This topic was also discussed by Harlan Mills of IBM in his paper, "Managing and motivating programming personnel," given in this morning's session on Resource Utilization in the Computing Community.
9. Damon, E. P., "Computer Performance Measurements at Goddard Space Flight Center," *CPEUG-FIPS Task Group 10*, National Bureau of Standards, Oct. 72.
10. Dijkstra, E. W., "The Structure of the "THE" Multiprogramming System," *CACM*, 11, May 1968, pp. 341-346.
11. McCracken, D., Weinberg, G., "How To Write a Readable FORTRAN Program," *Datamation*, 18,10, Oct. 1972, pp. 73-77.

Language selection for applications

by M. H. HALSTEAD

Purdue University
Lafayette, Indiana

It may strike you as a truism to note that if the solution to a problem depends upon too many variables, we are apt to reduce the multiplicity of variables to one, base an important decision upon that one, and thereafter proceed as though there had never been, and would never be, any reasonable alternative to our choice.

This approach, when applied to the problem of choosing a programming language with which to implement a given class of problems, results in advice of the following sort: For scientific and engineering problems, use FORTRAN; for problems with large data bases, use COBOL; for command-and-control, use JOVIAL; and for systems implementation, use PL/S. Except for academic work avoid ALGOL; and with respect to those opposite ends of the spectrum, APL and PL/1, merely report that you are waiting to see how they work out. Now, obviously, advice of this type might be correct, but just as clearly, it will sometimes be erroneous, primarily because it ignores most of the variables involved.

It would seem only prudent, therefore, to examine some of those dimensions along which languages might be compared, with a view toward increasing our understanding of their relative importance in the decision process. However, there are four items which we should discuss in some detail first. These are:

1. How a programmer spends his time.
2. The difference between local and global inefficiency.
3. The role of the expansion ratio.
4. Variance in programmer productivity.

For the first item, I will use the best data of which I am aware, from an unpublished study of Fletcher Donaldson,[1] done some years ago. Without presenting the details of his study, we note that his measurements of programmers in two different installations, when reduced to hours of programming activity per 40 hour week, yielded the following figures.

	Group 1	Group 2
A. Understanding Objective	3	1.24 Hours/Week
B. Devising Methods		
B1. Finding Approach	4	5.90
B2. Flow Charting	3	1.24
C. Implementation		
C1. Writing Program (Coding)	8	5.60
C2. Preparing Test Data	2	1.24
D. Testing		
D1a. Finding Minor (Syntactic) Errors	4	7.45
D1b. Correcting Minor Errors	1	4.95
D2. Eliminating Test Data Errors	1	.31
D3a. Finding Major (Logical) Errors	6	6.18
D3b. Correcting Major Errors	3	1.24
E. Documenting	3	1.24
Totals	40	40.00

From these data, let us combine those activities which are almost certainly independent of any language which might be chosen: Understanding the Objective, and Finding an Approach. For installation 1 this gives 7 hours, and for installation 2 it gives 7.14 hours, or roughly one day per week. From this it is apparent that, no matter how much improvement might be expected from a given language, it can only operate upon the remaining four days per week.

With respect to the problem of global versus local inefficiencies in programs, there are even fewer data, but the broad outlines are clear, and of great importance. Let us look first at local inefficiency. This is the inefficiency in object code produced by a compiler which results from the inevitable lack of perfection of its optimizing pass. According to a beautiful study reported by Knuth,[2] the difference to be expected from a "finely tuned FORTRAN-*H* compiler" and the "best conceivable" code for the same algorithm and data structure averaged 40 percent in execution time. This is not to say that the difference between well written machine code and code compiled from FORTRAN will show an average difference of the full 40 percent, for even short programs will seldom be "the best conceivable" code for the machine.

With the present state of measurements in this area it is too soon to expect a high degree of accuracy, but let us

accept for the moment a figure of 20 percent for the average inefficiency introduced by even the best optimizing compilers. Now this inefficiency is the only one we are in the habit of examining, since it applies linearly to small as well as to large programs, and we are usually forced to compare only small ones.

The other form of inefficiency is global, and much more difficult to measure. This is the inefficiency introduced by the programmer himself, or by a programming team, due to the size and complexity of the problem being solved. It must be related to the amount of material which one person can deal with, both strategically and tactically, at a given time. Consequently, this form of inefficiency does not even appear in the small sample programs most frequently studied. But because programs in higher level languages are more succinct than their assembly language counterparts, it is responsible for the apparent paradox which has been showing up in larger tests for more than a decade. This paradox probably first appeared to a non-trivial group during the AFADA Tests[3] in 1962, in which a Command-and-Control type program was done in six or seven languages and the results of each compared for efficiency against a "Standard" done in assembly language. When the initial results demonstrated that many of the higher level language versions, despite their obvious local inefficiencies, had produced object code programs which were measurably more efficient than the "Standard," the paradox was attributed to programmer differences, and the entire test was redone. In the second version, the assembly language version was improved to reflect the best algorithm used in a higher level language, and the paradox dissappeared. As recently as 1972, the paradox was still showing up, this time in the Henriksen and Merwin study.[4] In that study, several operating programs which had been written in assembly language were redone in FORTRAN.

If we direct our attention to their comparison of FORTRAN V and SLEUTH, the assembly language for the 1108, we find that two of their three programs gave the same execution times, while for the third they report:

> "In actual performance, the FORTRAN version ran significantly better than the SLEUTH version. If the SLEUTH version were to be recoded using the FORTRAN version as a guide, it is probable that it could be made to perform better than the FORTRAN version."

While the dimensions of global inefficiency have not yet been well established, it tends to become more important as the size of a job increases. Since it is a non-linear factor, it overtakes, and eventually overwhelms, the local inefficiency in precisely those jobs which concern all of us most, the large ones.

This brings us, then, to a consideration of the Expansion Ratio, the "one-to-many" amplification from statements to instructions, provided by computer languages and their compilers. Since the global inefficiency varies more than linearly with program size, it follows that anything which will make an equivalent program "smaller" in its total impact upon the minds of the programmers will reduce its global inefficiency even more rapidly than the reduction in size itself. With an expansion ratio of about four, it appears that the effects of local and global inefficiencies balance for programs of about 50 statements or 2000 instructions, but these figures are extremely rough.

The fourth item listed for discussion, the variance in programmer productivity, enters the language picture in several ways. First, of course, it is because of the large size of this variance that many of the measurements needed in language comparisons have been so inconclusive. But this need not blind us to another facet. Since the time, perhaps 15 years ago, when it was first noted that chess players made good programmers, it has been recognized that programming involved a nice balance between the ability to devise good over-all strategies, and the ability to devote painstaking attention to detail. While this is probably still true in a general way, the increasing power of computer languages tends to give greater emphasis to the first, and lesser to the second. Consequently, one should not expect that the introduction of a more powerful language would have a uniform effect upon the productivity of all of the programmers in a given installation.

On the basis of the preceding discussion of some of the general considerations which must be borne in mind, and leaning heavily upon a recent study by Sammet,[5] let us now consider some of the bulk properties which may vary from language to language. While it is true that for many purposes the difference between a language and a particular compiler which implements it upon a given machine is an important distinction, it is the combined effect of both components of the system which must be considered in the selection process. For that purpose, the costs of some nine items directly related to the system must somehow be estimated. These will be discussed in order, not of importance, but of occurrence.

1. Cost of Learning. If one were hiring a truck driver to drive a truck powered by a Cummins engine, it would be irrelevant as to whether or not an applicant's previous experience included driving trucks with Cummins engines. It would be nice if prior familiarity with a given language were equally unimportant to a programmer, but such is not quite the case. The average programmer will still be learning useful things about a language and its compiler for six months after its introduction, and his production will be near zero for the first two weeks. According to a paper by Garrett,[6] measurements indicate that the total cost of introducing a new language, considering both lost programmer time and lost computer runs, was such that the new language must show a cost advantage of 20 percent over the old in order to overcome it. Since those data were taken quite a long while ago, it is probably safe to estimate

that increasing sophistication in this field has reduced the figure to 10 percent by the present time.

2. Cost of Programming. Obviously, this is one of the most important elements of cost, but difficult to estimate in advance. If we restrict this item strictly to the activity classified as C1 in Donaldson's study, then it would appear to be directly related to the average expansion ratio obtained by that language for the average application making up the programming load. Here the average job size also becomes important, and for a given installation this appears to increase. In fact, some years ago Amaya[7] noted through several generations of computers, it had been true that "The average job execution time is independent of the speed of the computer."
3. Cost of Compiling. While it has been customary to calculate this item from measurements of either source statements of object instructions generated per unit machine time, some painstaking work by Mary Shaw[8] has recently shown that this approach yields erroneous results. She took a single, powerful language, and reduced it one step at a time by removing important capabilities. She then rewrote a number of benchmark programs as appropriate for each of the different compilers. She found that when compilation time was measured for each of the separate, equivalent programs, then the more powerful compiler was slower only if it contained features not utilizable in a benchmark. For those cases in which a more powerful statement was applicable, then the lessened speed per statement of the more powerful compiler was dramatically more than compensated for by the smaller number of statements in the equivalent benchmark program.
4. Cost of Debugging. Here again, since debugging varies directly with the size and complexity of a program, the more succinctly a language can handle a given application, the greater will be the reduction in debugging costs. While the debugging aids in a given compiler are important, their help is almost always limited to the minor errors rather than the major ones.
5. Cost of Optimizing. Since a compiler may offer the option of optimizing or not optimizing during a given compilation, it is possible to calculate this cost seperately, and compare it to the improvements in object code which result. For example, FORTRAN-H[9] has been reported to spend 30 percent more time when in optimization mode, but to produce code which is more than twice as efficient, FORTRAN-hand, Frailey[10] has demonstrated that, under some conditions optimization can be done at negative cost. This condition prevails whenever the avoidance of nonessential instructions can be accomplished with sufficient efficiency to overcompensate for the cost of generating them.
6. Cost of Execution. If, but only if, a good optimizing compiler exists for a given language, then it can be expected that the local inefficiencies of different high level languages will be roughly comparable. The inevitable global inefficiencies will still exist, however. It is here, as well as in programming cost, that a language well suited to the application can yield the greatest savings. Again, the data are inadequate both in quantity and quality, but cost savings of a factor of two are well within my own experience. This does not apply to those languages which are primarily executed in interpretive mode, such as SNOBOL, where large costs of execution must be recovered from even larger savings in programming costs.
7. Cost of Documentation. In general, the more powerful, or terse, or succinct a language is for a given application, the smaller the amount of additional documentation that will be required. While this is true enough as a general statement, it can be pushed too far. It seems to break down for languages which use more symbolic operators than some rough upper limit, perhaps the number of letters in natural language alphabets. Allowing for this exception in the case of languages of the APL class, it follows that the documentation cost of a language will vary inversely with the expansion ratio obtainable in the given applications area.
8. Cost of Modification. Since it is well known that any useful program will be modified, this item is quite important. Here any features of a language which contribute to modularity will be of advantage. Block structures, memory allocation, and compile time features should be evaluated in this area as well as for their effect on initial programming improvement.
9. Cost of Conversion. Since hardware costs per operation have shown continual improvements as new computers have been introduced, it is only reasonable to expect that most applications with a half-life of even a few years may be carried to a new computer, hence this element of potential cost should not be overlooked. If one is using an archaic language, or one that is proprietary, then the cost of implementing a compiler on a new machine may well be involved. While this cost is much less than it was a decade ago, it can be substantial. Even with the most machine-independent languages, the problems are seldom trivial. Languages which allow for the use of assembly language inserts combine the advantage of permitting more efficient code with the disadvantage of increasing the cost of conversion. As noted by Herb Bright,[11] a proper management solution to this problem has existed since the invention of the subroutine, and consists of properly identifying all such usage, and requiring that it comform to the subroutine linkage employed by the language.

In examining the preceding nine elements of cost, it is apparent that many of them depend upon the expansion ratio of a language in a given application area. In deciding

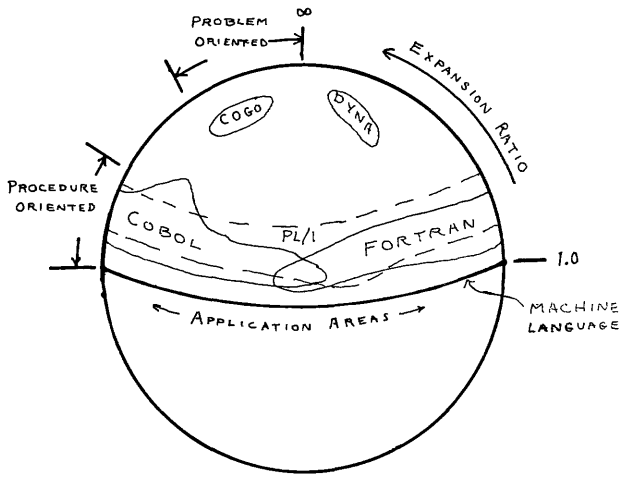


Figure 1

upon a new language, or between two candidates, it might be useful to attempt to plot them upon a globe, with longitude representing possible application areas, and latitude representing any convenient function of the expansion ratio. The plot should look something like Figure 1, where it can be seen that languages range from machine language, in which any application may be handled, girdling the equator, through general purpose, procedure oriented languages in the tropics, to highly specialized, problem oriented languages in the arctic. The pole, which implies an infinite expansion ratio for all applications areas, must be reserved for programming via mental telepathy.

While there is some current research under way[12] which may yield more basic insight into the problems in this area, it is only a year or so old, and the most that can be said is that it is not yet sufficiently developed to be of present assistance.

A very interesting technique which has moved part of the way from research to practice, however, should be mentioned in conclusion. This involves a practical approach to the problem of language extension. Unlike the extensible-language approach, which seemed to open the door to a dangerous, undisciplined proliferation of overlapping and even incompatible dialects within a single installation, this alternate approach to language extension is based upon the older concept of precompilers. As suggested by Garrett[6] and demonstrated by Ghan,[12] dramatic savings in programming costs can be achieved in shops having sufficient work in any narrow application area. This has been done by designing a higher level, more specialized language, and implementing a translator to convert it to a standard procedure oriented language. In this process, the higher-level language may readily permit inclusion of statements in the standard procedure oriented language, and merely pass them along without translation to the second translator or compiler. This process, by itself, has the obvious inefficiency that much of the work done by the first translator must be repeated by the second. While the process has proved economical even with this inefficiency, Nylin[14] has recently demonstrated the ability to reorganize, and thereby remove the redundant elements, of such a preprocessor-compiler system automatically, provided that both are written in the same language.

In summary, let us first note that we have not offered a simple table with line items of preassigned weights, nor a convenient algorithm for producing a yes-no answer to the question "Should I introduce a language specifically to handle a given class of programming jobs." Instead, we realize that, with the current state of the art, it has only been feasible to enumerate and discuss those areas which must be considered in any sound management decision.

From those discussions, however, we may distill at least four guidelines. First, it is abundantly clear that great economies may be realized in those cases in which the following two conditions prevail simultaneously:

(1) There exists a language which is of considerably higher level with respect to a given class of applications programs than the language currently in use, and
(2) The given class of applications programs represents a non-trivial programming work load.

Secondly, there is important evidence which suggests that a higher level language which is a true superset of a high level language already in use in an installation may merit immediate consideration.

Thirdly, it must be remembered that data based upon comparisons between small programs will tend to underestimate the advantage of the higher level language for large programs.

Finally, the potential costs of converting programs written in any language to newer computers should neither be ignored, nor allowed to dominate the decision process.

REFERENCES

1 Donaldson, F. W., *Programmer Evaluation*, 1967, unpublished.
2 Knuth, Donald E., "An Empirical Study of Fortran Programs," *Softwave—Practice and Experience*, Vol. 1, NR 2, April/June 1971, pp. 105-133.
3. The AFADA Tests, conducted by Jordan, were unpublished, but see Datamation, Oct. 1962, pps. 17-19.
4. Henriksen, J. O., and Merwin, R. E., "Programming Language Efficiency in Real-Time Software Systems, *SJCC 1972*, pp. 155-161.
5. Sammet, Jean, "Problems in, and a Pragmatic Approach to, Programming Language Measurement." *FJCC* Vol. 1 39, 1971, pp. 243-252.
6. Garrett, G. A., "Management Problems of an Aerospace Computer Center." *FJCC* 1965.
7. Amaya, Lee, *Computer Center Operation*, unpublished lecture.
8. Shaw, Mary, *Language Structures for Contractible Compilers*, Ph.D. Thesis, Carnegie-Mellon University, Dec. 1971.
9. Lowrey, Edward S., Medlock, C. W., "Object Code Optimization" *CACM*, Vol. 12, Jan. 1969, pp. 13-22.
10. Frailey, Dennis, *A Study of Code Optimization Using a General Purpose Optimizer*, Ph.D. Thesis, Purdue University, Jan. 1971.
11. Bright, Herb, private communications.
12. Halstead, M. H., "Natural Laws Controlling Algorithm Structure," *SIGPLAN Notices* Vol. 7, NR 2, Feb. 1972, pp. 22-26.
13. Ghan, Laverne, "Better Techniques for Developing Large Scale Fortran Programs," *Proc. 1971 Annual ACM Conf.* pp. 520-537.
14 Nylin, W. C. Jr., *Structural Reorganization of Multipass Computer Programs*, Ph.D. Thesis, Purdue, June 1972.

A national scientific and technical information system for Canada

by JACK E. BROWN

National Science Librarian
Ottawa, Canada

ABSTRACT

Canada is in the process of developing a national scientific and technical information (STI) system. It is designed to ensure that scientists, researchers, engineers, industrialists and managers have ready access to any scientific or technical information or publication required in their day-to-day work. In 1970 impetus was given the program when the National Research Council (NRC) was assigned formal responsibility for planning and development, with the National Science Library (NSL) serving as the focal point or coordinating agency for local STI services. During the last two years, emphasis has been placed on the strengthening of two existing networks—a network of 230 libraries linked to the NSL by the "Union List of Scientific Serials in Canadian Libraries"—the CAN/SDI network, a national current awareness service at present using 12 data bases, and linked to the NSL by 350 Search Editors located in all parts of Canada. This service is being expanded to provide remote access to the CAN/SDI data bases by an interactive on-line system. In recent months, steps have been taken to establish regional referral centres and link into the system little used pockets of subject expertise and specialized STI resources.

Global networks for information, communications and computers

by KJELL SAMUELSON

Stockholm University and Royal Institute of Technology
Stockholm, Sweden

ABSTRACT

When working with the concept of worldwide or global networks a clear distinction should be made between at least three different aspects. First of all, information networks based on globally distributed knowledge has a long time bearing on accumulated information and data. Secondly, computer networks that are gradually coming into existence provide various means of processing new or already existing information. For some years to come, computer networks will only to a limited extent provide adequate information and knowledge support. Thirdly, communication networks have existed for decades and are gradually improved by advancements in technology. The combined blend of all three kinds of international networks will have a considerable impact on global socio-economical and geo-cultural trends. If bidirectional broadband telesatellites and universal, multipoint person-to-person communications are promoted, there is hope for "free flow" of information. It appears recommendable that resources should be allocated to this trend rather than an over-emphasis on "massaged" and filtered data in computer networks.

A position paper—Panel session on intelligent terminals—Chairman's introduction

by IRA W. COTTON

National Bureau of Standards
Washington, D.C.

Intelligent terminals are those which, by means of stored logic, are able to perform some processing on data which passes through them to or from the computer systems to which they are connected. Such terminals may vary widely in the complexity of the processing which they are capable of performing. The spectrum ranges from limited-capability point-of-sale terminals through moderately intelligent text-oriented terminals up to powerful interactive graphics terminals. The common thread that ties all these types of terminals together is their processing power and the questions relating to it.

> What, for example, is the proper or most efficient division of labor between the terminals and the central computer?
> What are the limits, if any, to the power which can be provided in such terminals?
> Need we worry about the "wheel of reincarnation" syndrome <ME68> in which additional processing power is continually added to a terminal until it becomes free-standing . . . and then terminals are connected to it?

This session was planned to at least expose to critical discussion some of these questions, if not answer them. Position papers were solicited to cover each of the three points on the spectrum identified above.

> Thornton of the Bureau of Standards points out the need to take a total systems approach and to develop relevant standards—specifically for point-of-sale terminals, but the argument applies to the more sophisticated terminals as well.
>
> Engelbart at Stanford Research Institute discusses his experiences with "knowledge workshop" terminals, and predicts the widespread acceptance of such terminals by knowledge workers of all kinds. That day may be nearer than some might think: two of the three papers for this session were transmitted to the chairman via remote terminal, and one was actually reviewed online in a collaborative mode. In the latter case, author and reviewer were separated by some 3000 miles, and the U. S. Postal Service would not have sufficed to meet publication deadlines.
>
> Van Dam and Stabler of Brown University discuss the opportunities presented by a super intelligent terminal, or "intelligent satellite" in their terms. Such systems offer the most power, but also require the most caution, lest this power be misused or dissipated through poor system design.

It is, of course, impossible to report in advance on the panel discussion which is part of the session. The position papers raise most of the issues that I expect will be discussed. Perhaps some means can be found to report on any new points or insights gleaned from the discussion. In addition, all of the work is ongoing, and all of the authors (and the chairman) welcome further discussion beyond the confines of this conference.

> The standards program at NBS requires participation by manufacturers and users.
>
> Englebart specifically invites inquiries regarding his system in general and the mouse in particular.
>
> The academic community has always been the most open for critical discussion and the exchange of ideas.

In short, we recognize that a session such as this may well raise as many questions as it answers, but we hope that it may serve as a stimulus to further discussion.

A Short Bibliography on Intelligent Terminals

BA173 Bairstow, J. N., "The terminal that thinks for itself," *Computer Decisions*, January 1973, pp. 10-13.

BA268 Bartlett, W. S., et al., "SIGHT, a satellite interactive graphic terminal." 1968 *ACM National Conference*, pp. 499-509.

BE71 Bennett, William C. "Buffered terminals for data communications," *TELECOMM*, June 1971, p. 46.

CH67 Christensen, C., Pinson, E. N., "Multi-function graphics for a large computer system," *FJCC 1967*, pp. 697-712.

CO68 Cotton, Ira W., Greatorex, Frank S., "Data structures and techniques for remote computer graphics," *FJCC 1968*, pp. 533-544.

CP71 "CPU chip turns terminal into stand-alone machine," *Electronics*, 44:12, June 7, 1971, pp. 36-37.

CR72 Crompton, J. C., Soane, A. J., "Interactive terminals for design and management reporting in engineering consultancy," *Online 72*, International Conference on Online Interactive Computing, Sept. 4-7, 1972, Brunel University, Uxbridge, England, pp. 187-208.

DO72 Donato, A. J., "Determining terminal requirements," *Telecommunications*, Sept. 1972, pp. 35-36.

EN73 Engelbart, Douglas C., "Design considerations for knowledge workshop terminals," *NCCE* 1973.

GI73 Gildenberg, Robert F., "Word processing," *Modern Data*, January 1973, pp. 56-62.

GR72 Gray, Charles M., "Buffered terminals ... more bits for the buck," *Communications Report*, December 1972, p. 39.

HO71 Hobbs, L. C., "The rational for smart terminals," *Computer*, Nov.-Dec. 1971, pp. 33-35.

HO72 Hobbs, L. C., "Terminals," *Proceedings of the IEEE*, Vol. 60, No. 11, Nov. 1972, pp. 1273-1284.

IB71 "IBM enters terminal in credit checking race," *Electronics*, March 15, 1971, pp. 34/36.

IN72 "Intelligent terminals," *EDP Analyzer*, 10:4, April 1972, pp. 1-13.

KE72 Kenney, Edward L., "Minicomputer-based remote job entry terminals," *Data Management*, 10:9, Sept. 1971, pp. 62-63.

KN70 Knudson, D., Vezza, A., "Remote computer display terminals." *Computer Handling of Graphical Information*, Seminar, Society of Photographic Scientists and Engineers, Washington, D. C., 1970, pp. 249-268.

MA69 Machover, Carl, "The intelligent terminal." *Pertinent Concepts in Computer Graphics*, Proc. of the Second University of Illinois Conference on Computer Graphics, M. Fairman and J. Nievergelt (eds.), University of Illinois Press, Urbana, 1969, pp. 179-199.

MA172 Machover, C., "Computer graphics terminals—A backward look," *SJCC 1972*, pp. 439-446.

MA270 Marvin, Cloyd E., "Smart vs. 'dumb' terminals: cost considerations," *Modern Data*, 3:8, August 1970, pp. 76-78.

MC71 McGovern, P. J. (ed.), "Intelligent terminals start popping up all over," *EDP Industry Report*, June 30, 1971, pp. 1-7.

ME68 Meyer, T. H., Sutherland, I. E., "On the design of display processors," *CACM*, 11:6 June 1968, pp. 410-414.

NI68 Ninke, W. H., "A satellite display console system for a multi-access central computer," *Proc. IFIP Congress*, 1968, pp. E65-E71.

OB72 O'Brien, B. V., "Why data terminals," *Automation*, 19:5, May 1972, p. 46-51.

PR71 "Programmable terminals," *Data Processing Mag*, 13:2, February 1971, pp. 27-39.

RA68 Rapkin, M. D., Abu-Gheida, O. M., "Stand-alone/remote graphic system," *FJCC 1968*, pp. 731-746.

RO172 Roberts, Lawrence G., "Extension of packet communication technology to a hand-held personal terminal," *SJCC 1972*, pp. 295-298.

RO272 Rosenthal, C. W., "Increasing capabilities in interactive computer graphics terminals," *Computer*, Nov.-Dec. 1972, pp. 48-53.

RO367 Ross, Douglas T., et al., "The design and programming of a display interface system integrating multi-access and satellite computers," *Proc. ACM/SHARE 4th Annual Design Automation Workshop*, Los Angeles, June 1967.

RY72 Ryden, K. H., Newton, C. M., "Graphics software for remote terminals and their use in radiation treatment planning," *SJCC 1972, pp. 1145-1156.*

SA71 Sandek, Lawrence, "Once upon a terminal," Think, 37:9, Oct. 1971, pp. 39-41.

SC171 Schiller, W. L., et al., "A microprogrammed intelligent graphics terminal," *IEEE Trans. Computers*, C-20, July 1971, pp. 775-782.

SC272 Schneiderman, Ron., "Point-of-salesmanship," *Electronic News*, January 17, 1972, pp. 4-5.

SM171 "Smart remote batch terminals share computer processing loads," *Data Processing Mag.*, 13:3, March 1971, pp. 32-36.

SM270 Smith, M. G., "The terminal in the terminal-oriented system," *Australian Comp. J.*, 2:4, Nov. 1970, pp. 160-165.

TH73 Thornton, M. Zane, "Electronic point-of-sale terminals," *NCCE* 1973.

VA73 Van Dam, Andries, Stabler, George M., "Intelligent satellites for interactive graphics." *NCC&E* 1973.

WE73 Wessler, John J., "POS for the supermarket," *Modern Data*, January 1973, pp. 52-54.

A position paper—Electronic point-of-sale terminals

by M. ZANE THORNTON

National Bureau of Standards
Washington, D.C.

The electronic point-of-sale terminal is the newest form of computer technology being introduced into the retail industry. Industry interest in the terminal is focused on its potentially great advantages for retailers in improving their productivity and performance in merchandise control and credit customer control. The electronic point-of-sale terminal's appeal over the standard cash register lies in its potential for impacting the total merchandise system through increasing the speed and accuracy of transactions and providing a method of capturing greater quantities of data essential to the effective management of the merchandise system. At the check-out counter, the terminal equipped with an automatic reading device and credit verification equipment will permit the rapid completion of the sales transaction and, at the same time, capture and enter into the central system all the data necessary for closer, more effective control of the merchandise system.

The full potential of the electronic point-of-sale terminal cannot be realized by simply trying to insert it into the retail environment as a replacement for the electromechanical cash register. The terminal must be effectively integrated into an overall systems approach to the entire merchandising system. It must be equipped with an effective capability to automatically read merchandise tickets and labels; this, in turn, requires the adoption by the retail industry of merchandise identification standards and either a single technology or compatible technologies for marking merchandise and automatically reading the tickets and labels. Further, the terminal must be effectively integrated with supporting computer systems, which raises still other needs related to data communications interconnections, network design and optimization, data standards, and software performance standards and interchangeability criteria. Without a thorough systems approach encompassing the entire merchandising system, the great promise of the electronic point-of-sale terminal may never be realized; indeed, the terminal could become the costly instrument of chaos and widespread disruption in the retail industry.

The National Retail Merchants Association is taking steps to insure that the proper preparations are made to smooth the introduction of the electronic point-of-sale terminal on a broad scale. The Association's first major objective is to develop merchandise identification standards by the end of 1973. At the request of the NRMA, the National Bureau of Standards is providing technical assistance to this effort. Equipment manufacturers, other retailers, merchandise manufacturers, tag and label makers, and other interested groups are also involved.

Given the merchandise identification standards, the emphasis will shift to the implementation of the standards in operational systems where primary effort will be focused on network design, data communications and interfacing terminals with computers, and software development.

Design considerations for knowledge workshop terminals

by DOUGLAS C. ENGELBART

Stanford Research Institute
Menlo Park, California

INTRODUCTION

The theme of this paper ties directly to that developed in a concurrent paper "The Augmented Knowledge Workshop,"[1] and assumes that: "intelligent terminals" will come to be used very, very extensively by knowledge workers of all kinds; terminals will be their constant working companions; service transactions through their terminals will cover a surprisingly pervasive range of work activity, including communication with people who are widely distributed geographically; the many "computer-aid tools" and human services thus accessible will represent a thoroughly coordinated "knowledge workshop"; most of these users will absorb a great deal of special training aimed at effectively harnessing their respective workshop systems—in special working methods, conventions, concepts, and procedural and operating skills.

Within the Augmentation Research Center (ARC), we have ten years of concentrated experience in developing and using terminal systems whose evolution has been explicitly oriented toward such a future environment; from this background, two special topics are developed in this paper:

(1) What we (at ARC) have learned about controlling interactive-display services, and the means we have evolved for doing it—the partiuclar devices (mouse, keyset, keyboard), feedback, and protocol/skill features; and design data, usage techniques, learnability experience, and design data, usage techniques, learnability experience, and relevant needs and possibilities for alternatives and extensions.
(2) Our considerations and attitudes regarding the distribution of functions between terminal and remote shared resources—including assumptions about future-terminal service needs, our networking experience, and foreseen trends in the associated technologies.

References 2-19 include considerable explicit description of developments, principles, and usage (text, photos, and movies) to support the following discussion. Annotation is included, not only to provide a guide for selective follow up, but also to supplement the substance to the body of the paper by the nature of the commentary.

CONTROL MEANS

Introduction

Our particular system of devices, conventions, and command-protocol evolved with particular requirements: we assumed, for instance, that we were aiming for a workshop in which these very basic operations of designating and executing commands would be used constantly, over and over and over again, during hour-after-hour involvement, within a shifting succession of operations supporting a wide range of tasks, and with eventual command vocabularies that would become very large.

THE MOUSE FOR DISPLAY SELECTION

During 1964-65 we experimented with various approaches to the screen selection problem for interactive display work within the foregoing framework. The tests[6,7] involved a number of devices, including the best light pen we could buy, a joy stick, and even a knee control that we lashed together. To complete the range of devices, we implemented an older idea, which became known as our "mouse," that came through the experiments ahead of all of its competitors and has been our standard device for eight years now.

The tests were computerized, and measured speed and accuracy of selection under several conditions. We included measurement of the "transfer time" involved when a user transferred his mode of action from screen selection with one hand to keyboard typing with both hands; surprisingly, this proved to be one of the more important aspects in choosing one device over another.

The nature of the working environment diminished the relative attractiveness of a light pen, for instance, because of fatigue factors and the frustrating difficulty in constantly picking up and putting down the pen as the user intermixed display selections with other operations.

The mouse is a screen-selection device that we developed in 1964 to fill a gap in the range of devices that we were testing. It is of palm-filling size, has a flexible cord attached, and is operated by moving it over a suitable hard surface that has no other function than to generate the proper mixture of rolling and sliding motions for each of the two orthogonally oriented disk wheels that comprise two of the three support points.

Potentiometers coupled to the wheels produce signals that the computer uses directly for X-Y positioning of the display cursor. It is an odd-seeming phenomenon, but each wheel tends to do the proper mix of rolling and sideways sliding so that, as the mouse is moved, the wheel's net rotation closely matches the component of mouse movement in the wheel's "rolling" direction; one wheel controls up-down and the other left-right cursor motion.

Exactly the same phenomenon, applied in the mechanical integrators of old-fashioned differential analyzers, was developed to a high degree of accuracy in resolving the translation components; we borrowed the idea, but we don't try to match the precision. Imperfect mapping of the mouse-movement trajectory by the cursor is of no concern to the user when his purpose is only to "control" the position of the cursor; we have seen people adapt unknowingly to accidental situations where that mapping required them to move the mouse along an arc in order to move the cursor in a straight line.

That the mouse beat out its competitors, in our tests and for our application conditions, seemed to be based upon small factors: it stays put when your hand leaves it to do something else (type, or move a paper), and reaccessing proves quick and free from fumbling. Also, it allows you to shift your posture easily, which is important during long work sessions with changing types and modes of work. And it doesn't require a special and hard-to-move work surface, as many tablets do. A practiced, intently involved worker can be observed using his mouse effectively when its movement area is littered with an amazing assortment of papers, pens, and coffee cups, somehow running right over some of it and working around the rest.

ONE-HANDED, CHORDING KEYSET AS UNIVERSAL "FUNCTION" KEYBOARD

For our application purposes, one-handed function keyboards providing individual buttons for special commands were considered to be too limited in the range of command signals they provided. The one-handed "function keyboard" we chose was one having five piano-like keys upon which the user strikes chords; of the thirty-one possible chords, twenty-six represent the letters of the alphabet. One is free to design any sort of alphabetic-sequence command language he wishes, and the user is free to enter them through either his standard (typewriter-like) keyboard or his keyset.

The range of keyset-entry options is extended by cooperative use of three control buttons on the mouse. Their operation by the mouse-moving hand is relatively independent of the simultaneous pointing action going on. We have come to use all seven of the "chording" combinations, and for several of these, the effect is different if while they are depressed there are characters entered—e.g. (buttons are number 1 to 3, right to left) Button 2 Down-Up effects a command abort, while "Button 2 Down, keyset entry, Button 2 Up" does not abort the command but causes the computer to interpret the interim entry chords as upper case letters.

These different "chord-interpretation cases" are shown in the table of Appendix A; Buttons 2 and 3 are used effectively to add two bits to the chording codes, and we use three of these "shift cases" to represent the characters available on our typewriter keyboard, and the fourth for special, view-specification control. ("View specification" is described in Reference 1.)

Learning of Cases 1 and 2 is remarkably easy, and a user with but a few hours practice gains direct operational value from keyset use; as his skill steadily (and naturally) grows, he will come to do much of his work with one hand on the mouse and the other on the keyset, entering short literal strings as well as command mnemonics with the keyset, and shifting to the typewriter keyboard only for the entry of longer literals.

The keyset is not as fast as the keyboard for continuous text entry; its unique value stems from the two features of (a) being a one-handed device, and (b) never requiring the user's eyes to leave the screen in order to access and use it. The matter of using control devices that require minimum shift of eye attention from the screen during their use (including transferring hands from one device to another), is an important factor in designing display consoles where true proficiency is sought. This has proven to be an important feature of the mouse, too.

It might be mentioned that systematic study of the micro-procedures involved in controlling a computer at a terminal needs to be given more attention. Its results could give much support to the designer. Simple analyses, for instance, have shown us that for any of the screen selection devices, a single selection operation "costs" about as much in entry-information terms as the equivalent of from three to six character strokes on the keyset. In many cases, much less information than that would be sufficient to designate a given displayed entity.

Such considerations long ago led us to turn away completely from "light button" schemes, where selection actions are used to designate control or information entry. It is rare that more than 26 choices are displayed, so that if an alphabetic "key" character were displayed next to each such "button," it would require but one stroke on the keyset to provide input designation equivalent to a screen-selection action. Toward such tradeoffs, it even seems possible to me that a keyboard-oriented scheme could be designed for selection of text entities from the display screen, in which a skilled typist would keep his hands on keyboard and his eyes on the screen at all times,

where speed and accuracy might be better than for mouse-keyset combination.

NOTE: For those who would like to obtain some of these devices for their own use, a direct request to us is invited. William English, who did the key engineering on successive versions leading to our current models of mouse and keyset is now experimenting with more advanced designs at the Palo Alto Research Center (PARC) of Xerox, and has agreed to communicate with especially interested parties.

LANGUAGE, SKILLS AND TRAINING

I believe that concern with the "easy-to-learn" aspect of user-oriented application systems has often been wrongly emphasized. For control of functions that are done very frequently, payoff in higher efficiency warrants the extra training costs associated with using a sophisticated command vocabulary, including highly abbreviated (therefore non-mnemonic) command terms, and requiring mastery of challenging operating skills. There won't be any easy way to harness as much power as is offered, for closely supporting one's constant, daily knowledge work, without using sophisticated special languages. Special computer-interaction languages will be consciously developed, for all types of serious knowledge workers, whose mastery will represent a significant investment, like years of special training.

I invite interested skeptics to view a movie that we have available for loan,[13] for a visual demonstration of flexibility and speed that could not be achieved with primitive vocabularies and operating skills that required but a few minutes (or hours even) to learn. No one seriously expects a person to be able to learn how to operate an automobile, master all of the rules of the road, familiarize himself with navigation techniques and safe-driving tactics, with little or no investment in learning and training.

SERVICE NETWORK

One's terminal will provide him with many services. Essential among these will be those involving communication with remote resources, including people. His terminal therefore must be part of a communication network. Advances in communication technology will provide very efficient transportation of digital packets, routed and transhipped in ways enabling very high interaction rates between any two points. At various nodes of such a network will be located different components of the network's processing and storage functions.

The best distribution of these functions among the nodes will depend upon a balance between factors of usage, relative technological progress, sharability, privacy, etc. Each of these is bound to begin evolving at a high rate, so that it seems pointless to argue about it now; that there will be value in having a certain amount of local processor capability at the terminal seems obvious, as for instance to handle the special communication interface mentioned above.

EXTENDED FEATURES

I have developed some concepts and models in the past that are relevant here, see especially Reference 5. A model of computer-aided communication has particular interest for me; I described a "Computer-Aided Human-Communication Subsystem," with a schematic showing symmetrical sets of processes, human and equipment, that serve in the two paths of a feedback loop between the central computer-communication processes and the human's central processes, from which control and information want to flow and to which understanding and feedback need to flow.

> There are the human processes of encoding, decoding, output transducing (motor actions), and input transducing (sensory actions), and a complementary set of processes for the technological interface: physical transducers that match input and output signal forms to suit the human, and coding/decoding processes to translate between these signal forms in providing I/O to the main communication and computer processes.
>
> In Reference 5, different modes of currently used human communication were discussed in the framework of this model. It derived some immediate possibilities (e.g., chord keysets), and predicted that there will ultimately be a good deal of profitable research in this area. It is very likely that there exist different signal forms that people can better harness than they do today's hand motions or vocal productions, and that a matching technology will enable new ways for the humans to encode their signals, to result in significant improvements in the speed, precision, flexibility, etc. with which an augmented human can control service processes and communicate with his world.
>
> It is only an accident that the particular physical signals we use have evolved as they have—the evolutionary environment strongly affected the outcome; but the computer's interface-matching capability opens a much wider domain and provides a much different evolutionary environment within which the modes of human communication will evolve in the future.
>
> As these new modes evolve, it is likely that the transducers and the encoding/decoding processes will be built into the local terminal. This is one support requirement that is likely to be met by the terminal rather than by remote nodes.

To me there is value in considering what I call "The User-System, Service-System Dichotomy" (also discussed in 5). The terminal is at the interface between these two "systems," and unfortunately, the technologists who develop the service system on the mechanical side of the terminal have had much too limited a view of the user system on the human side of the interface.

That system (of concepts, terms, conventions, skills, customs, organizational roles, working methods, etc.) is to receive a fantastic stimulus and opportunity for evolutionary change as a consequence of the service the computer can offer. The user system has been evolving so placidly in the past (by comparison with the forthcoming era), that there hasn't been the stimulus toward producing an effective, coherent system discipline. But this will change; and the attitudes and help toward this user-system discipline shown by the technologists will make a very large difference. Technologists can't cover both sides of the interface, and there is critical need for the human side (in this context, the "user system") to receive a lot of attention.

What sorts of extensions in capability and application are reasonable-looking candidates for tomorrow's "intelligent terminal" environment? One aspect in which I am particularly interested concerns the possibilities for digitized strings of speech to be one of the data forms handled by the terminal. Apparently, by treating human speech-production apparatus as a dynamic system having a limited number of dynamic variables and controllable parameters, analysis over a short period of the recent-past speech signal enables rough prediction of the forthcoming signal, and a relatively low rate of associated data transmission can serve adequately to correct the errors in that prediction. If processors at each end of a speech-transmission path both dealt with the same form of model, then there seems to be the potential of transmitting good quality speech with only a few thousand bits per second transmitted between them.

The digital-packet communication system to which the "computer terminal" is attached can then become a very novel telephone system. But consider also that then storage and delivery of "speech" messages are possible, too, and from there grows quite a spread of storage and manipulation services for speech strings, to supplement those for text, graphics, video pictures, etc. in the filling out of a "complete knowledge workshop."

If we had such analog-to-digital transducers at the display terminals of the NLS system in ARC, we could easily extend the software to provide for tying the recorded speech strings into our on-line files, and for associating them directly with any text (notes, annotations, or transcriptions). This would allow us, for instance, to use cross-reference links in our text in a manner that now lets us by pointing to them be almost instantly shown the full text of the cited passage. With the speech-string facility, such an act could let us instantly hear the "playback" of a cited speech passage.

Records of meetings and messages could usefully be stored and cited to great advantage. With advances in speech-processing capability, we would expect for instance to let the user ask to "step along with each press of my control key by a ten-word segment" (of the speech he would hear through his speaker), or "jump to the next occurrence of this word". Associated with the existing "Dialogue Support System" as discussed in Reference 1, this speech-string extension would be very exciting. There is every reason to expect a rapid mushrooming in the range of media, processes, and human activity with which our computer terminals are associated.

ACKNOWLEDGMENTS

During the 10 year life of ARC many people have contributed to the development of the workshop using the terminal features described here. There are presently some 35 people—clerical, hardware, software, information specialists, operations researchers, writers, and others—all contributing significantly toward our goals.

ARC research and development work is currently supported primarily by the Advanced Research Projects Agency of the Department of Defense, and also by the Rome Air Development Center of the Air Force and by the Office of Naval Research. Earlier sponsorship has included the Air Force Office of Scientific Research, and the National Aeronautics and Space Administration. Most of the specific work mentioned in this paper was supported by ARPA, NASA, and AFOSR.

REFERENCES

1. Engelbart, D. C., Watson, R. W., Norton, J. C., The Augmented Knowledge Workshop, AFIPS Proceedings National Computer Conference, June 1973, (SRI-ARC Journal File 14724)
2. Engelbart, D. C., *Augmenting Human Intellect: A Conceptual Framework*, Stanford Research Institute Augmentation Research Center, AFOSR-3223, AD-289 565, October 1962, (SRI-ARC Catalog Item 3906)
   The framework developed a basic strategy that ARC is still following—"bootstrapping" the evolution of augmentation systems by concentrating on developments and applications that best facilitate the evolution and application of augmentation systems. See the companion paper[1] for a picture of today's representation of that philosophy; the old report still makes for valuable reading, to my mind—there is much there that I can't say any better today.

   In a "science-fiction" section of the report, I describe a console with features that are clear precedents to the things we are using and doing today—and some that we haven't yet gotten to.
3. Engelbart, D. C., A Conceptual Framework for the Augmentation of Man's Intellect Vistas," in *Information Handling*, Howerton and Weeks (Editors), Spartan Books, Washington, D. C., 1963, pp. 1-29, (SRI-ARC Catalog Item 9375)
   This chapter contains the bulk of the report[2]; with the main exclusion being a fairly lengthy section written in story-telling, science-fiction prose about what a visit to the augmented workshop of the future would be like. That is the part that I thought tied it all together—but today's reader probably doesn't need the help the reader of ten years ago did. I think that the framework developed here is still very relevant to the topic of an augmented workshop and the terminal services that support it.
4. Engelbart, D. C., Sorenson, P. H., *Explorations in the Automation of Sensorimotor Skill Training*, Stanford Research Institute, NAVTRADEVCEN 1517-1, AD 619 046, January 1965, (SRI-ARC Catalog Item 11736).
   Here the objective was to explore the potential of using computer-aided instruction in the domain of physical skills rather than of conceptual skills. It happened that the physical skill we chose, to make for a manageable instrumentation problem, was operating

the five-key chording keyset. Consequently, here is more data on keyset-skill learnability; it diminished the significance of the experiment on computerized skill training because the skill turned out to be so easy to learn however the subject went about it.

5. Engelbart, D. C., *Augmenting Human Intellect: Experiments, Concepts, and Possibilities—Summary Report* Stanford Research Institute, Augmentation Research Center, March 1965, (SRI-ARC Catalog Item 9691).
   This includes a seven-page Appendix that describes our first keyset codes and usage conventions—which have since changed. Two main sections of about twelve pages, each of which is very relevant to the general topic of "intelligent terminal" design, are discussed above under "Extended Features."
6. English, W. K., Engelbart, D. C., Huddart, B., *Computer Aided Display Control—Final Report* Stanford Research Institute, Augmentation Research Center, July 1965, (SRI-ARC Catalog Item 9692).
   About twenty percent of this report dealt explicitly with the screen-selection tests (that were published later in [7]; most of the rest provides environmental description (computer, command language, hierarchical file-structuring conventions, etc.) that is interesting only if you happen to like comparing earlier and later stages of evolution, in what has since become a very sophisticated system through continuous, constant-purpose evolution.
7. English, W. K., Engelbart, D. C., Berman, M. A., " Display-Selection Techniques for Text Manipulation," *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-8, No. 1, pp. 5-15, March 1967, (SRI-ARC Catalog Item 9694).
   This is essentially the portion of [6] above that dealt with the screen-selection tests and analyses. Ten pages, showing photographs of the different devices tested (even the knee-controlled setup), and describing with photographs the computerized selection experiments and displays of response-time patterns. Some nine different bar charts show comparative, analytic results.
8. Licklider, J. C. R., Taylor, R. W., Herbert, E., "The Computer as a Communication Device," *International Science and Technology*, No. 76, pp. 21-31, April 1968, (SRI-ARC Catalog Item 3888).
   The first two authors have very directly and significantly affected the course of evolution in time-sharing, interactive-computing, and computer networks, and the third author is a skilled and experienced writer; the result shows important foresight in general, with respect to the mix of computers and communications in which technologists of both breeds must learn to anticipate the mutual impact in order to be working on the right problems and possibilities. Included is a page or so describing our augmented conferencing experiment, in which Taylor had been a participant.
9. Engelbart, D. C., *Human Intellect Augmentation Techniques, Final Report* Stanford Research Institute, Augmentation Research Center, CR-1270, N69-16140, July 1968, (SRI-ARC Catalog Item 3562).
   A report especially aimed at a more general audience, this one rather gently lays out a clean picture of research strategy and environment, developments in our user-system features, developments in our system-design techniques, and (especially relevant here) some twenty pages discussing "results," i.e. how the tools affect us, how we go about some things differently, what our documentation and record-keeping practices are, etc. And there is a good description of our on-line conferencing setup and experiences.
10. Engelbart, D. C., "Augmenting Your Intellect," (Interview With D. C. Engelbart), *Research Development*, pp. 22-27, August 1968, (SRI-ARC Catalog Item 9698).
    The text is in a dialog mode—me being interviewed. I thought that it provided a very effective way for eliciting from me some things that I otherwise found hard to express; a number of the points being very relevant to the attitudes and assertions expressed in the text above. There are two good photographs: one of the basic work station (as described above), and one of an on-going augmented group meeting.
11. Engelbart, D. C., English, W. K., "A Research Center for Augmenting Human Intellect," *AFIPS Proceedings-Fall Joint Computer Conference*, Vol. 33, pp. 395-410, 1968, (SRI-ARC Catalog Item 3954).
    Our most comprehensive piece, in the open literature, describing our activities and developments. Devotes one page (out of twelve) to the work-station design; also includes photographs of screen usage, one of an augmented group meeting in action, and one showing the facility for a video-based display system to mix camera-generated video (in this case, the face of Bill English) with computer-generated graphics about which he is communicating to a remote viewer.
12. Haavind, R., "Man Computer 'Partnerships' Explored," *Electronic Design*, Vol. 17, No. 3, pp. 25-32, 1 February, 1969, (SRI-ARC Catalog Item 13961).
    A very well-done piece, effectively using photographs and diagrams to support description of our consoles, environment, working practices, and experiences to a general, technically oriented reader.
13. *Augmentation of the Human Intellect*—A Film of the SRI-ARC, Presentation at the 1969 ASIS Conference, San Francisco, (A 3-Reel Movie). Stanford Research Institute, Augmentation Research Center, October 1969, (SRI-ARC Catalog Item 9733).
14. Field R. K., "Here Comes the Tuned-In, Wired-Up, Plugged-In, Hyperarticulate Speed-of-Light Society—An Electronics Special Report: No More Pencils, No More Books—Write and Read Electronically," *Electronics*, pp. 73-104, 24 November, 1969, (SRI-ARC Catalog Item 9705).
    A special-feature staff report on communications, covering comments and attitudes from a number of interviewed "sages." Some very good photographs of our terminals in action provide one aspect of relevance here, but the rest of the article does very well in supporting the realization that a very complex set of opportunities and changes are due to arise, over many facets of communication.
15. Engelbart, D. C., "Intellectual Implications of Multi-Access Computer Networks," paper presented at *Interdisciplinary Conference on Multi-Access Computer Networks*, Austin, Texas, April 1970, preprint, (SRI-ARC Journal File 5255).
    This develops a picture of the sort of knowledge-worker marketplace that will evolve, and gives examples of the variety and flexibility in human-service exchanges that can (will) be supported. It compares human institutions to biological organisms, and pictures the computer-people networks as a new evolutionary step in the form of "institutional nervous systems" that can enable our human institutions to become much more "intelligent, adaptable, etc." This represents a solid statement of my assumptions about the environment, utilization and significance of our computer terminals.
16. Engelbart, D. C., SRI-ARC Staff, *Advanced Intellect-Augmentation Techniques—Final Report*, Stanford Research Institute, Augmentation Research Center, CR-1827, July 1970, (SRI-ARC Catalog Item 5140).
    Our most comprehensive report in the area of usage experience and practices. Explicit sections on: The Augmented Software Engineer, The Augmented Manager, The Augmented Report-Writing Team, and The Augmented Presentation. This has some fifty-seven screen photographs to support the detailed descriptions; and there are photographs of three stages of display-console arrangement (including the one designed and fabricated experimentally by Herman Miller, Inc, where the keyboard, keyset and mouse are built into a swinging control frame attached to the swivel chair).
17. Roberts, L. C., *Extensions of Packet Communication Technology to a Hand Held Personal Terminal*, Advanced Research Projects Agency, Information Processing Techniques, 24 January, 1972. (SRI-ARC Catalog Item 9120).
    Technology of digital-packet communication can soon support mobile terminals; other technologies can soon provide hand-held display terminals suitable for interactive text manipulation.
18. Savoie, R., *Summary of Results of Five-Finger Keyset Training Experiment*, Project 8457-21, Stanford Research Institute, Bioengineering Group, 4, p. 29, March 1972, (SRI-ARC Catalog Item 11101).

Summarizes tests made on six subjects, with an automated testing setup, to gain an objective gauge on the learnability of the chording keyset code and operating skill. Results were actually hard to interpret because skills grew rapidly in a matter of hours. General conclusion: it is an easy skill to acquire.

19. *DNLS Environment* Stanford Research Institute, Augmentation Research Center, 8, p. 19, June 1972, (SRI-ARC Journal File 10704).
Current User's Guide for ARC's Display Online System (DNLS). Gives explicit description on use of the keyset, mouse, and the basic interaction processes.

APPENDIX A: MOUSE AND KEYSET, CODES AND CASES

Note: We generally use the keyset with the left hand; therefore, "a" is a "thumb-only" stroke. Of the three buttons on the mouse, the leftmost two are used during keyset input effectively to extend its input code by two bits. Instead of causing character entry, the "fourth case" alters the view specification; any number of them can be concatenated, usually terminated by the "f" chord to effect a re-creation of the display according to the altered view specification.

Mouse Buttons:					000	010	100	110	
Case					-0-	-1-	-2-	-3-	
Keyset Code									
O	O	O	O	X	a	A	!		show one level less
O	O	O	X	O	b	B	"		show one level deeper
O	O	O	X	X	c	C	#		show all levels
O	O	X	O	O	d	D	$		show top level only
O	O	X	O	X	e	E	%		current statement level
O	O	X	X	O	f	F	&		re-create display
O	O	X	X	X	g	G	'		branch show only
O	X	O	O	O	h	H	(		g off
O	X	O	O	X	i	I	)		show content passed
O	X	O	X	O	j	J	@		i or k off
O	X	O	X	X	k	K	+		show content failed
O	X	X	O	O	l	l	−		show plex only
O	X	X	O	X	m	M	*		show statement numbers
O	X	X	X	O	n	N	/		hide statement numbers
O	X	X	X	X	o	o	^		frozen statement windows
X	O	O	O	O	p	p	0		frozen statement off
X	O	O	O	X	q	q	1		show one line more
X	O	O	X	O	r	R	2		show one line less
X	O	O	X	X	s	S	3		show all lines
X	O	X	O	O	t	T	4		first lines only
X	O	X	O	X	u	U	5		inhibit refresh display
X	O	X	X	O	v	V	6		normal refresh display
X	O	X	X	X	w	W	7		all lines, all levels
X	X	O	O	O	x	X	8		one line, one level
X	X	O	O	X	y	Y	9		blank lines on
X	X	O	X	O	z	Z	=		blank lines off
X	X	O	X	X	,	<	[		(nothing)
X	X	X	O	O	.	>	]		(nothing)
X	X	X	O	X	;	:	−		(nothing)
X	X	X	X	O	?	/	ALT		centerdot
X	X	X	X	X	SP	TAB	CR		(nothing)

APPENDIX B: PHOTOGRAPHS

Figure 1—Our current standard work station setup: Mouse in right hand controls cursor on screen: keyset under left hand supplements keyboard for special, two-handed command execution operation. Separation of control and viewing hardware is purposeful, and considered by us to be an advantage enabled by computerized work stations.

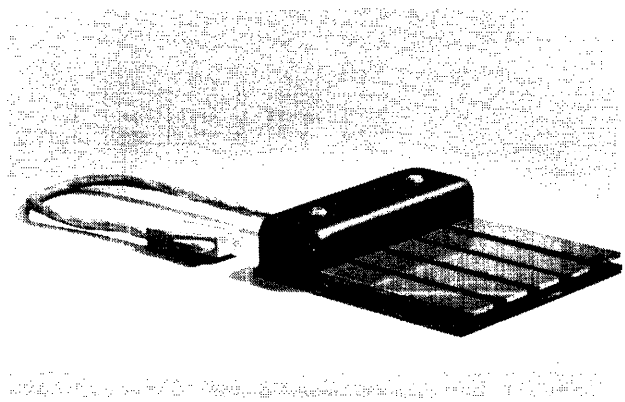


Figure 2—Closeup of Keyset. Finger pressure and key travel are quite critical. It took many successive models to achieve a really satisfactory design.

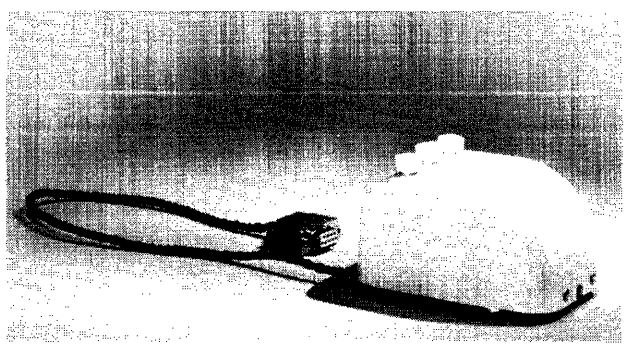


Figure 3—Closeup of Mouse. There are characteristics of the "feel," depending upon the edging of the wheels, the kind of bearings, etc. that can make considerable difference. We happened to hit on a good combination early, but there have been subsequent trials (aimed at improvements, or where others more or less copied our design) that didn't work out well. The play in the buttons, the pressure and actuating travel, are also quite important.

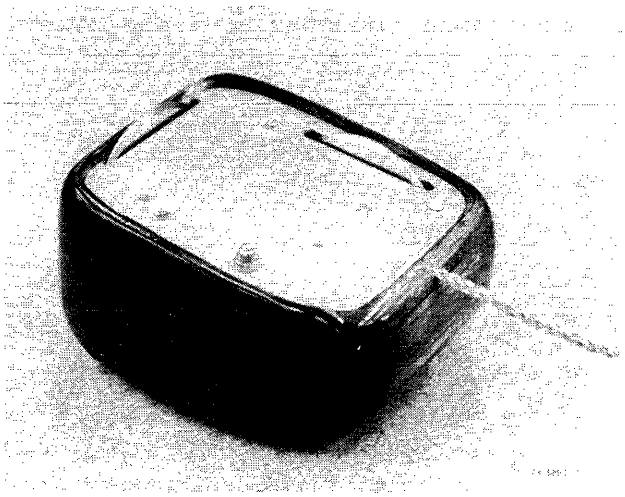


Figure 4—Closeup of underside of mouse (old model), showing orthogonal disk-wheels. We now bring the flexible cable out the "front." Size and shape haven't changed, in later models. Bill English (the operator in Fig. 1, and mentioned in the text above) is now experimenting with new mouse sizes and shapes.

Intelligent satellites for interactive graphics*

by ANDRIES VAN DAM

and

GEORGE M. STABLER

Brown University
Providence, Rhode Island

INTRODUCTION

Semantics

In the last four or five years it has become increasingly fashionable to speak of "intelligent," "smart," or "programmable" terminals and systems. Very few mainframe or peripheral manufacturers omit such a device from their standard product line. Although "intelligence," like beauty or pornography, is in the eye of the beholder, the adjective generally connotes that the device has a degree of autonomy or processing ability which allows it to perform certain (classes of) tasks without assistance from the mainframe to which it is connected. Many such devices are programmable by virtue of including a mini, microprogrammable or micro computer.**

While operational definitions are pretty hazy and nonstandard, we call a device a terminal if a user interacts with a mainframe computer (host) through it (e.g., a teletype or an alphanumeric display console). Hobbs[15] lists 6 classes of terminals:**

(1) keyboard/printer terminals;
(2) CRT terminals;
(3) remote-batch terminals;
(4) real-time data-acquisition and control terminals;
(5) transaction and point-of-sale terminals;
(6) smart terminals.

We consider the terminal to be intelligent if it contains hard, firm, and/or software which allows it to perform alphanumeric or graphic message entry, display, buffering, verifying, editing, and block transmissions, either on mainframe or human command. Note that if the terminal contains a mini, micro or microprogrammable computer which runs a standard program to service the terminal, and not arbitrary, user loaded programs, the terminal has a fixed function and is still just an intelligent *terminal* by our definition† (e.g., a VIATRON, or SYCOR alphanumeric display). Only when the device contains a general purpose computer which is *easily* accessible to the ordinary user for any purpose and program of his choice, do we promote the terminal to an intelligent *satellite* (computer). Note that this notation is in conflict with that of Hobbs who calls this last category smart or intelligent terminal, and does not discuss our class of intelligent terminals. Machover's[19] definition tallies more with ours since his intelligent terminal could be constructed purely with nonprogrammable hardware (e.g., the Evans and Sutherland LDS-1 display).

Our view of the idealized intelligent satellite is one which is powerful enough to run a vast number of jobs (e.g., student programs) completely in stand alone mode. In satellite mode it uses the host less than 50 percent of the time, as a fancy "peripheral" which supplies an archival (possibly shared) database, massive computational power (e.g., floating point matrix inversion for the analysis part of the application), and input/output devices such as high speed printers, plotters, microfilm recorders, magnetic tape, etc.

Distributed computing

The term "distributed computing" refers both to devices at remote locations, and logic up to the point of a programmable computer, which has been used to enhance the intelligence of the devices.‡ Such distributed or

* The research described in this paper is supported by the National Science Foundation, Grant GJ-28401X, the Office of Naval Research, Contract N00014-67-A-0191-0023, and the Brown University Division of Applied Mathematics.

** A synonym is the "Computer on a Chip," e.g., the INTEL 8008.

*** Rather than repeating or paraphrasing the several excellent surveys on terminals here, the reader is referred directly to them. Suggested are References 4,6,15 and 19, as well as commercial reports such as those put out by Auerbach and DataPro.

† This is true even if the program can be modified slightly (or replaced in the case of Read Only Memory) to allow such things as variable field definitions on a CRT displayed form, etc.)

‡ Note that the development of ever more sophisticated channels and device controllers started the trend of distributing intelligence away from the CPU. (Programmable) data concentrators and front end processors are further extensions of the principle.

decentralized computing with remote intelligent terminals and especially satellites is a fact of life today. This is so despite the attempts by most computer center administrators in the middle and late 1960's to have individual and department funds pooled to acquire a large, central, omni-purpose computer satisfying everyone's needs. Indeed, the hardware architects,[12] (ACM 72 National Conference panel participants) are predicting even more decentralization, with complete mini or micro computers in homes, offices, and schools, both to provide programmable intelligence to terminals and peripherals, and to serve as local general purpose computers for modest processing tasks. Some of the reasons for this phenomenon are psychological, others technical:

(1) The hardware technology has made it possible; the price of mass produced logic and memory (even of small disks and drums) has decreased dramatically, and more so for mini's than for mainframe hardware; even Foster's "computer on a chip"[12] is a reality before his predicted 1975 date. Consequently, minis (and even midis) have become widely affordable; OEM minis may be part of scientific equipment or of terminals costing under $7,000! This is partially due to the pricing structure of the mini manufacturers who do not tack on mainframe manufacturer's type overhead for such frills as universal software and customer services.

(2) The advantages of distributed (remote) logic and computing are indisputable:
   (a) the convenience and psychological advantage of having a terminal or remote job entry station in or near your office, especially if it can do simple things like accumulate messages and print or edit them locally.
   (b) in the ideal case, the even greater convenience and sense of power rightfully restored to someone who controls the destiny of his very own little, but complete, computer system. No more fighting with the computing center, or contending with other users for scarce resources; presumably less lost data and fewer system crashes since there's no interuser interference within a fragile operating system. (To be fair, if the local machine is then hooked into the central host in satellite mode, the problems of reliability may be worse, due to communications errors for example.)
   (c) The advantage of avoiding extraneous *user effort* by being able to build a message or a picture* *locally*, and immediately verify and edit it, before entering information into the mainframe; this convenience is in contrast to cycles of enter/verify/correct steps which are separated in time.
   (d) The corresponding *conservation of resources*, of both mainframe and communications link, due to fewer and more compact interactions and transmissions; the vastly superior user *response time* (especially given the saturated multi-programmed or time-shared operating systems typical today); and the real or "funny" *money savings* of not unnecessarily using the host.
   (e) The ability to do (significant) work, minimally message composition using simple cassettes, while the host is down or a core partition for the applications program is not available.
   (f) Returning to the transmission link, the advantage of being able to live without its constant availability; incurring fewer errors due to lesser use; being able to be satisfied with a lower speed and therefore less delicate, more widely available and lower cost link.
   (g) The ability, with sufficient intelligence, to emulate existing older devices, to the point of providing with one device "plug for plug compatible" replacements for several existing ones.
   (h) And finally, the enormous advantage of having locally, hopefully general purpose, processing power for arbitrary customization.

An outline of potential problem areas

While distributed logic and computing offer genuinely enhanced capability and cost effectiveness, a number of problems which the user ought to be aware of do crop up.

Hardware problems

(a) Either you choose among the many off-the-shelf software supported, but unchangeable devices, ** *or*
(b) you build yourself, in hardware, or preferably in firmware, a customized terminal just right for your application. You then have to implement your device *and* the supporting software, both probably incompatible with anybody else's gear.

Interfacing problems

(a) If you buy a standard device, connected over a standard (channel or front end) interface, you might be lucky and have no operating system support problems. The non-standard device might need a special purpose interface and might not be recognized (the "foreign attachment" gambit). Even standard interfaces are notorious for crashing operating systems. In any case, "mixed systems" containing multiple vendor hardware are coming of age, but lead to many "our system works, it must

* Let alone scaling and rotating it, doing lightpen tracking to build it, etc.

** Note that if the device does contain a user accessible general purpose computer, inflexibility may be tempered with appropriate software.

be their system" traumas. As an aside, maintenance on the device may not be cheap, so many installations use "on call" arrangements.

(b) To make interfacing easy, obtain flexible remoteness, and avoid the foreign attachment maintenance support problem, telephone communications links are very popular. Modems and lines are expensive for more than 1200 baud transmission, however, and even low speed lines are notoriously noisy (especially in rainy weather)!

Host operating system support

Terminal systems today generally are supported with a host operating system: minicomputers rarely are. Homebrew customized systems, by definition, are not. Providing your own host support at the I/O level for such systems is usually a reasonable task for experienced systems programmers; higher level, truly integrated support, however, may be a real research problem.

Local software support

This type of support ranges from minimal (say a local assembler) to reasonably complete; for a mini it is often a full disk operating system, with FORTRAN and, if you're very lucky, a cross compiler for a higher level procedural language, which compiles on the host, and produces code for the satellite. Even if a standard language like FORTRAN is available on both host and satellite, thereby obviating having to learn to program in two languages, the versions will be incompatible, due to differences in architecture and instruction set of the machines, and in compiler implementation.

Reserving host resources

Making sure that real (or virtual) core and disk space are available for the new device is an often overlooked or underestimated problem.

THE EVOLUTION FROM TERMINAL TO SATELLITE

Despite all the facilities intelligent terminals provide, they still contribute only a relatively small (but possibly quite sufficient) amount of processing power to the total needs of the program or system. For example, display regeneration and housekeeping, communications handling, and simple local interrupt fielding are typical tasks which can be allocated to an intelligent terminal for interactive graphics. Such functions can be lumped together into an area we will call "hardware enhancement." This term is meant to indicate that the basic goal to which the intelligence of the terminal is being directed is the simulation of a more sophisticated piece of hardware. Raising the level of the interface which the terminal presents to the mainframe effectively lightens the load placed on the mainframe by simplifying the requirements of the low ("access method" or "symbiont") level support for the terminal.

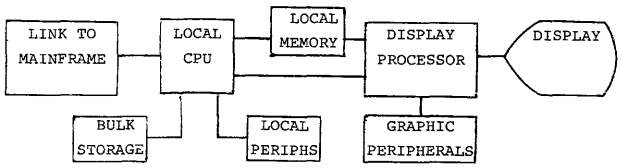


Figure 1

The operational distinction we have made between intelligent terminals and satellites is the use to which the intelligence of the remote terminal system is put. For intelligent terminals this intelligence is *primarily* directed into areas such as hardware enhancement and low level support. On the other hand, the intelligence of a satellite is sufficiently high to allow it to be applied directly to the processing requirements of the application program. (Some hardware implications of this requirement are discussed below.)

The transformation of an intelligent terminal into a full satellite is a long and, as Myer and Sutherland have pointed out[20] somewhat addictive process ("... for just a *little* more money, we could..."). For example, in the graphics case, one first adds a data channel to mainframe core to service the display. Then special display registers and channel commands are incrementally added. A local memory is inserted to free mainframe core. The display is given its own port (i.e., primitive data channel) into the local memory, and the wheel of reincarnation is rolling. The end result of this process is a system of the general form shown in Figure 1.

THE SATELLITE CONFIGURATION

The diagram in Figure 1 says nothing about the power or complexity of the various components. As Foley[11] has pointed out, there are many ways in which the pieces of such a satellite system can be chosen, and the implementation of an optimal (highest cost/performance ratio) configuration for a particular application may entail the examination of hundreds or even thousands of different combinations of subsystems. In the following, we are not as concerned with optimality as with defining a lower bound on the total processing power of the satellite below which it becomes infeasible to view the satellite as a general processor (at least for the purpose of satellite graphics).

The argument for having a satellite system as opposed to an intelligent terminal is to do nontrivial local processing (real-time transformations and clipping, local attention handling, prompting, providing feedback, data-base editing, etc.), while leaving large-scale computation and data-base management to the mainframe. In order to perform satisfactorily for a given (class of) job(s), the satellite must possess "critical intelligence." Analogous to

the "critical mass" concept, critical intelligence defines a threshold of local power which allows the majority of tasks to be executed locally without recourse to the mainframe; it is a complex and application-dependent figure of merit describing such parameters as the architecture and instruction set of the processor, and the primary and secondary storage capacity and access time. It is unfortunately not readily expressed quantitatively, although Foley does try to quantize trade-offs for several classes of applications. Below critical intelligence, i.e., if the satellite does not have (reasonably fast) secondary storage, sufficient local memory, and a powerful instruction set, it simply may not be able to do enough processing fast enough to make the division of labor worthwhile. Many minicomputers used as satellites have too few general-purpose registers and core, inadequate bit and byte manipulation instructions, and minimal operating systems. On such machines, it is seldom possible to handle non-trivial applications programs in *stand-alone* mode satisfactorily (i.e., not just drawing, but handling data structure editing as well), manufacturers' claims notwithstanding.*

In some cases, the shortcomings of a standard minicomputer can be overcome by the use of microprogramming. Microprogrammed processors have a very real advantage over those which are hardwired in that it is frequently possible to redefine a weak architecture or instruction set. Hardware deficiencies such as minimal core may be offset by, for example, an instruction set which has been designed to accommodate often used algorithms. Entire subroutines or their most critical parts may be put in the firmware. An example of the considerable savings in core and execution time which can be achieved with thoughtful firmware instruction set design is described in Reference 2.

Given a satellite system sufficiently powerful to run stand-alone applications, the question arises, "Why go on?" If the satellite has gone once around the wheel and has become self-sufficient, we no longer have a satellite, but another mainframe, and the need for satellite/mainframe interaction disappears. Indeed, this approach was taken, for example, by Applicon, Inc., whose IBM 1130/storage tube system has been carefully and cleverly tailored to a modest application (IC layout), providing one of the few money making instances of computer graphics.[3] ADAGE's[13] more powerful midi systems also were enhanced for example with a fast, high-capacity disk to a point where they support a wide variety of graphic applications without recourse to an even larger computer.

If stand-alone mode is no longer sufficient, the local system may again be enhanced, but in most cases a duplication of facilities with an existing large mainframe is not cost effective, and a crossover point is reached at which communication with the mainframe becomes cheaper than satellite enhancement. In a sense, the satellite is a saddlepoint (an optimal strategy) in the range of configurations bounded at one end by simple display terminals and at the other by full stand-alone graphics processors.

SOFTWARE STRATEGIES AND APPLICATIONS

Given the typical hardware configuration outlined above, it is interesting to examine some of the ways in which such systems have been utilized. These will be presented roughly in order of increasing utilization of the satellite's intelligence.

Hardware enhancement

At the low end of the spectrum, the facilities of the satellite system are used soley to complement capabilities of the display processor. In such systems, the display processor can typically do little more than display graphic data out of a contiguous data list in core. All other display functions—subroutining, transformations, windowing, clipping, etc.—are performed by the satellite processor. Little if any processing power is left over for more application-oriented processing. In fact, there is little difference between a satellite used in this manner and an intelligent terminal; the approach is noted here only because some intelligent terminals may have the hardware structure described in Figure 1.

Device simulation/emulation

Another "non-intelligent" use of a satellite is to emulate and/or simulate another display system. The rationale for this is almost always the presence of a large package of existing software for the display being emulated. Rather than discard many man-years of software development, the choice is made to under-utilize the facilities of the satellite system in order to support the existing applications. Another reason for using the satellite in simulation/emulator mode is that it may be possible to provide higher performance and access to new facilities for existing programs (e.g., control dial and joystick input for a 2250 program). In addition, satellite simulation may allow remote access graphics (over telephone lines or a dedicated link) where not previously possible.**

At Brown, we have implemented three such systems to provide support for IBM 2250 Mod 1 and Mod 3 programs. Two of these were simulators using an IBM 1130/2250 Mod 4[27] and an Interdata Mod 3/ARDS storage

* Yet it is astonishing how many programmers who would not dream of writing a program in 32 kilobytes of 360 user core, with the powerful 360 instruction set and good disks behind it, have the *chutzpah* (typically justly rewarded) to write the same program for a 32-kilobyte mini with a slower, more primitive architecture and instruction set and a painfully slow disk.

** As a commercial example, ADAGE is now offering a display system with simulator software incorporating many of these ideas [ADAGE, 1972].

tube;[26] and one now being completed is an emulator using a Digital Scientific Meta 4 and a Vector General display system. Our experience has indicated that it is indeed useful to be able to continue to run old programs while developing more suitable support for the new system. In addition, we have found that device emulation is a good "benchmark" application for gaining experience with and assessing the capabilities of the new system. On the other hand, in no instance have we found that device emulation made full use of the satellite facilities.

Black box approaches

We are now to the point of considering programs which require the full services of the mainframe/satellite configuration. Such programs are characterized by a need for the graphics capability of the satellite processor and a set of other requirements (computing power, core, bulk secondary storage, etc.) which cannot be completely satisfied by the satellite. In addition, we will assume that the satellite possesses the critical intelligence referred to above.

It is at this point that the "division of labor" problem arises, i.e., determining the optimal way in which the various subtasks of a large application should be allocated to the two processors. This question has as yet received no adequate treatment (a possible approach is outlined below), and it is indicative of the difficulties inherent in the problem that no current system for satellite graphics provides a truly general solution.

The most common treatment of the satellite processor is as a black box. GINO[31] and SPINDLE[16] typify this approach in which all hardware and operating system details of the satellite are hidden from the applications program. The satellite is provided with a relatively fixed run-time environment which performs tasks such as display file management, attention queueing, light-pen tracking, and data-structure management (in conjunction with mainframe routines). Access to these facilities from the application program is usually in the form of a subroutine library callable from a high-level language (e.g., FORTRAN).

This approach has the attractive feature of "protecting" the applications programmer from getting involved in multiple languages, operating system details, and hardware peculiarities. In addition, a well-designed system of this type can be easily reconfigured to support a new satellite system without impacting the application program. On the other hand, defining a fixed satellite/mainframe task allocation may incur unnecessary systems overhead if the allocation should prove to be inappropriate for a particular application. Particularly in the case of high-powered satellites, it may be difficult to provide access to all the facilities of the satellite without requiring the applications programmer to "get his hands dirty" by fiddling with various pieces of the system. Worst is poor use (inadequate use) of local facilities, wasting power and incurring charges on the mainframe.

Systems for interconnected processing

While the black-box approach hides the satellite from the programmer, interconnected processor (ICP) systems (connecting one or more small satellites to a large host) allow (and sometimes require) cognizance of both the mainframe and satellite processors. At the lowest level, such a "system" consists of no more than a communications package or access method. At a slightly higher level are packages such as IBM's processor-to-processor (PTOP) routines for 360/1130 communications.[22] These routines provide a high-level communication interface together with data conversion capabilities.

More sophisticated systems are exemplified by Bell Labs' GRIN-2[7] and UNIVAC's Interactive Control Table (ICT) approach.[8] In these systems, a special-purpose language is provided with which the application programmer specifies the detailed data structure manipulation and/or attention handling which is to take place during an interactive session. Once this specification has been made, it becomes part of the system environment of *both* processors. The Univac system allows this environment to be changed at runtime by providing for the dynamic loading of new satellite programs for full attention handling and data structure manipulation. Thus the programmer has at least some control over the activities of the satellite processor.

A very general system of this type has been outlined by Ross et al.,[24] in which a Display Interface System (DIS) is described. The DIS consists of "minimal executives" in both the satellite and mainframe processors. These executives act in conjunction to provide attention and program-handling mechanisms in both machines. Additional features, such as satellite storage management and display file handling, are available in the form of system-provided routines which can be allocated to either processor. Effectively, the DIS provides a "meta-system" which can be used by a systems programmer to tailor the appearance of the mainframe/satellite interface to provide optimal utilization of the satellite configuration. While a full DIS system has not yet been implemented, the basic design principles were used with apparently good success in the design of the GINO package.[31]

What objections to the preceding systems can be raised? The Univac system requires bi-linguality and the overhead of a local interpreter, a deficiency recognized by the implementers.[9] This particular system also failed to achieve critical intelligence since the hardware on which it was implemented lacked mass storage, general purpose registers, and a decent instruction set. The Grin-2 experiment is somewhat more ambiguous, with in-house users apparently satisfied and some outsiders dissatisfied, with such features as a fixed (ring) data structure, an unwieldy programming language, not enough local core, etc. The GINO satellite system, though used very successfully, has had only relatively minor housekeeping and transformation functions executed locally, thereby not saving very much on host resources in this intelligent

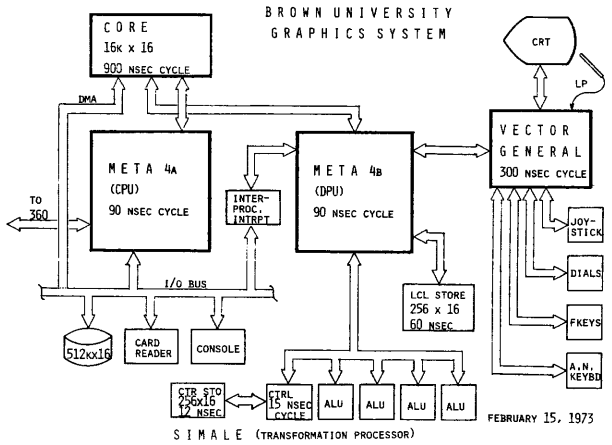


Figure 2

terminal mode. Thus, a truly general purpose, flexible, easy to use, and cost effective system for host-satellite communication is yet to be achieved.

A SYSTEM FOR STUDYING SATELLITE GRAPHICS

The Brown University graphics system (BUGS)

For the past eighteen months the Graphics Project at Brown University has been implementing a laboratory system for the investigation of a variety of questions relating to satellite graphics. The salient features of the system (shown in Figure 2)* are as follows.**

The local processor: The general-purpose processor of the system is a microprogrammable Digital Scientific Meta 4. It has been provided with a 360-like firmware defined instruction set with additions and modifications to enhance the ability of the processor to meet the needs of the operating system and graphics applications.

The display processor: A second Meta 4 was chosen to serve as a programmable display processor to drive the Vector General display. While the Vector General itself is a relatively powerful processor, this "level of indirectness" was added to allow complete freedom in designing (and altering) the display instruction set seen by the user.

The SIMALE—It was determined that even with the high speed of the Meta 4, it would not be possible to provide full three-dimensional transformations (with windowing and clipping) at a speed sufficient to display 1000-2000 vectors. For this reason, we have designed the SIMALE (Super Integral Microprogrammed Arithmetic Logic Expediter) to perform homogeneous transformations, windowing, and clipping. SIMALE is a high-speed parallel processor with writeable control store! Webber, 1973!.

The 360 Interface—The communications link to the mainframe (an IBM 360/67 under CP67/CMS) is a multiplexer channel interface. The interface, which is driven from firmware, is non-specific, that is, it can appear as any IBM-supported device. Eventually, this link will be downgraded to a medium to low speed (e.g., 1200 BAUD) communications line.

The local operating system—The operating system which runs on the local processor was built using the "extended machine" or "structured" approach typified by Dijkstra's THE System.[10] With this approach, the operating system is developed as a series of distinct levels, each level providing a more intelligent "host" machine to the next level. The design of the system facilities the vertical movement of various facilities between levels as experience dictates. As facilities become stabilized on the lowest level, they can be moved into the firmware with no impact on user programs.

An environment for interconnected processing

The system outlined above, which admittedly has gone around the wheel of reincarnation several times, has been designed with the goal of keeping each subsystem as open-ended as possible, thus allowing maximum flexibility in subsystem/task allocation. This approach is also being taken in the design of system software to support graphics applications requiring both satellite and mainframe facilities.

With currently extant systems, the ramifications of splitting an application between the mainframe and the satellite are many and frequently ugly. Minimally, the programmer must become acquainted with a new instruction set or an unfamiliar implementation of a higher-level language. In addition, the vagaries of the satellite operating system must be painfully gleaned from Those-in-the-Know. With luck, there will be some sort of support for I/O to the mainframe, but probably no good guidelines on how best it should be used. Most importantly, the programmer will have little or no knowledge about how to split his application between the two processors in such a way as to make optimal use of each. In other words, mistakes are bound to occur, mistakes of the kind which frequently require a significant amount of recoding and/or redesign.

The basic goal of the ICP system outlined below is to alleviate these problems *while* placing minimal constraints on the programmer's use of the two processors. The aim is to provide not merely a high-level access method or system environment through which the satellite can be referenced, but rather a set of tools which will allow the programmer to subdivide an applications program or system between the satellite and the mainframe without constant reference to the fact that he is working

* A PDP 11/45 with a Vector General display is a cheaper commercial system sharing many of the characteristics of the BUGS system. We are providing a FORTRAN based graphics subroutine package for this system, both for standalone and for 360/370 satellite mode.

** For more complete information about the system, see References 2 and 28

with two dissimilar processors. These tools include the following:*

- A completely transparent I/O interface between the mainframe and the satellite. I/O between the two processors should take place at a purely logical level with no consideration of communications protocol, interface characteristics, or timing dependencies.
- A run-time environment to support inter-process communication as described below. In the limit, this environment should be sufficiently powerful to allow dynamic (run time) redefinition of the task/processor allocation.
- A high-level language with translators capable of generating code for both machines. Minimally, this language should let the programmer disregard as far as possible differences in the hardware and operating system defined characteristics of the two processors.** Optimally, the language should provide constructs to maximize the ease with which the various tasks of a large application can be moved from one processor to the other.

The language for systems development

Of the tools mentioned above, the most important is the high-level language in which the application is going to be written. Indeed, the language is the heart of our approach to ICPing since it provides the uniform environment in which the programmer can work with little or no regard for the final task/processor subdivision of the application. If he wishes, the language will also let him hand-tool each routine for the processor on which it will run.

The language which we are using for both the ICP system and application programs is the Language for Systems Development (LSD).[5] LSD is a general-purpose procedure-oriented language with many of the features and much of the syntax of PL/I. In contrast to PL/I, however, the language enables the programmer to get as close as he wants to the machine for which he is programming rather than hiding that machine from him. Thus, while the ordinary applications programmer can simply use it as a FORTRAN replacement, a systems programmer can explicitly perform operations on main memory locations and registers; he can intersperse LSD code with assembly language or machine language (through the CODE/ENCODE construct). LSD also will provide a variety of extension mechanisms to permit the user to tailor the language to specific problems or programming styles. Some of the features of LSD which make it an ideal vehicle for the ICP system are the following:

- The ability to completely redefine the compile-time code generators. This allows implementation of a compiler which will generate code for either the satellite or the mainframe.
- Extensibility mechanisms. In particular, semantic extensibility allows the definition of new data types, storage classes, and scopes for variables.
- The ON ACCESS mechanism. This facility, which is similar to PL/I's CHECK statement, allows compile-time definition of routines which are to be invoked whenever a particular variable is accessed at run time.
- Operating system independence. The constructs which are generated by the compiler have been designed to be as independent as possible of the operating system under which they are to run. In addition, the run-time environment required by an LSD program has been kept as small as possible.
- Run-time symbol table access. Complete information about the scope, type and storage class of all variables is available at run time.

LSD extended for the ICP system

The fundamental extension to be made to LSD for ICP'ing is the addition of a new scope for variables and procedures. Currently, an LSD variable may have one of three scope attributes—LOCAL, GLOBAL, or EXTERNAL. A LOCAL variable is accessible only within the procedure in which it is allocated. A GLOBAL variable may be known to all procedures in the system. EXTERNAL indicates that the variable has been defined as a GLOBAL in some other procedure which is external to the current procedure. A procedure can be either external or internal. An internal procedure is defined within another procedure. An external procedure is the basic unit of programming within the system; that is, it can be compiled separately from all other procedures with no loss of information.

For the use of ICP applications, a new scope will be defined which will be referred to here as ICPABLE. A declaration of ICPABLE defines the named variable or procedure as one which *may* reside in the other processor.† This declaration will force the compiler to take the following actions:

- On variable access or assignment, a run-time routine must be called which has the task of returning the value (or address) of the variable, possibly accessing the other processor to obtain the current value of the variable.
- On a call of a procedure which has been declared ICPABLE, a similar check must be made as to the current whereabouts of the procedure. If the proce-

* The approach taken is similar to that of J. D. Foley in his current National Science Foundation sponsored research in computer graphics; the envisioned run-time environments and facilities, however, are quite dissimilar, as will be described below.

** Note that microprogramming is a very handy implementation tool for making the satellite architecture and instruction set somewhat similar to that of the host, thereby reducing the load on the compiler design team.

† This definition is similar to Foley's GLOBAL; however, assignments between processors are not run-time alterable in Foley's system, a significant and far reaching difference.

dure is in the other processor, an appropriate mechanism for passing of control and parameters must be invoked.

The end effect of this extension will be that the programmer need have only very vague ideas about the eventual disposition of his programs and data. During program development, any unclear areas can be resolved by declaring all affected variables and routines as ICPABLE. If the referenced object is in the same processor as the "referencor," overhead will be minimal; if it is not, overhead beyond the necessary communications delay will hopefully still be minimal.* Once the application has been shaken down, this minimal overhead can be removed by suitable redeclarations of the ICPABLE variables and procedures.

The run-time environment

As currently envisioned an application requiring satellite graphics will run in an environment consisting of five levels.

- At the lowest level will be a set of routines (in each processor) which handle the lowest level physical I/O. A standard interface will be defined between these routines and higher levels to ensure flexibility with respect to the variety of possible satellite/mainframe links.
- Above the low level I/O package will be an (LSD callable) access method for explicit use by the LSD programmer as well as the routines supporting implicit inter-process communication. Also at this level will be any system supplied routines which are found necessary to interface with the lowest level facilities on the two processors (e.g., a routine to actually start display regeneration).
- The access method will be used for the most part by a third level of routines in charge of performing all necessary data transmission and conversion.
- Between the data conversion routines and the actual LSD program will be a supervisory package which keeps track of the current procedure/variable/processor assignment. When dynamic movement of variables and procedures between processors becomes feasible, it also will be undertaken at this level.
- The highest level of the run-time environment will consist of a "meta-system" which is used for system resource utilization, response measurements, and dynamic reconfiguring of the application program. The idea here is to provide a logical "joystick" with which the programmer (or user) can make real-time decisions as to the optimal deployment of the various pieces of the application. By moving the stick in the "360 direction" he causes some of the modules to be loaded and executed in that machine; by moving in "the other direction," he causes modules to be shifted to the satellite. Observing response or some graphically displayed resource utilization and cost data, he can manipulate the stick, trying for a (temporal) local optimum.

A hypothetical example

In order to illustrate a use of the system described above, we offer for consideration a piece of a larger application consisting of four procedures—DISKIO, DSUPDATE, BUFFGEN, and ATTNWAIT—together with a MAINLINE representing the rest of the application. DISKIO is a routine which handles I/O to bulk storage on the mainframe; DSUPDATE is in charge of modifying and updating the graphic data structure; BUFFGEN generates a display file from the data structure; and ATTNWAIT processes attentions from the graphic input devices.

While the programmer is writing these routines, he disregards the eventual destinations of these routines, and programs as if the system were to appear as:**

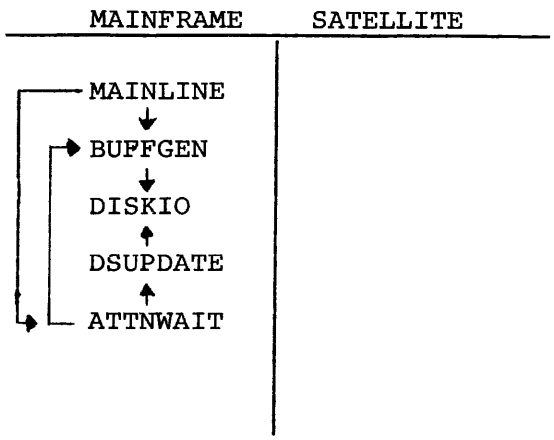


Figure 3

However this disregard while implementing does not mean that the programmer is unaware of the fact that he is indeed working with two processors, and that at some point certain processor/task assignments are going to be made. While he is reasonably certain that DISKIO is going to run in the mainframe and ATTNWAIT in the satellite, he is less sure about BUFFGEN and DSUPDATE and therefore declares these routines ICPABLE. He then (in effect) tells the compiler, "Compile DISKIO for the mainframe, ATTNWAIT for the satellite, and DSUPDATE and BUFFGEN for both processors."

When the system is ready for testing, he invokes the highest level of the run-time environment and requests

* The overhead inflicted by various flavors of special purpose run-time environments is notoriously unpredictable; the "minimal" overhead for ICPABLE variables and procedures could prove to be entirely unacceptable.

** Arrows represent calls; system routines have been omitted for clarity.

his program be run with a trial allocation of tasks as follows:

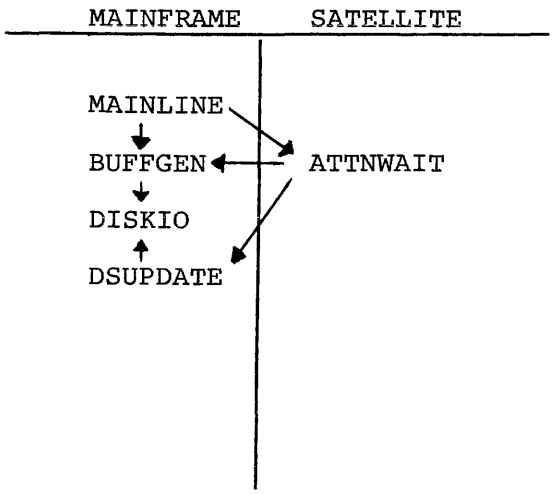


Figure 4

After running with this allocation for a while, the programmer reenters the "meta-system" level to see what kind of statistics have been gathered during the session. The statistics indicate that during those times when some demand was being made on the total system (i.e., periods during which the total application was not quiescent while waiting for a user action), the satellite processor was relatively idle. The programmer therefore decides to try a reallocation of tasks by moving the BUFFGEN routine into that satellite, resulting in:

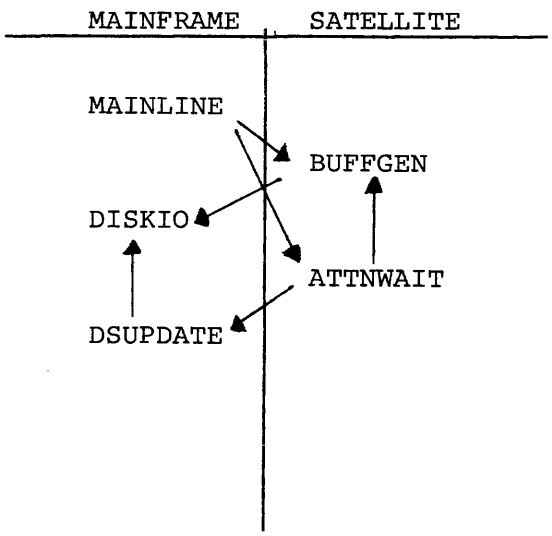


Figure 5

This configuration improves the utilization of the satellite, and the programmer returns to the application environment for further experimentation.

After a while, response time degrades drastically. On investigation it is found that two systems and four PL/I compiles are currently running in the mainframe. To make the best of a bad situation, the programmer moves the DSUPDATE routine into the satellite:

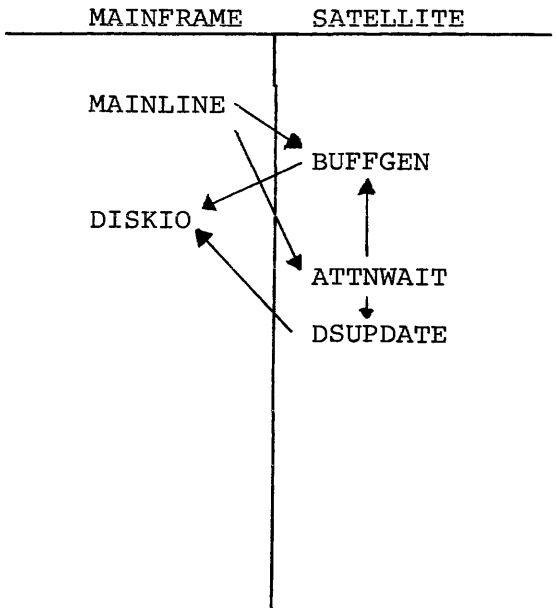


Figure 6

While the satellite may now be overloaded (e.g., overlays may be needed), response time is still improved since less work is being done in the currently "crunched" mainframe.

Hopefully this example gives some idea of the power we envision for the full ICP system. Obviously this description has glossed over many of the difficulties and implementation problems which we will encounter. In particular, the problem of data base allocation and the transporting of pieces of a data structure between the two processors with different machine defined word sizes presents a formidable problem. Since we do not want to "inflict" a built-in data structure facility on the user it may become necessary to require of him more than minimal cognizance of the two processors for problems involving data structure segmentation.

CONCLUSIONS

Decentralization and distribution of computing power is coming of age, and is expected to be the standard of the future, with perhaps several levels and layers of hosts and satellites, embedded in a network. Special-purpose-application dedicated intelligent terminals are already proliferating because they are cost effective and their hardware/firmware/software design is straightforward. Intelligent satellite computing, on the other hand, is still in its infancy, especially in its full generality where there is a genuine and changeable division of labor between host and satellite. Few design rules of thumb exist beyond the "buy-low/sell-high" variety (for example, interactions on the satellite, number-crunching and data base manage-

ment on the host). Even fewer *tools* exist for studying varying solutions, and their implementation is a far from trivial task.

We hope to be able to report some concrete results of our investigation into this important problem area in the near future.

REFERENCES AND SELECTED BIBLIOGRAPHY

1. ADAGE Corp., Technical Description of ADAGE Interactive Graphics System/370, News release, March 1972.
2. Anagostopoulos, P., Sockut, G., Stabler, G. M., van Dam, A., Michel, M., "Computer Architecture and Instruction Set Design, *NCCE*, June 4, 1973.
3. *The Design Assistant*, Applicon Corp., News Release, 1972.
4. Bairstow, J. N., "The Terminal that Thinks for Itself," *Computer Decisions*, January 1973, 10-13.
5. Bergeron, D., Gannon, J., Shecter, D., Tompa, F., van Dam, A., "Systems Programming Languages," *Advances in Computers*, 12, Academic Press, October 1972.
6. Bryden, J. E., "Visual Displays for Computers," *Computer Design*, October 1971, pp. 55-79.
7. Christensen, C., Pinson, E. N., "Multi-Function Graphics for a Large Computer System," *FJCC* 31, 1967, pp. 697-712.
8. Cotton, I. W., Greatorex, F. S., Jr., "Data Structures and Techniques for Remote Computer Graphics," *FJCC* 33, 1968, pp. 533-544.
9. Cotton, I. W., "Languages for Graphic Attention-Handling," *International Symposium on Computer Graphics*, Brunel, April 1970.
10. Dijkstra, E. W., "The Structure of THE Multiprogramming System," *CACM*, 11, 1968, pp. 341-346.
11. Foley, J. D., "An Approach to the Optimum Design of Computer Architecture," *CACM*, 14, 1971, pp. 380-390.
12. Foster, C., "A View of Computer Architecture," *CACM*, 15, 1972, pp. 557-565.
13. Hagan, T. G., Nixon, R. J., Schaefer, L. J., "The Adage Graphics Terminal," *FJCC*, 33, 1968, pp. 747-755.
14. Hobbs, L. C., "The Rationale for Smart Terminals," *Computer*, November-December 1971, pp. 33-35.
15. Hobbs, L. C., "Terminals," *Proc. IEEE*, 60, 1972, pp. 1273-1284.
16. Kilgour, A. C., "The Evolution of a Graphics System for Linked Computers," *Software—Practice and Experience*, 1, 1971, pp. 259-268.
17. Knudson, D., and Vezza, A., "Remote Computer Display Terminals," *Computer Handling of Graphical Information*, Seminar, Society of Photographic Scientists and Engineers, Washington, D.C., 1970, pp. 249-268.
18. Machover, C., "Computer Graphics Terminals—A Backward Look," *SJCC*, 1972, pp. 439-446.
19. Machover C., "The Intelligent Terminal, *Pertinent Concepts in Computer Graphics," Proc. of the Second University of Illinois Conference on Computer Graphics*, M. Faiman and J. Nievergelt (eds.), University of Illinois Press, Urbana, 1969, pp. 179-199.
20. Myer, T. H., Sutherland, I. E., "On the Design of Display Processors, *CACM* 11, 1968, p. 410.
21. Ninke, W. H., "A Satellite Display Console System for a Multi-Access Central Computer," *Proc. IFIPS*, 1968, E65-E71.
22. Rapkin, M. D., Abu-Gheida, O. M., "Stand-Alone/Remote Graphic System," *FJCC*, 33, 1968, pp. 731-746.
23. Rosenthal, C. W., "Increasing Capabilities in Interactive Computer Graphics Terminals," *Computer*, November-December 1972, pp. 48-53.
24. Ross, D. T., Stotz, R. H., Thornhill, D. E., Lang, C. A., "The Design and Programming of a Display Interface System Integrating Multi-Access and Satellite Computers," *Proc. ACM/SHARE 4th Annual Design Workshop*, June 1967, Los Angeles.
25. Ryden, K. H., Newton, C. M., "Graphics Software for Remote Terminals and Their Use in Radiation Treatment Planning," *SJCC*, 1972, pp. 1145-1156.
26. Schiller, W. L., Abraham, R. L., Fox, R. M., van Dam., A., "A Microprogrammed Intelligent Graphics Terminal," *IEEE Trans. Computers* C-20, July 1971, pp. 775-782.
27. Stabler, G. M., *A 2250 Model 1 Simulation Support Package*, IBM PID No. 1130-03.4.014, 1969.
28. Stabler, G. M., *The Brown University Graphics System*, Brown University, Providence, R. I., February, 1973.
29. van Dam, A., "Microprogramming for Computer Graphics," *Annual SEAS Conference*, Pisa, Italy, September 1971.
30. Webber, H., "The Super Integral Microprogrammed Arithmetic Logic Expediter (SIMALE)," *SIGMICRO*, 6, 4, 1972.
31. Woodsford, P. A., "The Design and Implementation of the GINO 3D Graphics Software Package." *Software—Practice and Experience*, 1, 1972, pp. 335-365.

Fourth generation data management systems

by KEVIN M. WHITNEY

General Motors Research Laboratories
Warren, Michigan

INTRODUCTION AND HISTORY

Many hundreds of programming systems have been developed in recent years to aid programmers in the management of large amounts of data. Some trends in the development of these data management systems are followed in this paper and combined with ideas now being studied to predict the architecture of the next generation of data management systems. The evolution of data management facilities can be grouped into several generations with fuzzy boundaries. Generation zero was the era when each programmer wrote all his own input, output, and data manipulation facilities. A new generation of facilities occurred with the use of standard access methods and standard input/output conversion routines for all programs at an installation or on a particular computer system. The second generation of data management was the development of file manipulation and report writing systems such as RPG, EASYTRIEVE, and MARK IV. Much more comprehensive facilities for the creation, updating, and accessing of large structures of files are included in the third generation of generalized data management systems such as IMS/2, IDS, and the CODASYL specifications. Each of these generations of data management systems marked great increases in system flexibility, generality, modularity, and usability. Before speculating on the future of data management, let us survey this history in more detail.

Standard access methods, such as ISAM, BDAM and SAM which formed the first generation data management facilities were mainly incorporated in programming languages and merely gathered some of the commonly performed data manipulation facilities into standard packages for more convenient use. The main benefit of these standard facilities was to relieve each application programmer of the burden of recoding common tasks. This standardization also reduced program dependency on actual stored data structures and formats. Although these access methods and input/output conversion routines were often convenient to use, they could only accommodate a limited number of different data structures and data types.

As computer systems became more common, more widely used, more highly depended on, and eventually more essential to the functioning of many businesses, a second generation of data management facilities became available to speed the job of generating reports and writing new application packages. These report generators, file maintenance packages, and inquiry systems provided easy-to-use facilities for selecting, summarizing, sorting, editing, and reporting data from files with a variety of formats. A user language simple enough for non-programmers to use was sometimes provided, although using more advanced system facilities often required some programming aptitude. This second generation of data management systems brought non-procedural user languages and a greater degree of program independence from data formats and types.

Much more general capabilities were incorporated into third generation systems such as IDS, IMS, and the CODASYL report specifications.[1] All of these systems emphasize the effective management of large amounts of data rather than the manipulation or retrieval of data items. All have facilities to manage data organized in much more complicated data structures than the sequential structures used by earlier systems. Relationships among data items in many different files are expressible and manipulable in the data management systems. Provisions for the recovery from many different types of system errors, head crashes, erroneous updates, etc. are integral parts of these systems. Audit trails of modifications of the data base are often automatically maintained, and new file descriptions are handled by standard system facilities. In general, many of the functions which previous generations of data management systems left to the operating system or to user application programs are automatically incorporated in these systems. Many of the third generation systems provide a convenient user inquiry language, a well-defined interface for user application programs, and new language facilities to aid the data administrator in the description and maintenance of the data base.

A fourth generation data management system will continue the development of all these trends toward generality, flexibility, and modularity. Other improvements will result from theories and concepts now being tried in experimental systems. Much greater degrees of data independence (from user program changes, from data description and storage changes, from new relationships among the data) will be common; user languages will

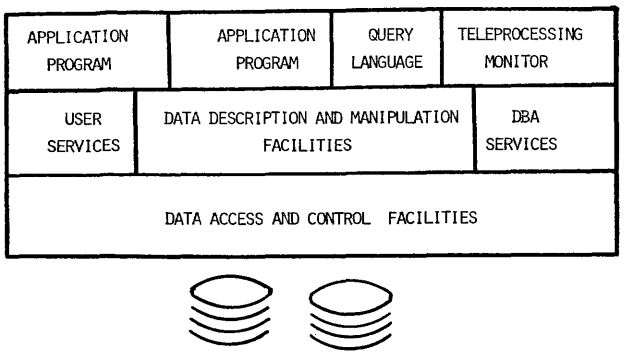


Figure 1—Information management system structure.

become much less procedural; and data manipulation facilities for use in writing application programs will become simpler and more powerful. Concepts from set theory and relation theory will become more widely used as the advantages of a sound theoretical basis for information systems become more widely appreciated. Increasingly, information management systems will make more of the optimization decisions relating to file organization and compromises between different user requirements. The trend to bending the computer toward user requirements rather than bending the user to the requirements of the computer will continue resulting in progressively easier to use systems.

One organization of the facilities of an information management system may be the division into modules shown in Figure 1. This modularity may represent software or hardware (or both) boundaries and interfaces. A data access and control module is needed to manage data flow to and from the storage media. This data management module is used by the data description and manipulation module in providing data description and manipulation at a level less dependent on the storage structure of this data. The user can access data through application processors or through a system provided query language processor. A variety of system services are grouped into the user services module and the data base administrator services module. User services include such conveniences as on-line manuals, help and explain commands, and command audit trails. Data base administrator services include facilities to load and dump data bases, to perform restructuring of data and storage organizations, to monitoring performance, and to control checkpointing and recovery from errors.

Impacting the fourth generation information management systems are the theories and methodologies of data description and manipulation, the relational view of information, the establishment of sound theoretical foundations for information systems, and the development of networks of cooperating processors. Each of these topics will be discussed in one of the following sections. Following those sections is a description of our experiences with RDMS, a relational data management system which illustrates some of these new theories and methodologies.

DATA DESCRIPTION AND MANIPULATION

Certainly the most complex and difficult problem facing the designers, implementers, and users of an information management system is the selection of language facilities for the description and manipulation of data. Although many attempts have been made to separate data description from data manipulation, it must be noted that data description and data manipulation are inextricably intertwined. While the declarative statements which describe a data base may indeed be kept separate from the statements in application programs which manipulate the data in the data base, nevertheless the data description facilities available determine and are determined by the data manipulation facilities available. Descriptive statements for vectors aren't very useful without vector manipulation statements, and vice versa.

The description of data may be done at a wide variety of levels of generality ranging from general statements about the relationships between large sets of data items to explicit details about the actual storage of the data items. Information management systems of the next generation will have data description facilities at a variety of levels to serve different classes of users. At least three main levels of data description can be distinguished, the information structure for the users, the data structure for the data base administrators, and the storage structure for the system implementers and the system.

The *information structure* which determines the user's view of the data is (ideally) quite abstract, indicating the relationships among various types of data items, but omitting details such as data item types, precisions, field lengths, encodings, or storage locations. The user should also be free of system performance considerations such as indexing schemes or file organizations for efficient retrieval of the data items, particularly since his access requirements may conflict with those of other users. In any event the system or the data administrator is in a better position than any one user to make those arbitrations. An example of the information structure for some data described by a relational model is shown in Figure 2.

A second level of data description is necessary to specify the structure (logical) of the data which will facilitate the efficient retrieval of the data representing the information in the data base. This second level of description is the *data structure* and represents additional informa-

```
PEOPLE (NAME, AGE, HEIGHT, WEIGHT)

SHIRTS (SHIRT#, COLOR, SIZE, COLLAR)

SLACKS (SLACKS#, COLOR, SIZE, FABRIC)

OWNS-SHIRTS (NAME, SHIRT#)

OWNS-SLACKS (NAME, SLACKS#)
```

Figure 2—An information structure specified by a relational model

tion about the patterns of retrieval expected for information in the data base. The set occurrence selection facility of the CODASYL proposal is an example of this level of data description. CODASYL schema data description statements for the information structure of Figure 2 are shown in Figure 3.

A still more detailed level of data description is the *storage structure* of the data which represents the actual organization of the data as stored in the system's storage media. At the storage structure level, items of concern include field types, lengths, encodings, relationship pointers and index organizations. Figure 4 shows a diagram of pointers and storage blocks specifying a storage structure for the data described in Figure 3.

Data manipulation facilities must also exist at a variety of levels corresponding to the data description levels. As data description becomes more general, the corresponding data manipulation becomes less procedural and more descriptive. E. F. Codd's relational model of data[2] described in the next section provides an example of a highly non-procedural data manipulation language.

A RELATIONAL MODEL FOR INFORMATION

Information management systems deal fundamentally with things such as people, ages, weights, colors, and sizes which are represented in a computer by integers, floating point numbers, and character strings. A collection of items of similar type is called a domain. Domains may overlap, as for example with the domains of people and of children.

```
SET IS PEOPLE; OWNER IS SYSTEM.
    MEMBER IS PERSON  DUPLICATES NOT ALLOWED FOR NAME.

RECORD IS PERSON.
    NAME          TYPE CHARACTER 12.
    AGE           TYPE FIXED; CHECK RANGE 5, 50.
    HEIGHT        TYPE FIXED.
    WEIGHT        TYPE FIXED.
    #SHIRTS       TYPE FIXED.
    #SLACKS       TYPE FIXED.
    SHIRTS        OCCURRS #SHIRTS.
       2 COLOR    TYPE BIT 4; ENCODING CTABLE.
       2 SIZE     PICTURE 99.
       2 COLLAR   PICTURE 99.
    SLACKS        OCCURRS #SLACKS.
       2 COLOR    TYPE BIT 4; ENCODING CTABLE.
       2 SIZE     PICTURE 99.
       2 FABRIC   PICTURE 99.
```

Figure 3—A data structure specified in the CODASYL data description language

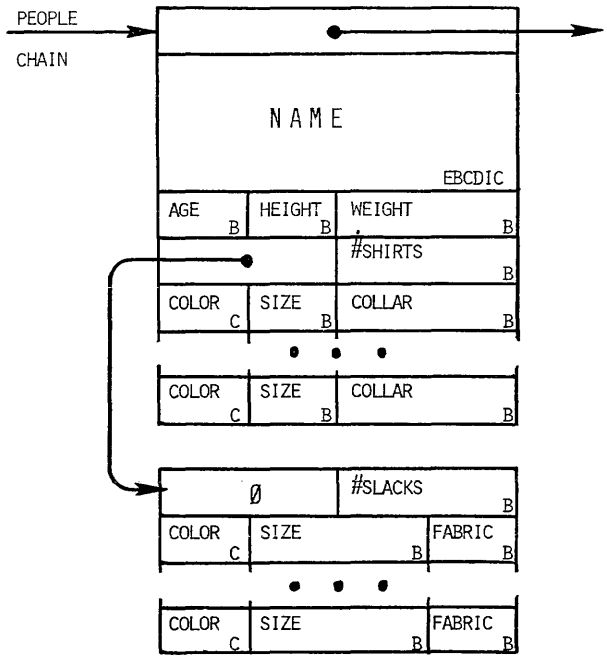


Figure 4—A storage structure specified by a pointer diagram

A relationship is an association among one or more not necessarily distinct domains. "Is married to" is a relationship associating the domains of men and of women. An occurrence of a relationship is an ordered collection of items, one from each domain of the relationship, which satisfy the relationship. Each occurrence of a relationship associating N domains is an ordered collection (John and Mary is an occurrence of the relationship "is married to" if John is married to Mary).

A relation is a set of some tuples satisfying a relationship. The number of domains of a relation is called its degree and the number of tuples of a relation is called its cardinality.

This simple structure is adequate to represent a great variety of information structures. Certain data manipulation facilities arise naturally from the relational description of information. The basic operations are the traditional operations of set theory (union, intersection, difference, etc.) and some new operations on sets of relations (projection, join, selection, etc.). Projection reduces a relation to a subset of its domains, while join creates a relation by combining two component relations which have one or more common domains. Selection extracts the subset of the tuples of a relation which satisfy some restriction on the values of items in a particular domain. In combination with the usual control and editing commands, these operations provide a convenient and non-procedural data manipulation language. Note that nothing need be known about the data or storage structures of the information represented by a relational model. The user may concern himself entirely with the domains of interesting objects and their relationships.

THEORETICAL FOUNDATIONS FOR INFORMATION SYSTEMS

As information systems become increasingly complicated, it will be more important to base their designs on sound theoretical foundations. Although it has always been customary to test a system as comprehensively as possible, larger systems can never completely be tested. Thus it is important to find other methods of verifying that a system will function as specified. Current work in proving program correctness has this same aim with regard to programs. In this section three contributions to the theoretical foundations of information system structure will illustrate some possible effects of new theory on system design. D. L. Childs[3] has devised an ordering for set structures useful in the implementation of set operations. E. F. Codd[4] has developed the relational calculus as a sound basis for user languages, and W. T. Hardgrave[6] has proposed a method for eliminating ambiguous responses to queries of hierarchical data structures.

Childs proposed a general set theoretic data structure based on a recursive definition of sets and relations. His theory guarantees that it is always possible to assign unique key values to any set element in such a structure. These key values may be used to create an ordered representation of the data structure. Set operations are much more efficient on ordered sets than on unordered sets. Thus the theory leads to efficient implementations of complex structures.

In a series of papers, E. F. Codd has developed the relational model of data which was explained in the previous section. One important theoretical result of this theory is a proof that any relational information in a data base of relations can be retrieved by a sequence of the basic relational and set operations defined in the previous section of this paper. Furthermore, it is possible to estimate the system resources necessary to answer the query without answering it. Thus a user can be warned if he asks a particularly difficult query. Although not all queries on a data base of relations can be answered as relations, the inclusion of functions on relations (counts, sums, averages, etc.) guarantee a very wide range of legal queries. This theory assures a system design that a general purpose system will not suddenly fail when someone asks an unexpected new query.

Research into storage structures underlying the relational data model by Date & Hopewell[5] show that a variety of possible anomalies in the storage, updating, and retrieval of information do not arise when the information is stored in Codd's canonical third normal form. Thus by using a relational storage structure for data, certain types of consistency are automatically assured. These studies show also a variety of efficient methods of implementing a relational data model.

A third investigation into the fundamentals of information systems design is W. T. Hardgrave's study of information retrieval from tree structured data bases with Boolean logical query languages. Hardgrave analyzed anomalies resulting from the use of the "not" qualifier in boolean queries. Finding unavoidable problems with the usual set theoretic retrieval methods, he formulated additional tree operations which separate the selection of data items from the qualification of items for presentation. These capabilities may be considered a generalization of the HAS clause of the TDMS system. This study not only focuses attention on possible multiple interpretations for some Boolean requests applied to items at different levels of a tree hierarchy, but also presents more severe warnings about possible problems with the interpretation of network structured data such as in the CODASYL proposal.

NETWORKS OF COOPERATING PROCESSORS

Continuing decreases in the cost of data processing and storage equipment and increases in the cost of data manipulation software will bring further changes in the architecture of information management systems. The example of Figure 5 shows the natural trend from soft-

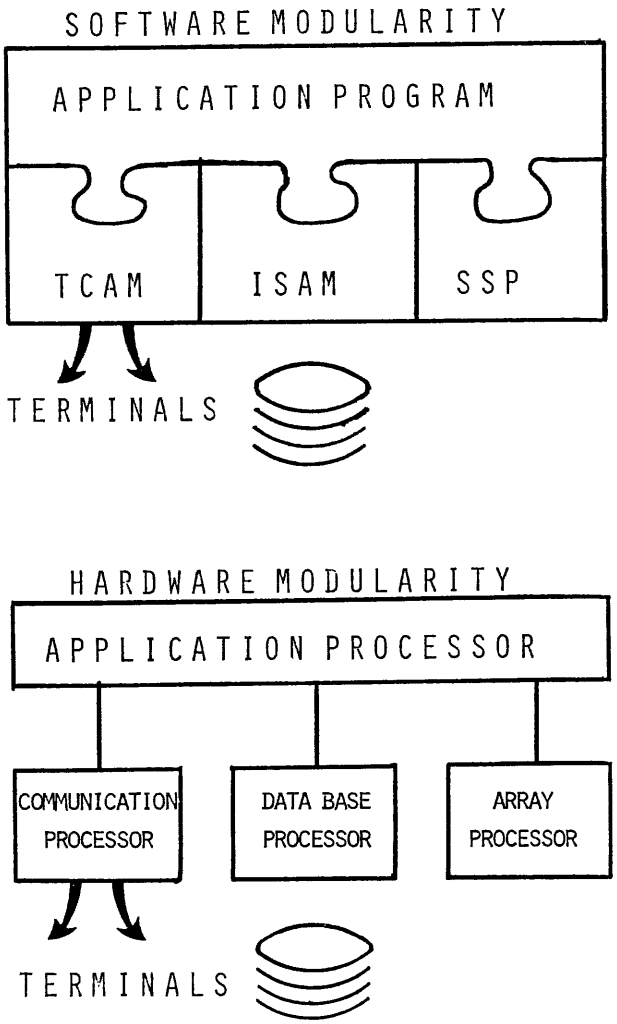


Figure 5—The trend from software modularity toward hardware modularity

ware modularity of third generation systems to hardware modularity of fourth generation systems.

This trend toward hardware modularity has been under way for many years. Control Data Corporation has advocated the use of independent peripheral processors to handle some of the more mundane functions of computing systems such as input/output spooling, paging, disk and drum interfaces, etc. The use of front end processors such as the IBM 3705 to handle communications functions independently of the central processor is already common. IBM uses the Integrated Storage Controller, a small independent processor, to control 3330 disk drives. Special purpose computer systems for Fourier transforms and for matrix manipulation are being used as peripheral or attached processors in current computing systems.

Two specific examples of independent processors in data management systems are intelligent remote inquiry terminals and the data base computer. While intelligent remote terminals are already common, independent data base computers are still in research laboratories. These data base machines consist of an independent control processor and a large storage media. The processor not only manages the control commands necessary to handle the storage media, but also can perform logical extraction or selection of data records to reduce the amount of data transmitted to the host processor. As more experience with independent data base computers is accumulated, they will assume additional tasks such as selecting data compaction and compression methods for optimal data storage and selecting indexing methods for optimal data access.

RDMS, A RELATIONAL DATA MANAGEMENT SYSTEM

To gain some experience with the advantages and limitations of data management using a relational model of information, the RDMS system was designed and implemented in PL/I on a large virtual memory computer system. The system consists of three main sections, a command (query and data manipulation) language interpreter, set and relational data manipulation facilities, and an interface to the operating system. This modular design resulted in a readily portable group of set and relation manipulation facilities with rigidly defined interfaces to the user programs and to the operating system. Sets are used internally by the system in self-describing data structure which catalogs each user's sets and their characteristics.

Because one of the main benefits of a relational model of information is its simplicity, care was taken to keep the data description and manipulation languages simple. All data managed by RDMS is organized in relation sets viewed by the user only as named collections of named domains. The user need not concern himself with the type of a domain, nor its precision, nor its storage representations. All data manipulation facilities store their output in sets which may be manipulated by other RDMS commands. The command format is simple and consistent containing an output set name (if the command produces an output set), a keyword identifying the command, and the parameters of that command. These parameters may be set names, domain names, domain values, and character strings to label output displays. For example, a command which extracts a subset of a set is: "CHEAP_ WIDGET _ SET = SUBSET OF WIDGET _ SET WHERE COST LT 100.00". Commands are typed on a keyboard, and the system responds with a full screen graphic display.

RDMS commands may be grouped in four main classes: *Set control commands* which manipulate entire sets (create, destroy, save, catalog, uncatalog, universe, etc.); *Display commands* which display or print the contents of sets (list, graph, histogram, print, etc.); *Set manipulation commands* which specify, modify, analyze, select, and combine the contents of sets (union, intersection, subset, join, statistics, summary, set, domains, etc.); and *Special purpose commands* which provide miscellaneous facilities for system maintenance, bulk input and output, and assorted user conveniences (explain, command list and trace, read from, describe, etc.).

Several small data analysis and manipulation applications were tried using RDMS to test its adequacy for flexible data manipulation. A medical records analysis was particularly informative because the problems to be solved were specified by persons with neither programming nor data base experience. We were given thirty data items of eight different data types for each of several hundred mother-child pairs and asked to find any effects of a particular medication. The large amount of information displayed on the graphic console and the power of individual commands were demonstrated by answering an initial set of 41 questions with only 35 commands. Some of the more pleasing features of the system are the following. Combining individual commands into very complex requests is greatly facilitated by maintaining all data in sets used both for inputs and outputs of commands. Histograms and graphs of data from sets may either be displayed on the graphic terminal or printed on the printer. A permanent record of all commands, any screen display, and the contents of any set can be printed. Mistakes can often be undone by the REMEMBER and FORGET commands which provide an explicit checkpointing facility for each set. The main complaints from users were the paucity of data types available, the difficulty of editing erroneous or missing data items, and the inability to distinguish missing data by a special null code.

We feel the RDMS system was a successful and useful experiment from the viewpoints of both the system user and the system implementer. Sets of relations manipulated by powerful descriptive commands are usable in real information systems. Non-programmers can readily adapt to the relational model of data and the corresponding types of data manipulation commands. For the system implementer, the relational data structure provides a

simple and consistent collection of data description and manipulation facilities with which to build a variety of information systems.

SUMMARY

Data management has evolved through at least three distinct stages and is entering a fourth, from access methods, to file management systems, to data management systems, and toward information systems. In this evolution, the user's facilities for dealing with large amounts of information have become more general, flexible, extensible, and modular. Gradually he has been unburdened of various tedious levels of detail allowing him to focus his attention more directly on the relationships among various types of information and their manipulation. These trends will continue, spurred on by advances in information system design theory and in computing system hardware and software.

ACKNOWLEDGMENTS

The author is indebted to E. F. Codd, C. P. Date, G. G. Dodd, and A. Metaxides for many long hours of discussion on the subjects in this paper.

REFERENCES

1. CODASYL Data Base Task Group April 1971 Report, Available from ACM.
2. Codd, E. F., "A Relational Model of Data for Large Shared Data Bases," *Communications of the ACM*, Vol. 13, No. 6, June 1970.
3. Childs, D. L., "Feasibility of a Set-Theoretic Data Structure," *Proceedings of IFIP*, 1968.
4. Codd, E. F., "Relational Completeness of Data Base Sublanguage," *Proceedings of Courant Institute Symposium on Data Base Systems*, 1971.
5. Dayl, C. J., Hopewell, P., "Storage Structure and Physical Data Independence," *Proceedings of ACM SIGFIDET Workshop*, 1971.
6. Hardgrave, W. T., "BOLTS - A Retrieval Language for Tree Structured Data Base Systems," *Proceedings of the Fourth International Conference on Information Systems* (COINS-72), December 1972.
7. Whitney, V. K. M., "A Relational Data Management System (RDMS)," *Proceedings of the Fourth International Conference on Information Systems* (COINS-72), December 1972.

Representation of sets on mass storage devices for information retrieval systems

by STUART T. BYROM

University of Tennessee
Knoxville, Tennessee

and

by WALTER T. HARDGRAVE

CERN—European Organization for Nuclear Research
Switzerland

INTRODUCTION

Information retrieval systems utilizing Boolean operators to manipulate subsets of the data base require an efficient method of set representation. Commands in the user language permit the selection of subsets by retrieving occurrences in the data base that obey user specified conditions. Compound conditions may then be obtained by performing the traditional Boolean operators AND, OR, and NOT to selected subsets. Entries in the data base may be assigned identification numbers in order that the representations of subsets may be in the form of a set of positive identification numbers. Thus, the problem of manipulating large sets of occurrences reduces to one of efficiently representing subsets of positive integers.

For example, information stored in nodes structured as data trees may be retrieved via a qualification clause which selects nodes satisfying an attribute-relation-value (*a-r-v*) triple. The *a-r-v* triple represents a subset of the universe of all nodes in the tree. If the nodes in the data tree are assigned consecutive positive integer numbers, then a set of nodes may be represented by a set of identification numbers. The number of nodes in the universe will be assumed to be no more than 2^{31}, or approximately two billion, although the assumption will also be made that any one subset will be small with respect to the universe. However, this assumption will not hold for the complement of a set, that is, the result of a Boolean NOT. Because sets may be quite large, it is necessary to store them as efficiently as possible. One method of storing sets of positive integers in order to conserve storage space is the Bit Stream Set Representation (BSSR) which assigns a single binary digit (bit) in storage to each node in the universe.

BIT STREAM SET REPRESENTATION

The definitions involving the Bit Stream Set Representation will be followed by the algorithms for performing the Boolean operations on these representations along with several examples.

Definition of a bit stream set representation

Let P be the set of positive integers. Let S be a subset of the set of contiguous integers 1 through k, and let R be a Bit Stream Set Representation (BSSR) of S, defined as follows:

$$R=(b_0, b_1, \ldots, b_k)$$

such that

$$b_i \in R, \quad i \in P \quad \text{for} \quad 1 \leq i \leq k. \tag{1}$$

$$b_i = 1 \quad \text{for all} \quad i \in S,$$

and

$$b_i = 0 \quad \text{for all} \quad i \notin S. \tag{2}$$

$$\text{if } R \text{ is in } \textit{complement form} \text{ (to be defined later)}, \quad \text{then} \quad b_0=1, \quad \text{otherwise} \quad b_0=0. \tag{3}$$

Thus, the BSSR is a one-to-one mapping from the set of integers 1 through k to the ordered $k+1$ bits. The inclusion of the integer i in the set S is represented by the ith bit having the value "1". Likewise, the integer i not in the set S is represented by its corresponding ith bit having the value "0". Each subset of P has a unique bit stream representation. Furthermore, every subset of the integers 1 through k may be represented by a bit stream of length $k+1$.

For example, the set {2, 5, 100000, 2000000000} is a subset of the integers 1 through $2^{31}-1$ (i.e., 2,137,483,647) and may be represented by the BSSR

$$(0010010 \ldots 010 \ldots 010 \ldots 0)$$

such that

$$b_i = 0 \quad \text{for} \quad 0 \leq i \leq 2^{31}-1$$

except where

$$b_2 = b_5 = b_{100000} = b_{2000000000} = 1.$$

In this manner, any subset of the integers 1 through $2^{31}-1$ may be represented by a BSSR of length 2^{31}.

Bit stream set representation with directories

When a set is represented in the bit stream form, a substream of zero bits may be eliminated in order to save space with no loss of information. By dividing the BSSR into equal length substreams called blocks, it is possible to omit those which contain all zero bits. However, in order to maintain the contiguous numbering scheme of the bits, it is necessary to indicate any omissions with a directory.

Definition of a block

Let R be the BSSR of the set S as previously defined. Let R be divided into k/m distinct substreams $B_{0,j}$ of length m. Then each $B_{0,j}$ is a block of R and is defined as follows:

$$B_{0,j}=(b_{mj}, b_{mj+1}, \ldots, b_{m(j+1)-1})$$

such that

$$B_{0,j}\subseteq R \quad \text{for} \quad 0\leq j\leq k/m-1.$$

In addition, the definition implies the following:

$$B_{0,0}\cap B_{0,1}\cap \ldots \cap B_{k/m-1}=(\,) \tag{1}$$

$$B_{0,0}\cup B_{0,1}\cup \ldots \cup B_{k/m-1}=R. \tag{2}$$

Finally, to indicate a block of all zero bits, the notation

$$B_{0,j}=\bar{0}$$

implies for all $b_i\in B_j$ that $b_i=0$ for $mj\leq i\leq m(j+1)-1$.

In the previous example, the BSSR of the set S may be divided into 2^{20} blocks of length $m=2^{11}$ bits as follows:

$$\begin{aligned}
B_{0,0}&=(0010010\ldots0)\\
B_{0,1}&=(0\ldots\quad\ 0)\\
&\vdots\\
B_{0,48}&=(0\ldots010\ldots0)\\
&\vdots\\
B_{0,976562}&=(0\ldots010\ldots0)\\
&\vdots\\
B_{0,1048575}&=(0\ldots\quad\ 0)
\end{aligned}$$

such that

$$B_{0,0}\neq\bar{0}, \quad B_{0,48}\neq\bar{0}, \quad B_{0,976562}\neq\bar{0},$$

otherwise

$$B_{0,j}=\bar{0} \quad \text{for} \quad 1\leq j\leq 2^{20}-1.$$

Definition of a directory

Let R be a BSSR of the set S and let $B_{0,0}, B_{0,1}, \ldots, B_{0,k/m-1}$ be the blocks of R as defined above. Then the bit stream directory D_1 may be defined as follows:

$$D_1=(d_{1,0}, d_{1,1}, \ldots, d_{1,k/m-1})$$

such that

if $B_{0,j}\neq\bar{0}$, then $d_{1,j}=1$
if $B_{0,j}=\bar{0}$, then $d_{1,j}=0$ for $0\leq j\leq k/m-1$.

In the example, the directory for the BSSR of S is

$$D_1=(10\ldots010\ldots010\ldots0)$$

where

$$d_{1,0}=d_{1,48}=d_{1,976562}=1$$

otherwise

$$d_{1,j}=0 \quad \text{for} \quad 0\leq j\leq 2^{20}-1.$$

By prefixing the directory to the three non-zero blocks of the BSSR of S, the following unambiguous representation results:

$$\underbrace{(10\ldots010\ldots010\ldots0}_{D_1}\ \underbrace{0010010\ldots0}_{B_{0,0}}\ \underbrace{0\ldots010\ldots0}_{B_{0,48}}\ \underbrace{0\ldots010\ldots0)}_{B_{0,976562}}$$

The length of the resulting BSSR is $2^{20}+3(2^{11})=1{,}054{,}720$ which is substantially less than the BSSR without a directory 2^{31}.

The recursive property of directories

As described above, the directory D_1 indicates which blocks of the BSSR are omitted in order to conserve storage space. However, the length of D_1 (possibly very large itself) may also contain relatively long substreams of zero bits. Thus, by dividing this directory into blocks of length n, a higher level directory D_2 may be constructed to indicate omissions in D_1. In fact, the idea of a "directory to a directory" may be applied recursively as many times as desired to eliminate all blocks containing all zero bits.

Definition of a directory block

A directory block is defined in the same manner as is a block of R (i.e., $B_{0,j}$). That is, first let the level L directory D_L be divided into substreams of length n_L; then let $B_{L,j}$ be a directory block of D_L defined as follows:

$$B_{L,j}=(d_{L,0}, d_{L,1}, \ldots, d_{L,n_L})$$

such that

$$B_{L,j}\subseteq D_L \quad \text{for} \quad L\geq 1$$

and

$$\text{for} \quad 0\leq j\leq \frac{k}{mn_1n_2\ldots n_L}-1=\max_L.$$

Definition of higher level directories

The directory D_1 is defined to be the lowest level directory ($L=1$) to R in a BSSR. Using the above definition of directory blocks, a higher level directory D_L may be defined as follows:

$$D_L=(d_{L,0}, d_{L,1}, \ldots, d_{L,\max_L})$$

such that

if $B_{L-1,j}\neq\bar{0}$, then $d_{L,j}=1$
if $B_{L-1,j}=\bar{0}$, then $d_{L,j}=0$ for $0\leq j\leq\max_L$.

The level 1 directory D_1 of the example BSSR may be divided into blocks of length $n_1=2^{11}$ as follows:

$$\begin{aligned} B_{1,0} &= (10\ldots010\ldots0)\\ B_{1,1} &= (\ 0\ldots\qquad\quad 0)\\ &\vdots\\ B_{1,428} &= (\ 0\ldots010\ldots0)\\ &\vdots\\ B_{1,511} &= (\ 0\ldots\qquad\quad 0) \end{aligned}$$

where all blocks except $B_{1,0}$ and $B_{1,428}$ contain all zero bits. Thus, the level 2 directory for the BSSR is

$$D_2=(10\ldots010\ldots0)$$

such that

$$d_{2,0}=d_{2,428}=1,$$

and otherwise

$$d_{2,j}=0 \quad \text{for} \quad 0\leq j\leq 2^9-1.$$

Each D_1 directory block that contains all zeros may be omitted when prefixed by D_2. In the example BSSR, the representation would then be

$$\underbrace{(10\ldots010\ldots0}_{D_2}\ \underbrace{10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{0\ldots010\ldots0}_{B_{1,428}}$$

$$\underbrace{0010010\ldots0}_{B_{0,0}}\ \underbrace{0\ldots010\ldots0}_{B_{0,48}}\ \underbrace{0\ldots010\ldots0)}_{B_{0,976562}}$$

The length of this stream is $k/mn_1+n_1+3m=2^9+2^{11}+3(2^{11})=8704$. Although this length is small in comparison with a BSSR without directories (i.e., 2,131,483,648), it is somewhat long for a representation of only four values. However, it is clear that a much larger number of values could be included in the representation without increasing its length. For example, if each bit of the three blocks of R were "1" (D_1 and D_2 remaining unchanged), then the number of values represented would be $3(2^{11})=6144$. While it is unlikely that any subset chosen would "fit" this well, it is not unlikely for a set of values to be somewhat grouped, although not necessarily contiguous (as subsets of the integer valued nodes of a tree structured data base). In addition, by selecting different block sizes and more levels of directories, it is possible to further improve savings in storage requirements. In general, the optimum block sizes are dependent upon the universe and the characteristics of the types of subsets formed.

Boolean operations on BSSR's

One of the advantages of representing sets with bit streams is the ease with which Boolean operations may be performed. The operations AND and OR are possible by pairing the bits of both BSSR's corresponding to each integer value in the universe. For example, to perform the Boolean AND on two subsets

$$S_1=\{2, 5, 100000, 2000000000\}$$
$$S_2=\{1, 2, 100000\}$$

(from the universe 1 through $2^{31}-1$), each pair of bits in the BSSR's

$$\begin{aligned} \text{BSSR}_1 &= (0010010\ldots010\ldots010\ldots0)\\ \text{BSSR}_2 &= (0110000\ldots010\ldots\qquad 0) \end{aligned}$$

may be operated on by the Boolean AND. The resulting bit stream will be called BSSR*, such that

$$\text{BSSR}^* = (0010000\ldots010\ldots\qquad 0).$$

The Boolean NOT may be performed by simply reversing each binary value (except b_0), or by setting b_0 to the value "1" indicating the complement. When $b_0=1$, the BSSR is defined to be in *complement form*. Both of the following BSSR's (without directories) represent the set $\sim S_1$ but only the first is said to be in complement form:

$$(1010010\ldots010\ldots010\ldots0)$$
$$(0101101\ldots101\ldots101\ldots1).$$

For BSSR's without directories, the complement will require exactly the same amount of storage. However, in large universes it is impractical not to use directories to eliminate long streams of zeros. In this case, the BSSR for the complement of a small subset may be unmanageably large. Thus, it may suffice to set the complement flag b_0 to "1" in order to conserve storage. Since complements may often be used for intermediate, or temporary, results it is more efficient to convert a BSSR to its complement only when necessary.

For BSSR's without directories, Boolean operations are straightforward. However, more useful are BSSR's with one or more levels of directories which require algorithms to produce the resultant BSSR*. The algorithms to follow assume that the BSSR's used in a Boolean operation will have common block lengths and levels of directories. (If necessary, higher level directories of "all ones" may be generated for a BSSR to meet this requirement). The algorithms permit operations on BSSR's in complement form so that conversion to the actual complement of a BSSR (via the Boolean NOT) need not be made.

Algorithm for the Boolean AND

If both BSSR's are in complement form, then use the algorithm that performs the Boolean OR to obtain a BSSR* also in complement form; by theorem $\sim R_1$ AND $\sim R_2$ is equivalent to $\sim(R_1$ OR $R_2)$. (Note that both of these BSSR's should be treated in the OR algorithm *as if* each were not in complement form.)

Step 1: Let L be equal to the number of the highest directory level of the two BSSR's.

Step 2: If neither of the two BSSR's are in complement form, then the resultant level L blocks of BSSR* are equal to the Boolean AND of the two BSSR's level L blocks.

If *one* of the two BSSR's is in complement form, then the level L blocks of BSSR* are equal to those of the uncomplemented BSSR.

Step 3: If a BSSR is not in complement form, then for each $d_{L,j}^*=0$ in D_L^* such that $d_{L,j}=1$ in that BSSR, omit the block $B_{L-1,j}$ from D_{L-1}. Furthermore, omit all lower level blocks (through level zero) within the range of $B_{L-1,j}$.
Step 4: If $L>1$ then subtract "1" from L and go to Step 2.
Step 5: If neither BSSR is in complement form, go to Step 6; otherwise, replace the level zero blocks of the BSSR in complement form with their complement (Boolean NOT). Then, for each $d_{1,j}^*=1$ in D_1^* such that $d_{1,j}=0$ in D_1 of that BSSR, insert the block $B_{0,j}=\overline{1}$ (which is a stream of all ones) in the proper relative position in D_0.
Step 6: The level zero blocks of BSSR* are equal to the Boolean AND of the level zero blocks of the two BSSR's.

Algorithm for the Boolean OR

If both BSSR's are in complement form, then use the AND algorithm to obtain the BSSR* in complement form since by theorem $\sim R_1$ OR $\sim R_2$ is equivalent to $\sim(R_1$ AND $R_2)$. (Again note that both of these BSSR's should be treated in the AND algorithm as if each were not in complement form.)

Step 1: Let L be equal to the number of the highest directory level of the two BSSR's.
Step 2: The level L blocks of BSSR* are equal to the Boolean OR of each corresponding pair of bits of the level L blocks of the two BSSR's.
Step 3: For each $d_{L,j}^*=1$ in D_L^* such that $d_{L,j}=0$ in a BSSR, insert the block $B_{L-1,j}=\overline{0}$ in the proper relative position in D_{L-1} in that BSSR.
Step 4: If $L>1$, then subtract "1" from L and go to Step 2.
Step 5: If neither BSSR is in complement form, go to Step 6; otherwise, replace the level zero blocks of the BSSR *not* in complement form with their complement (Boolean NOT). Finally, perform the Boolean *AND* on the level zero blocks of the two BSSR's to obtain the resultant level zero blocks of BSSR* in complement form. (Terminate procedure).
Step 6: The level zero blocks of BSSR* are equal to the Boolean OR of the level zero blocks of the two BSSR's.

Example Boolean operations

The sets S_1 and S_2 and their complements will be used in the examples to follow where

$$S_1=\{2, 5, 100000, 2000000000\}$$
$$S_2=\{1, 2, 100000\}.$$

Example: S_1 AND S_2

BSSR$_1$:

$$\underbrace{(10\ldots010\ldots0}_{D_2}\ \underbrace{10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{0\ldots010\ldots0}_{B_{1,428}}$$

$$\underbrace{0010010\ldots0}_{B_{0,0}}\ \underbrace{0\ldots010\ldots0}_{B_{0,48}}\ \underbrace{0\ldots010\ldots0)}_{B_{0,976562}}$$

BSSR$_2$:

$$\underbrace{(10\ldots0}_{D_2}\ \underbrace{10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{0110\ldots0}_{B_{0,0}}\ \underbrace{0\ldots010\ldots0)}_{B_{0,48}}$$

Step 1: $L=2$
Step 2: Find Boolean AND of level 2 directories:

D_2 of BSSR$_1$ = (10 . . . 010 . . . 0)
D_2 of BSSR$_2$ = (10 . . . 0)
D_2* of BSSR* = (10 . . . 0)

Step 3: Since $d_{2,428}^*=0$ in D_2^* and $d_{2,428}=1$ in D_2 of BSSR$_1$, omit block $B_{1,428}$ from BSSR$_1$. Next omit all blocks within the range of $B_{1,428}$ in BSSR$_1$; i.e., block $B_{0,976562}$. Thus, BSSR$_1$ =

$$\underbrace{(10\ldots0}_{D_2}\ \underbrace{10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{0010010\ldots0}_{B_{0,0}}\ \underbrace{0\ldots010\ldots0)}_{B_{0,48}}$$

Step 4: $L=2-1=1$
Step 2: Find Boolean AND of level 1 directories:

D_1 of BSSR$_1$ = (10 . . . 010 . . . 0)
D_1 of BSSR$_2$ = (10 . . . 010 . . . 0)
D_1* of BSSR* = (10 . . . 010 . . . 0)

Step 3: (No occurrences)
Step 4: $L=1$
Step 5: (Neither is in complement form so continue to Step 6).
Step 6: Find Boolean AND of level 0 blocks:

R of BSSR$_1$ = (0010010 . . . 0 0 . . . 010 . . . 0)
R of BSSR$_2$ = (0110000 . . . 0 0 . . . 010 . . . 0)
R of BSSR* = (0010000 . . . 0 0 . . . 010 . . . 0)

Since $b_2=b_{100000}=1$, the resulting BSSR* represents the set {2, 100000}.

Example: S_1 AND $\sim S_2$

BSSR$_1$ is shown in the previous example; BSSR$_2$ is as follows in complement form:

$$\text{BSSR}_2=\underbrace{(10\ldots0}_{D_2}\ \underbrace{10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{1110\ldots0}_{B_{0,0}}\ \underbrace{0\ldots010\ldots0)}_{B_{0,48}}$$

Step 1: $L=2$
Step 2: Since BSSR$_2$ is in complement form, then
D_2* of BSSR* = D_2 of BSSR$_1$ = (10 . . . 010 . . . 0)
Step 3: (No occurrences)
Step 4: $L=2-1=1$
Step 2: Since BSSR$_2$ is in complement form, then

$$D_1^* \text{ of BSSR}^* = D_1 \text{ of BSSR}_1 = \underbrace{(10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{0\ldots010\ldots0)}_{B_{1,428}}$$

Step 3: (No occurrences)
Step 4: $L=1$
Step 5: Since $BSSR_2$ is in complement form, then replace its level zero blocks with their complement (reverse each binary value).

$$R \text{ of } BSSR_2 = \underbrace{(0001\ldots1}_{B_{0,0}}\ \underbrace{1\ldots101\ldots1)}_{B_{0,48}}$$

Since $d_{1,976562}{}^*=1$ in $D_1{}^*$ and $d_{1,976562}=0$ in D_1 of $BSSR_2$, then insert the block $B_{0,976562}=\bar{1}$.

$$R \text{ of } BSSR_2 = \underbrace{(0001\ldots1}_{B_{0,0}}\ \underbrace{1\ldots101\ldots1}_{B_{0,48}}\ \underbrace{1\ldots\quad 1)}_{B_{0,976562}}$$

Step 6: Find Boolean AND of level zero blocks:

$$\begin{aligned} R \text{ of } BSSR_1 &= (0010010\ldots00\ldots010\ldots00\ldots010\ldots0) \\ R \text{ of } BSSR_2 &= (0001111\ldots11\ldots101\ldots11\ldots\quad 1) \\ R \text{ of } BSSR^* &= \underbrace{(0000010\ldots00\ldots}_{B_{0,0}}\ \underbrace{\quad 00\ldots010\ldots0)}_{B_{0,48}\quad B_{0,976562}} \end{aligned}$$

Since $b_5=b_{2000000000}=1$, the resulting BSSR* represents the set $\{5, 2000000000\}$.

Example: S_1 OR S_2

The BSSR's for S_1 and S_2 are shown in a previous example.
Step 1: $L=2$

Step 2: Find Boolean OR of level 2 directories:

$$\begin{aligned} D_2 \text{ of } BSSR_1 &= (10\ldots010\ldots0) \\ D_2 \text{ of } BSSR_2 &= (10\ldots\quad 0) \\ D_2{}^* \text{ of } BSSR^* &= (10\ldots010\ldots0) \end{aligned}$$

Step 3: Since $d_{2,428}{}^*=1$ and $d_{2,428}=0$ in $BSSR_2$, insert the block $B_{1,428}$ into $BSSR_2$.

$$D_1 \text{ of } BSSR_2 = \underbrace{(10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{0\ldots0)}_{B_{1,428}}$$

Step 4: $L=2-1=1$
Step 2: Find Boolean OR of level 1 directories:

$$\begin{aligned} D_1 \text{ of } BSSR_1 &= (10\ldots010\ldots0\ 0\ldots010\ldots0) \\ D_1 \text{ of } BSSR_2 &= (10\ldots010\ldots0\ 0\ldots\quad 0) \\ D_1{}^* \text{ of } BSSR^* &= \underbrace{(10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{0\ldots010\ldots0)}_{B_{1,428}} \end{aligned}$$

Step 3: Since $d_{1,976562}{}^*=1$ and $d_{1,976562}=0$ in $BSSR_2$, insert the block $B_{0,976562}=\bar{0}$ into $BSSR_2$.

$$R \text{ of } BSSR_2 = \underbrace{(0110\ldots0}_{B_{0,0}}\ \underbrace{0\ldots010\ldots0}_{B_{0,48}}\ \underbrace{0\ldots\quad 0)}_{B_{0,976562}}$$

Step 4: $L=1$
Step 5: Both BSSR's are not in complement form so continue to Step 6.

Step 6: Find Boolean OR of level zero blocks:

$$\begin{aligned} R \text{ of } BSSR_1 &= (0010010\ldots00\ldots010\ldots00\ldots010\ldots0) \\ R \text{ of } BSSR_2 &= (0110000\ldots00\ldots010\ldots00\ldots\quad 0) \\ R \text{ of } BSSR^* &= \underbrace{(0110010\ldots0}_{B_{0,0}}\ \underbrace{0\ldots010\ldots0}_{B_{0,48}}\ \underbrace{0\ldots010\ldots0)}_{B_{0,976562}} \end{aligned}$$

Since $b_1=b_2=b_5=b_{100000}=b_{2000000000}=1$, the resulting BSSR* represents the set $\{1, 2, 5, 100000, 2000000000\}$.

Example: S_1 OR $\sim S_2$

The BSSR's for S_1 and $\sim S_2$ are shown in a previous example.

Step 1: $L=2$
Step 2: Find Boolean OR of level 2 directories:

$$\begin{aligned} D_2 \text{ of } BSSR_1 &= (10\ldots010\ldots0) \\ D_2 \text{ of } BSSR_2 &= (10\ldots\quad 0) \\ D_2{}^* \text{ of } BSSR^* &= (10\ldots010\ldots0) \end{aligned}$$

Step 3: Since $d_{2,428}{}^*=1$ and $d_{2,428}=0$ in $BSSR_2$, insert the block $B_{1,428}=\bar{0}$ into $BSSR_2$.

$$D_1 \text{ of } BSSR_2 = \underbrace{(10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{0\ldots\quad 0)}_{B_{1,428}}$$

Step 4: $L=2-1=1$
Step 2: Find Boolean OR of level 1 directories:

$$\begin{aligned} D_1 \text{ of } BSSR_1 &= (10\ldots010\ldots0\ 0\ldots010\ldots0) \\ D_1 \text{ of } BSSR_2 &= (10\ldots010\ldots0\ 0\ldots\quad 0) \\ D_1{}^* \text{ of } BSSR^* &= \underbrace{(10\ldots010\ldots0}_{B_{1,0}}\ \underbrace{0\ldots010\ldots0)}_{B_{1,428}} \end{aligned}$$

Step 3: Since $d_{1,976562}{}^*=1$ and $d_{1,976562}=0$ in $BSSR_2$, insert the block $B_{0,976562}=\bar{0}$ into $BSSR_2$.

$$R \text{ of } BSSR_2 = \underbrace{(1110\ldots0}_{B_{0,0}}\ \underbrace{0\ldots010\ldots0}_{B_{0,48}}\ \underbrace{0\ldots\quad 0)}_{B_{0,976562}}$$

Step 4: $L=1$
Step 5: Replace the level zero blocks of the BSSR not in complement form with their complement (reverse each binary value).

$$R \text{ of } BSSR_1 = \underbrace{(1101101\ldots1}_{B_{0,0}}\ \underbrace{1\ldots101\ldots1}_{B_{0,48}}\ \underbrace{1\ldots101\ldots1)}_{B_{0,976562}}$$

Next, find the Boolean AND of the level zero blocks:

$$\begin{aligned} R \text{ of } BSSR_1 &= (1101101\ldots11\ldots101\ldots11\ldots101\ldots1) \\ R \text{ of } BSSR_2 &= (1110000\ldots00\ldots010\ldots00\ldots\quad 0) \\ R \text{ of } BSSR^* &= (1100000\ldots00\ldots\quad 00\ldots\quad 0) \end{aligned}$$

Since $b_1=1$ and $b_0=1$, then the resulting BSSR* is in the complement form representing the set $\sim\{1\}$.

Algorithm for the Boolean NOT

In order to perform the Boolean NOT on a BSSR in *complement form* (as previously defined), simply reverse the value of $b_0=1$ to $b_0=0$. In general, the resultant BSSR* of a BSSR not in complement form will be large for small subsets, possibly approaching the size of the universe itself. Therefore, the use of a BSSR in *complement form* as opposed to BSSR*, the result of a NOT algorithm, should be significantly more efficient.

Step 1: Let H be equal to the number of the highest level directory of the BSSR, and let $L=0$. (Note that $R=D_0$)
Step 2: Perform the Boolean NOT on each bit in the level L blocks by reversing its binary value (with the exception of b_0 in $B_{0,0}$).
Step 3: For each $B_{L,j}\neq\bar{0}$, change its corresponding $L+1$ directory bit to "0".
Step 4: Insert all omitted blocks from D_L with blocks of "all ones."
Step 5: Omit all level L blocks where $B_{L,j}=\bar{0}$.
Step 6: Add "1" to L. If $L<H$ go to Step 2. Otherwise terminate.

Example: NOT S_1

The BSSR for S_1 is shown in a previous example.
Step 1: $H=2$ and $L=0$
Step 2: Find Boolean NOT of R:

$$D_0=R=(0010010\ldots0\ 0\ldots010\ldots0\ 0\ldots010\ldots0)$$
$$D_0{}^*=R^*=(\underbrace{0101101\ldots1}_{B_{0,0}}\ \underbrace{1\ldots101\ldots1}_{B_{0,48}}\ \underbrace{1\ldots101\ldots1}_{B_{0,976562}})$$

Step 3: Since $B_{0,0}\neq\bar{0}$, $B_{0,48}\neq\bar{0}$, and $B_{0,976562}\neq\bar{0}$, then $d_{1,0}=d_{1,48}=d_{1,976562}=0$.

$$D_1{}^*=(\underbrace{0\ldots0}_{B_{1,0}}\ \underbrace{0\ldots0}_{B_{1,428}})$$

Step 4: Insert all omitted blocks (with blocks of "all zeros") into R:

$$R^*=(\underbrace{0101101\ldots1}_{B_{0,0}}\ \underbrace{1\ldots1}_{B_{0,1}}\ \ldots\ \underbrace{1\ldots101\ldots1}_{B_{0,48}}\ \ldots$$
$$\underbrace{1\ldots101\ldots1}_{B_{0,976562}}\ \ldots\ \underbrace{1\ldots1)}_{\ldots\ B_{0,1048575}}$$

Step 5: (each block $B_{0,j}\neq\bar{0}$)
Step 6: $L=0+1=1$
Step 2: Find Boolean NOT of $D_1{}^*$:

$$D_1{}^*=(\underbrace{1\ldots1}_{B_{1,0}}\ \underbrace{1\ldots1)}_{B_{1,428}}$$

Step 3: Since $B_{1,0}\neq\bar{0}$ and $B_{1,428}\neq\bar{0}$, then $d_{2,0}=d_{2,428}=0$.

$$D_2{}^*=(0\ldots0)$$

Step 4: Insert all omitted blocks (with blocks of "all ones") into D_1.

$$D_1{}^*=(\underbrace{1\ldots1}_{B_{1,0}}\ \underbrace{1\ldots1}_{B_{1,1}}\ \ldots\ \underbrace{1\ldots1)}_{B_{1,511}}$$

Step 5: (each block $B_{1,j}\neq\bar{0}$)
Step 6: $L=1+1=2$ (Terminate)

Design of tree networks for distributed data

by R. G. CASEY

IBM Research Laboratory
San Jose, California

INTRODUCTION

The advent of computerized information networks connotes a change in the normal data processing concept of information as a localized object, a disk file tied to computer X, for instance. Having available a facility for communication between computers it is more natural to think of related data files at different sites as elements of a single unified data collection. As an example, crime reports gathered in various cities across the nation inherently constitute an integrated data base of more than local interest. In banking, manufacturing, and other applications the same is often found to be true; information gathered and maintained in one geographical area is closely related to data collected elsewhere. Computer networks offer not only a replacement for unwieldy document-oriented methods of communicating between distributed data files, but also promise the flexibility needed to develop new ways of applying these data collections.

The technology, particularly the software, for managing information on a global scale is still in the future. It is not premature, however, to develop descriptive models of information networks and to seek rational design techniques for specifying key parameters.

In this paper the overall configuration of an information network is investigated. Employing estimates of users' needs for interaction with the various components of a data base, we seek to determine how the data resources should be deployed, and what communications links should be implemented in order to provide the required service at least cost.

While communications design, even with applications to computer networks, has a considerable literature,[1,2] we shall see that a network of relocatable data files poses design questions not answered by conventional analyses. We shall show that for at least one important class of networks an iteractive algorithm, essentially an extension of a reported technique for constructing a net to communicate with a fixed data base, offers promise as an automatic design procedure.

The nature of distributed data networks

Present-day information networks are often insufficient for dealing with large, distributed data bases. In many applications (e.g., airline reservations,[3] inventory systems) data files are centralized and all interaction in the net is between an outlying node and the central facility. More flexible networks, e.g., ARPA,[4] transact with data either by establishing a terminal-type connection between it and the user, or else cause whole files to be shipped to the users' location. In all these systems the user must have precise knowledge of implementational details such as data location and format.

In the field of data management present-day trends are toward freeing the user of the need to be familiar with device particulars as a prerequisite to accessing data.[5] Under the concept of "data independence" a user ideally would interact with a data collection in natural semantic terms rather than by means of a series of hardware instructions. We feel confident that this goal will be important in the management of distributed data as well as in centralized operations. In fact the need will be greater since network data bases will be merged facilities, with users unfamiliar with either data structures or machines at remote locations.

The concept of an information network postulated here is one in which data can be freely allocated. The overall data base is assumed to be partitioned into files that can be stored independently in the network. If convenient, the same file may even be maintained at more than one location. This type of operation can be carried out with present day networks given a well-schooled set of users. However, as indicated above future network-oriented information systems will probably have such flexibility built in.

Configuration design

We shall approach the task of establishing a network of distributed data files as if we were freshly creating such a system. In practice this will usually be a poor assumption. Typically the data collection, storage, and processing activities in question will have been in operation for some time, and limited communication will be carried on over teleprocessing lines, or at least by mail. Given a flexible new tool for coordinating all these activities and for instituting new applications it may still be desirable to ease the transition by gradual conversion to the new system. In addition, other pressures than economy and efficient

operation may have a determining effect on design parameters. Staff competency, company traditions, managerial ambitions, and other human elements may (and one might hope, will) influence network characteristics. It is still reasonable to expect that the network administration will want to have at hand an ultimate plan for the network, a vision of what would constitute the most efficient network, possibly under design constraints arising through non-technological factors.

In the following we provide a method for developing such a plan. The objective is to design a network that minimizes the cost, in communications channel rentals and in data storage and maintenance charges, of carrying on a given (estimated) transaction load over the network.

A MODEL FOR A DISTRIBUTED DATA NETWORK

The model for network behavior that we investigate is a static one. We concern ourselves with average traffic rates over the lines.

In addition we assume that information can be transmitted between two locations through intermediate nodes that may also be carrying on independent communications using the same channels. Presumably this rules out switched line networks. Consequently the data communications net must be of the store-and-forward type in which high line utilization can be achieved by transmitting arbitrary messages as a series of shorter message packets, each having its own destination code. This class of operation is in agreement with our view of the network as a multi-user, real-time interactive facility.

Given data

(1) A list of user node locations and a list of possible storage node and processing locations.
(2) A list of files.
(3) A matrix of users vs. files listing the average transaction volume between the two in (a) query mode and (b) update mode.
(4) A matrix of files vs. storage nodes expressing storage costs.
(5) A list of available channel capacities.
(6) $C(i,j,k)$=the monthly rental cost of a communication channel having the k^{th} capacity and linking nodes i and j.

Design objectives

To find an assignment of files to storage sites, and a network topology and associated line capacities that will carry the resulting traffic load, while minimizing the total cost of leasing the lines and storing the files.

The Appendix expresses in symbols the optimization problem posed verbally above. Two features of this model distinguish it from conventional problems in network design:

(1) inclusion of a discrete table of line capacities and costs.
(2) file assignment as a design variable.

The first point above is important in a practical sense, since channel capacity is offered in discrete chunks by the common carriers,[6] and the tariff rate tables defy approximation by elementary functions (although this has been studied[2]).

The model is nonlinear and contains integer variables as well as continuously variable quantities. Standard techniques in network flow and linear programming do not appear to be directly applicable to the task of finding the optimal network. In such a problem area it is necessary to turn to heuristic approaches to network design.

One plausible idea is to treat file assignment parametrically; i.e., solve a family of network synthesis problems, one for each possible file assignment strategy. In practice this would be a time-consuming approach. For instance, the 2-file, 18-node example we consider later would pose approximately 64 billion such problems.

We prefer to consider the design problem in the context of a search procedure. We assume that a good, if not necessarily optimal, network design can be achieved by starting with an arbitrary network and iteratively subjecting it to elementary transformations, each of which improves the cost somewhat. Eventually such a program arrives at a point where no elementary transformation will reduce cost further. The goodness of this "local optimum" relative to the absolute minimum cost design depends on three factors:

(1) the class of elementary transformations employed.
(2) the initial network.
(3) the algorithm for performing successive transformations.

TREE DESIGN

The design variables we seek to optimize can be analyzed into four distinct groups.

(1) assignment of each file to a set of locations.
(2) selection of network topology.
(3) designation of routing paths between communicating nodes.
(4) specification of channel capacity for each link of the network.

The solution space for even small networks is immense. If there are n nodes, k files, and c capacities (including zero capacity) offered for each link then there are $c^{\binom{n}{2}}$ possible link capacity assignment strategies (including trivial ones), and there are 2^{kn} file location strategies. The total number of network candidates is greater than the simple product of these two quantities, since each network will pose a number of alternative routing strategies.

The magnitude of this solution space is not necessarily an absolute block to good design. The ARPA[7] design was carried out in a space lacking only the file allocation variables, and with a more complex cost criterion (involving response time as well as line costs). In the present study, however, we seek to optimize over a restricted class of networks. In particular we restrict consideration to tree-type networks (networks that are connected but contain no loops), with the following gains to compensate for the loss of generality.

(1) Routing decisions are drastically reduced. In a tree there is only a single path between any two nodes. The only routing decision, then, is which of several file copies should be queried by a given user.
(2) The effect of a local change in tree structure is easy to calculate. Thus a fast iterative algorithm is achievable.
(3) Tree networks are of practical interest. The minimum cost network to communicate with a centralized data base is a tree, and these are employed in practice.[8]

An optimal tree is a useful piece of design knowledge. Although not the best network in general (see Figure 1), a tree could, for example, be a starting point for a more general design procedure introducing criteria such as reliability or response time. In a companion paper, routines for generating more complex nets from an arbitrary starting point are described.[9]

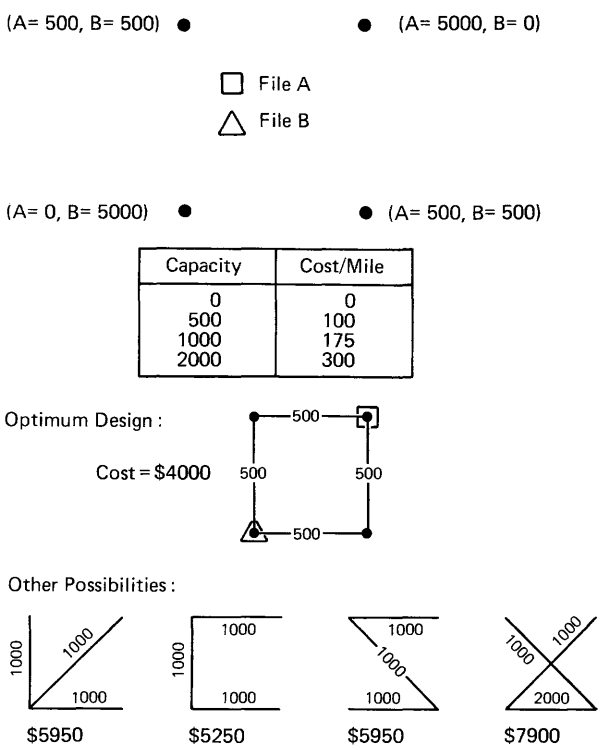


Figure 1—A sample problem in distributed network design. Traffic generated at each of the nodes (located at corners of a 10-by-10 square) is given in parentheses. A high proportion of this traffic is assumed to be updated, so that only a single copy of each file is stored. The optimal network is not a tree

A NETWORK DESIGN PROCEDURE

The Esau-Williams algorithm

In a paper by Esau and Williams[10] the problem of designing a telecommunications tree network for transmitting data from a set of outlying terminals to a single, fixed data center is analyzed. A heuristic algorithm for network design is presented. Experience with this algorithm[11,12] has shown it to be quite suitable for the design task and indeed a programmed version is commercially available.[13]

The algorithm operates within the spirit of a search for the optimal network using local transformations. It starts with a star network, every user directly connected to the data center, and tests single reconnections, retaining that reconnection that results in the greatest cost reduction. At each step a link to the data center is broken and a new connection made so as to retain network connectivity and reduce cost.

The problem attacked by the Esau-Williams technique has several features in common with the design of a distributed data network:

(1) the same loading parameters, traffic flow from each user to the data base; and (2) the same criterion, minimum line cost.

The chief differences are that only a single line capacity is assumed to be available, and the data files exist only at the given location. For this latter reason there is no need to distinguish between query and update transactions since both follow the same flow paths. The telecommunications design problem can therefore be considered to be a highly constrained version of the distributed data network design.

We make use of this relationship in constructing a design procedure. We employ the Esau-Williams algorithm as an initial step in order to design a centralized network, and then relax the constraint on file location and duplication in order to design a more general tree network. This approach has several advantages:

(1) It uses the fast, efficient Esau-Williams technique to good advantage in order to produce a network candidate for further optimization.
(2) It affords a direct comparison of the relative costs of centralized design vs. distributed design.

An extended design algorithm

A flow diagram of the design procedure analyzed here is shown in Figure 2. As shown the routine begins with an

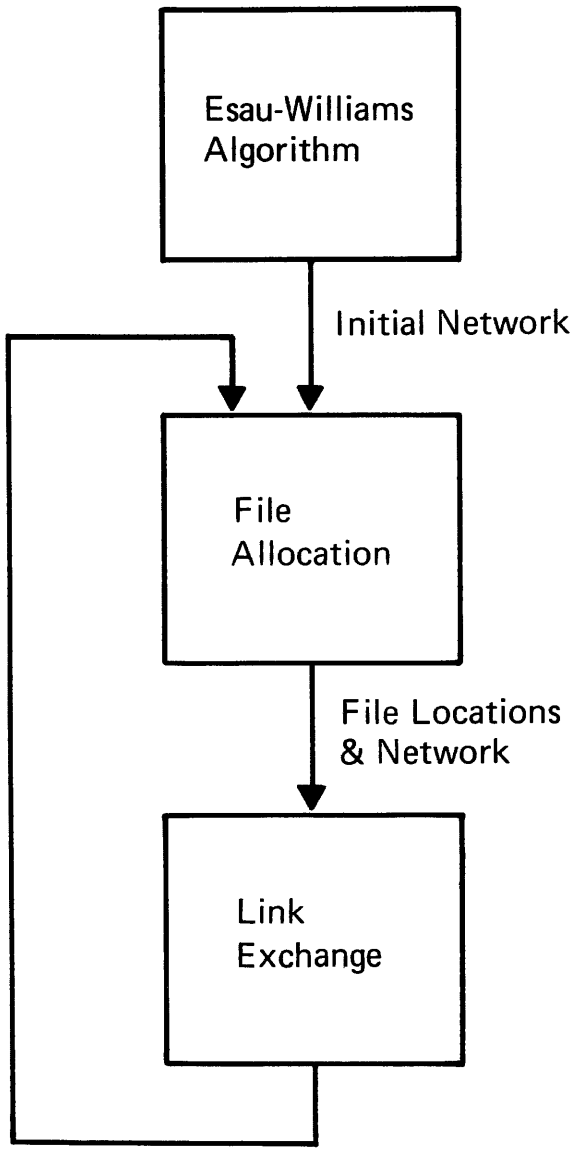


Figure 2—Flow diagram of the design process

application of the Esau-Williams algorithm assuming a single copy of each file is located at a specified node. The data center node is arbitrarily selected, but ordinarily is one of the storage nodes near the geographic center of the collection of nodes.

Having designed a centralized data network the next step is to relax the centralization constraint. Holding the network fixed we seek the optimal location strategy for each file in turn. To do this we find, using a previously reported algorithm,[14] the file assignment policy that minimizes the product of distance times bits transmitted, summed over all links of the network, and added to file maintenance costs. New link capacities are computed for this configuration and further improvements in file location are obtained by trying local perturbations in file location (sequentially moving each file copy to a neighboring location and recomputing cost). The configuration having least cost is retained, and the algorithm now attempts to redesign the network topology while holding the files in their newly determined locations. At this point the problem is not one in centralized network design, for the files are presumably scattered across storage sites. The network design problem is essayed by means of a sequence of local transformations.

The transformations used are similar to those employed in the Esau-Williams algorithm. At each step a candidate node is connected to one of its neighbors on a trial basis, and an existing connection from this node is broken. Establishment of the new link creates a circuit in the network (see Figure 3). The link to be removed is the one that completes the return path to the candidate node. When this is done the network is a tree once more and a new distribution of traffic flow and resulting line cost can be computed. Because the simple link exchange described does not generally alter the flow pattern everywhere in the network, the new traffic parameters can be calculated rapidly.

During one design iteration all network nodes are tested for possible reconnection. The link exchange that results in the greatest cost reduction is put into effect, and the process is repeated until no such improvement is found.

A somewhat more general link exchange procedure[15] is to determine the network cost that results when each link along the circuit of Figure 3 is broken. In this way a family of local transformations is tested rather than only the one-for-one exchange described above. The Esau-Williams algorithm does not do this, nor has it been implemented in the experiments reported here, but the conception is worth exploring in future work.

Link Exchange

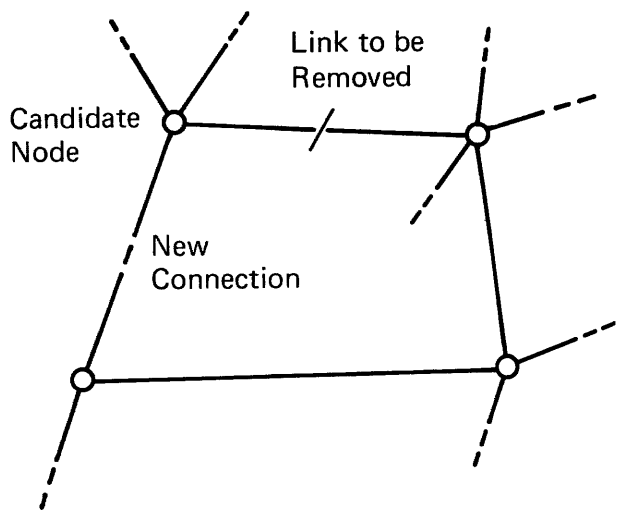
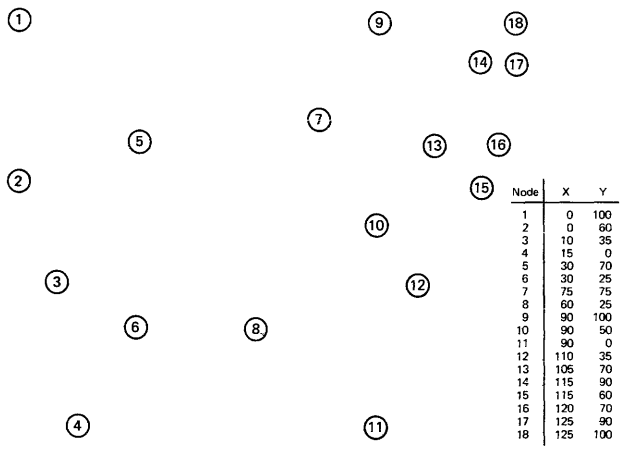


Figure 3—Link exchange



Node	X	Y
1	0	100
2	0	60
3	10	35
4	15	0
5	30	70
6	30	25
7	75	75
8	60	25
9	90	100
10	90	50
11	90	0
12	110	35
13	105	70
14	115	90
15	115	60
16	120	70
17	125	90
18	125	100

Figure 4—Sample node locations

EXPERIMENTS

In order to test the design algorithm an artificial, 18-node, 2-file example has been constructed. Each user node was considered to be a potential storage site as well in this example. Figure 4 shows the geographical distribution of the nodes, while Figure 5 depicts the volume of user query and update transactions with each of the files. The traffic parameters were chosen randomly with overall ratio of queries to updates selected to favor multiple copies in the network.

The allowable link capacities are shown in Figure 5. The cost c_{ij} (k) of leasing a link having the k^{th} capacity and connected from node i to node j is assumed to be a linear function of the internode distance, a relationship characteristic of present-day tariff rates, although in no way required by the model. Storage costs were neglected in this example.

Available Line Capacities

0 10 16 30 50 96 192

Line Costs:

Cost = A + B x Dist

Dist is the distance (miles) between nodes.

A & B are functions of capacity as follows:

Capacity	A	B
0	0	0
10	55	90
16	100	120
30	115	150
50	250	225
96	340	300
192	400	450

Nodal Traffic

File No. 1

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Query	16	11	3	11	16	1	9	15	7	8	6	5	18	6	10	12	3	5
Update	0	2	1	2	0	2	1	2	2	0	2	2	2	0	2	2	0	0

File No. 2

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Query	9	9	5	9	4	0	2	9	9	0	0	0	9	1	5	9	3	2
Update	0	2	0	0	2	2	2	2	1	0	2	2	1	2	1	2	1	1

Figure 5—Cost and traffic parameters

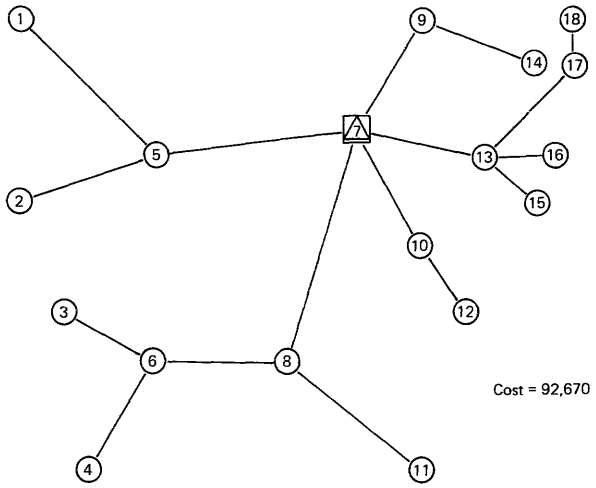


Figure 6—Centralized network

The design routine shown in Figure 2 was programmed in the APL language. Figure 6 shows the centralized network obtained using the Esau-Williams algorithm. At this point both files are located at node 7; however, a test with the file allocation algorithm indicates that the total bit-miles of network traffic would be minimized by adding a copy of file 1 at node 8. When this is done the actual line costs increase, from 92,670 to 97,300. Upon applying the link exchange algorithm to the centralized network a series of node reconnections is performed, leading to the network of Figure 7. In all, the distributed tree provides a savings of about 1600 compared with a centralized net. The savings (about 2 percent in this case) achieved by distributed design tends to increase as the volume of query-type messages increases relative to the update load.[14]

Several other file locations that scored well in the allocation experiment were also tried with the Esau-Williams net and the link exchange program. All these trials yielded more expensive networks.

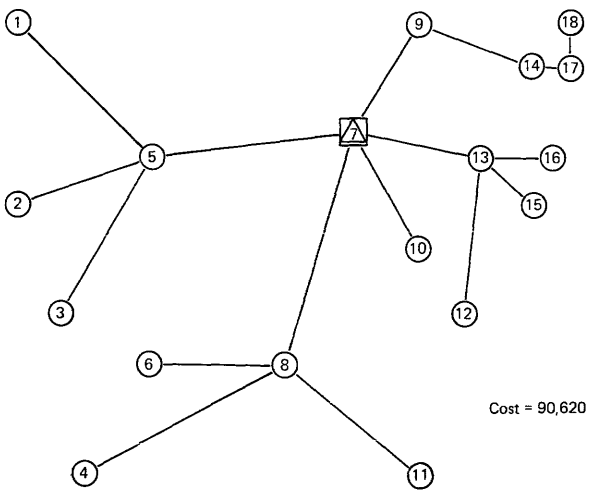


Figure 7—Distributed network

CONCLUSIONS

The problem of locating information resources and choosing a topology for a network of distributed data files has been formulated in a model that, while simple, retains important practical features such as discrete capacity assignment, economy of scale, and distinction between query and update transactions. Even for a simplified model it appears that heuristic approaches to design are required, and a sample algorithm has been formulated and tested for the special case of tree design. The technique can be viewed as an extension of a conventional method for automatic synthesis of centralized data nets, and seems to offer a useful reference point against which to compare future investigations in this area.

The algorithm presented here should be considered an initial attempt in the design of distributed data networks, and not a final answer. It is primarily a test of the concept of alternating between processes for file allocation and topology design. The experiments show that network cost reductions can be achieved in this manner. On the other hand the two sets of variables interact strongly. When a network has been designed for a given disposition of data the topology obtained is bound to determine the locations at which the files can be gainfully redeployed. We have included a complete description of our example for the benefit of researchers interested in trying alternative approaches and comparing results.

ACKNOWLEDGMENT

This work was supported by the Office of Naval Research under contract number N00014-72-C-0299.

REFERENCES

1. Frank, H., Frisch, I. T., *Communication Transmission and Transportation Networks*, Addison Wesley, 1972.
2. Kleinrock, L., "Analytic and Simulation Methods in Computer Network Design," *Proceedings AFIPS*, SJCC, May 1970.
3. Knight, J., "A Case Study—Airline Reservation Systems," *Proceedings IEEE*, Vol. 60, No. 11, pp. 1423-1431, November 1972.
4. Crocker, S., et al, "Function Oriented Protocols for the ARPA Computer Network," *Proceedings AFIPS*, SJCC, May 1972.
5. Astrahan, M., Altman, E., Fehder, P., Senko, M., "Concepts of a Data Independent Accessing Model," *Proceedings ACM SIGFIDET Workshop*, Denver, November 29-30, 1972.
6. Gould, R. G., "Comments on Generalized Cost Expressions for Private Line Communications Channels," *IEEE Trans. on Comm. Tech.*, pp. 374-377, September 1965.
7. Frank, H., Frisch, I., Chou, W., "Topological Considerations in the Design of the ARPA Computer Network, *Proceedings AFIPS*, SJCC, May 1970.
8. Schwartz, M., et al., "Terminal-Oriented Computer Communication Networks," *Proceedings IEEE*, Vol. 60, No. 11, pp. 1408-1423, November 1972.
9. Friedman, T., "A System of APL Functions to Study Computer Networks," presented to *National Computer Conference* 1973.
10. Esau, L., Williams, K., "A Method for Approximating the Optimal Network," *IBM Systems Journal*, Vol. 5, No. 3, 1966.
11. Whitney, V. K., "Comparison of Network Topology Optimization Algorithms," *Proceedings ICCC*, Washington, D. C., October 1972.
12. Chandy, K. M., Russel, R. A., "The Design of Multipoint Linkages in a Teleprocessing Tree Network," *IEEE Transactions on Computers*, Vol. C-21, No. 10, October 1972.
13. *Communications Network Design Program/360, Service Description Manual*, IBM Data Processing Division, IBM Tech. Publications Dept., January 1970.
14. Casey, R. G., "Allocation of Copies of a File in an Informatin Network," *Proceedings AFIPS*, SJCC, May 1972.
15. Frank, H., et al, "Optimal Design of Centralized Computer Networks," *Networks*, Vol. 1, pp. 43-57, 1971.

APPENDIX

Symbolic description of the model
given data

processor nodes: $k=1, 2, \ldots, n$
users: $w=1, 2, \ldots, n$
files: $a=1, 2, \ldots, r$
query traffic between wth user and ath file: Q_{wa}
update traffic " " " " " ": U_{wa}
fixed cost of storing and maintaining a copy of the ath file at the kth processor node: S_{ka}
admissible link capacities: c_t, $t=0, 1, 2, \ldots, h$ (where $c_0=0$)
cost of a link of the tth capacity and connecting nodes i and j: $d_{ij}(t)$
(in particular $d_{ij}(0)=0$)

variables

$X=\{x_{ij}: i, j=1, 2, \ldots, (m+n)\}$

where $x_{ij}=0, 1, 2, \ldots, h$ is the capacity index assigned to the link between nodes i and j.

$Y=\{y_{ka}: i=1, 2, \ldots, n; a=1, 2, \ldots, r\}$

where $y_{ka}=1$ if a copy of the ath file is stored at the kth node

$y_{ka}=0$ otherwise

$$\begin{bmatrix} q_{ij}(a, w, k) \\ u_{ij}(a, w, k) \end{bmatrix} = \begin{bmatrix} \text{query} \\ \text{update} \end{bmatrix}$$ traffic leaving node i on link (i, j) from wth user and directed to a copy of the ath file located at the kth node.

note: $q_{ij}=-q_{ji}$

$u_{ij}=-u_{ji}$

The network optimization problem is to specify the network variables X and Y in such a way as to minimize the cost function:

$$D(t, Y)=\sum_{a, i} y_{ai}s_{ai}+\sum_{i, j} d_{ij}(x_{ij})$$

subject to the following constraints.

(1) delivery of messages and conservation of messages.

(a) no messages lost or gained at an intermediate node.

$$\sum_j q_{ij}(a, w, k) = 0$$

each i, a, w, k such that $i \neq w, i \neq k$

$$\sum_j u_{ij}(a, w, k) = 0$$

(b) total query traffic from wth user to all copies of file a.

$$\sum_{j,k} q_{wj}(a, w, k) \cdot y_{ka} = Q_{wa} \qquad \text{each } w, a$$

(c) total update traffic from wth user to each copy of ath file.

$$\sum_j u_{wj}(a, w, k) = U_{wa} \cdot y_{ak} \qquad \text{each } a, w, k$$

(d) absorption of query traffic at file nodes

$$\sum_{k,i} q_{ik}(a, w, k) \cdot y_{ka} = Q_{wa} \qquad \text{each } w, a$$

(e) All updates transmitted to each file copy.

$$\sum_i u_{ik}(a, w, k) = U_{wa} \cdot y_{ka} \qquad \text{each } a, w, k$$

(2) Capacity constraint: no overloaded links.

$$\sum_{a,w,k} [\,|\, q_{ij}(a, w, k) \,|+|\, u_{ij}(a, w, k) \,|\,] \leq c_{xij} \quad \text{each } i, j$$

Specifications for the development of a generalized data base planning system

by J. F. NUNAMAKER, JR., D. E. SWENSON and A. B. WHINSTON

Purdue University
West Lafayette, Indiana

INTRODUCTION

Societal problems and industrial problems, if they are to be studied analytically, will require two broad components: first, a structure or model to analyze the interaction of the variables; and second, a large scale data base. The data base may be used in various ways, such as validation of the model's parameters, input data for actual runs of the model, and simply providing the data itself for other interests. While the ability to devise meaningful models with appropriate supporting data is of primary importance for the advancement of our capacity to serve societal and industrial needs, the possibility of integrating the data base handling techniques with techniques of simulation and optimization will greatly facilitate this work.

With the advent of large scale computers and complex operating systems during the last six years, the capacity exists for the processing of large scale applications. In addition, much work has been done with respect to the development of Generalized Data Management Systems (GDMS), which were designed to handle and manage large data bases. Typically, such systems have been used by management of large industrial and military organizations. These systems are used for handling inventory, receivables, customer accounting and billing, quality control and other administrative tasks.

Generalized Data Management Systems have contributed to increased use of computers, but a major void still exists. The problem of many application programs interacting automatically with a single data base remains a major barrier to general use of an information system as a planning system.

The use of an information system as a planning system will come about only when a methodology exists for automatically creating the data files needed by the many application programs and answering general queries of the data base at the same time. What is needed is a software system that, as a result of a specific query from a user, can: (1) retrieve the data necessary to answer the query; and/or (2) set up the application program or model that must be run to answer the query. This requires that the data be automatically retrieved and arranged in the proper format required by the model.

The value of such a software system is based upon: (1) the efficiency of storing and retrieving data; and (2) the range of services provided through the interactive query system.

The planning system must be designed so that the user is freed from the mundane task of data preparation, which can be tedious and frought with human errors, in order to run a model. Often, the user is not familiar with the problems and procedures of data handling, and, in most cases, would prefer not being bothered with the data handling at all.

A user who has just queried a data base will have gained very little if he must further select, re-arrange, reformat, and punch his retrieved data for input to an application program or model. Thus what is needed is a Generalized Data Base Planning System (GPLAN), which results from the extension of a Generalized Data Management System to handle the automatic setup of models from a Data Base as instructed by the user through the Query Language. GPLAN is a natural extension of GDMS and represents the next generation of GDMS's.

In many areas, there has been considerable work on development of simulation and optimization packages, with the end result that these packages are not really used by the people who should be using them. The reasons for this huge investment in application packages with little resultant usefulness are simple. The packages are so difficult to use, requiring either very much technical knowledge in the particular mathematical programming technique and/or complex file setup and manipulation steps, that few people are able to use the package without relying heavily on technical help or considerable education in the specifics of each particular package.

It is obvious that to increase the usefulness of simulation and optimization packages, the nontechnical and/or management personnel who are knowledgeable in the general area of endeavor should be able to easily use these packages.

What is proposed here is to generalize data base planning systems. For a specific area of human endeavor needing such a system, this would mean that it would be easy to set up a data base, link it with application packages, and query it in a meaningful manner—in short, it

would be easy to set up a generalized planning system. This would mean that ad hoc solutions to specific areas would be replaced with a generalized approach comprehensive enough to solve many of the problems occurring in setting up specific planning systems.

In order to take advantage of the knowledge that can be gained from the consideration of specific systems, a regional water pollution control planning system is described briefly and used for some examples throughout the rest of the paper. This system is discussed in order to develop the proper motivation for a Generalized Planning System.

WATER POLLUTION CONTROL PLANNING SYSTEM

The purpose of the water pollution control planning system is to develop a plan that minimizes the cost of pollution abatement structures while satisfying a set of water quality goals throughout an entire river basin. This planning system uses the most prevalent measure of water quality in use today, the level of dissolved oxygen concentration.

The constraints of this model are constructed by dividing the river into sections and constraining the water quality, interpreted as the dissolved oxygen deficit level, to be met at the end of each section. Starting from the headwaters of the river, new sections are defined whenever the river parameters change significantly, such as effluent flow entering the river, incremental flow entering the river (tributary flow, ground water, etc.), the flow in the main channel being augmented or diverted, or the parameters describing the river changing gradually over a longer distance.

The quality constraints are sequentially dependent in that the quality in each section is a function of the quality in the preceding section. But the possibility of tributaries, flow augmentation, and incremental and effluent flows entering at downstream points, complicates the relationship between the constraints.

Three possible treatment techniques are allowed for in the model: (1) by-pass piping; (2) regional and on-site treatment plants; and (3) flow augmentation. Thus piping flows are allowed from each polluter to each river section, from each polluter to each treatment plant, and from each treatment plant to each river section.

In addition to quality constraints, flow conservation constraints are needed for both the polluters and the treatment plants.

The solution technique of the model is the use of a general purpose non-linear algorithm adapted for this model. The major problem involved in adapting the algorithm was the calculation of the partial derivatives of both the constraints and the objective function. These partials were necessary to set up a local Linear Programming problem to determine the direction of search in the stepwise nonlinear problem. Starting with a point in the domain of the objective function, a new point is calculated from it by making a step to either reduce the value of the objective function, if the original point is a feasible solution to the nonlinear programming problem, or obtain a "more feasible" solution if the point is an infeasible solution ("more feasible" by reducing the value of the most infeasible constraint).

The cost function essentially involves the costs of new or upgraded treatment plants, reservoirs for flow augmentation, and various costs of new pipes to or from the polluters, treatment plants, and river sections.

DIFFICULTY WITH THE PRESENT APPROACH

Let us take an example of solving a mathematical programming problem (although any complex application problem would be similar), such as the nonlinear programming model discussed in the water pollution control planning system. A programmer with knowledge of mathematical programming theory, or a group of people which together has the required knowledge, would write and check out a program to solve the specific problem. Then data would be gathered and stored in the format necessary for the program. To check out the data, separate programs would have to be written to test each part of the input data file for which testing was needed. Several sets of corrections to the data file would probably need to be made.

The programmer(user) would then have to fill out some control cards giving various parameters of his data. Obtaining these parameters might involve some manual calculations and may require running some other programs on the original data.

Finally, the mathematical model could be run and, with several iterations and interpretations by the knowledgeable mathematical programming person, the problem could be solved. A report would have to be written explaining the problem and its computer solution. To solve a similar problem would take almost the same steps, except that a lot of the existing programs could probably be used, although major revisions are also possible.

The existing data on a file for solving this problem is most likely usable only for this application, even though parts of it may be usable for other applications.

The time lag between problem recognition and problem solution will be too long and the cost will be expensive. It is possible that, without countless hours of detailed documentation, the program written is unusable except for one or a few people.

If we consider the water pollution control planning system, set up as an application program with a file, the information on treatment plant cost data and the parameters of the rivers cannot be used easily by anyone other than the developers of the system, unless the new user knows the specific format of the data and how it is stored on auxiliary memory.

MOTIVATION FOR THE DEVELOPMENT OF GENERALIZED DATA BASE PLANNING SYSTEMS

We must define what a Generalized Data Management System is, before we define the characteristics of a Generalized Data Base Planning System.

Groner and Goel[3] define and characterize a Generalized Data Management System as follows:

"A GDMS consists of data, structure, and a set of algorithms for manipulating the data and the structure. It acts as a communication channel between the user community and the data base. Minker[4] characterizes a GDMS as follows: 'A data management system is considered generalized when it permits the manipulation of newly defined files and data with the existing programs and systems.' [A GDMS] facilitates reference to data by name and not by physical location,' and, '[it] facilitates the expression of logical relations among data items.' A well designed GDMS permits users to access and manipulate elements of the data base in a way that is both natural and convenient for them and efficient in terms of its system utilization.

"Much of the benefit derived from a GDMS results from the insulation of the people and programs from the data. In conventional systems the structure of the data must be explicitly embodied in each program accessing the data. This limits the application of programs to data whose structure has been defined to them. It also limits the ability to restructure data in response to new needs without modifying every program embodying the old structure. These limitations in applicability of programs and the flexibility of the data base impose a rigidity upon conventional data processing systems. While this rigidity is not serious in repetitive well defined tasks such as payroll or inventory control, it is a decided obstacle to the successful performance of systems that must respond to continually changing requirements.

"GDMS systems evolved from sequential formatted file systems. This evolution has been in the direction of more complex logical data structures and more complex operations upon them."[11]

The work on Generalized Data Management Systems until now has focused on two related, but distinctly different approaches to the design of data management systems.[5] The first approach involved the design of a special query system for retrieving data from a data base. This approach permitted new programmers to readily access data and to ask sophisticated questions of the data base. The query capability made this approach very popular, but it has a serious drawback. Namely, that it is extremely difficult to process other applications written in FORTRAN, COBOL, PL/1, etc., against the data base. Examples of the first approach are Informatics Mark IV and GIS of IBM. As a result of this deficiency, many people preferred the second approach which involved extensions to host languages (FORTRAN, COBOL, etc.) that gave the programmer some general file handling capability. However, the non-programming user found this approach undesirable, since, if he wanted a query answered, he had to write the program(s) in the host language to retrieve the data. Examples of the second approach are General Electric's IDS (Integrated Data Store) and Burrough's Disk FORTE (Disk File Organization Technique).

Now the CODASYL Data Base Task Group[6] has proposed a solution to the problem, but in its specifications has given first priority to the host language approach, with COBOL as the first language. To interface between a host programming language and a data base, a specific subschema Data Description Language and an appropriate Data Manipulation Language need to be provided. When an interface is provided, applications still must be implemented in the specific host language system. (While this may not be a problem when and if the standardized implementation of CODASYL DBTG concepts occurs throughout the industry for the major higher level languages, it is quite a drawback for several years to come.) Even if CODASYL's concepts were all implemented, we are still without a system that a planner or manager could easily use. GPLAN is proposed as such a system.

GPLAN is a synthesis of components from other systems. There are a number of systems that exhibit some, but not all, of the features of GPLAN. Examples of some of these systems are:

1. NAPSS: Numerical Analysis Problem Solving System.[7]
2. SODA: Systems Optimization and Design Algorithm.[8]
3. GDMS: Generalized Data Management Systems.[4,5,6]
   (a) System 2000—MRI[9]
   (b) RAMIS—Mathematica, Inc.[10]
   (c) IMS—IBM[11]
   (d) DISK FORTE—Burroughs Corporation[12]
4. OPTIMA.[13]

NUMERICAL ANALYSIS PROBLEM SOLVING SYSTEM (NAPSS)

A system that is a special case of GPLAN is the Numerical Analysis Problem Solving System (NAPSS) designed and implemented at Purdue University.

A long recognized goal of Computer Science has been to facilitate the stating of problems in languages appropriate to the specific fields in which the problems exist, and then to provide for this solution without the services of highly trained programmers and analysts. These systems are "problem solving systems" and their languages are problem-oriented languages. Thus the aim of the NAPSS project is to make the computer behave as if it had some of the knowledge, ability, and insight of a professional numerical analyst.

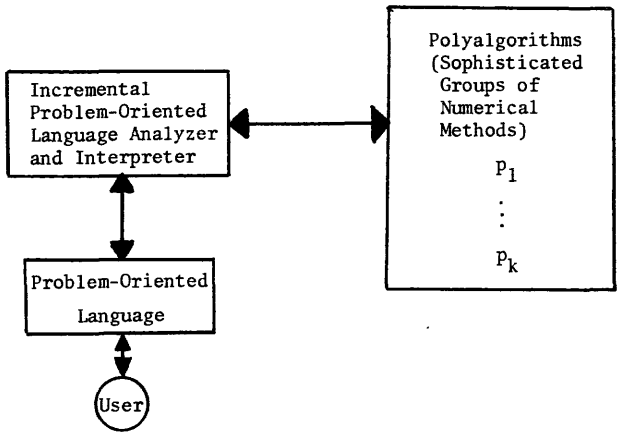


Figure 1a—NAPSS—Numerical Analysis problem solving system

A user unskilled in numerical analysis can describe relatively complex problems in a simple mathematical language. Then the system selects algorithms, performs analyses, and gives diagnostics of possible difficulties and meaningless results.

The problem-oriented language of NAPSS uses some of the applicable notation of Fortran, Algol, and PL-1, since they have quite similar facilities for describing computational algorithms. But it goes beyond these languages to include mathematical concepts such an integration, differentiation, algebraic and differential equations, and approximation as part of the basic language.

The basic approach to the system design is through the development of polyalgorithms which become the numerical analysis packages that are the essential elements of the problem solving system. "A polyalgorithm is formed by the synthesis of a group of numerical methods and a logical structure into an integrated procedure for solving a specific type of mathematical problem."[1] The goal of a polyalgorithm is to combine a number of algorithms (corresponding to numerical methods) with a strategy for their selection, and use a procedure which is relatively efficient and very reliable.

The NAPSS system exists as an extension of a procedure-oriented language in an environment, permitting both on-line and remote use of the system. While NAPSS operates most frequently in an online time-sharing environment, it will accept programs submitted for batch processing. Parameter values that are needed by NAPSS can be entered at a terminal in conversational mode or be present on a standard or user-named input file for either conversational or remote mode.

In Figures 1a and 1b we can see how NAPSS can be put into a more general structure with a renaming of its components.

SODA (SYSTEMS OPTIMIZATION AND DESIGN ALGORITHM)

SODA is a computer-assisted decision making system for the design of information processing systems. SODA generates a complete information systems design, along with cost/performance projections of how the designed system will perform on a specified hardware/software configuration. SODA consists of four major components:

SSL:SODA STATEMENT LANGUAGE
SSA:SODA STATEMENT ANALYZER
SGA:SODA GENERATOR OF ALTERNATIVES
SPE:SODA PERFORMANCE EVALUATOR

SSA is a computer program that analyzes the requirements of an information processing system stated in SSL. The Statement Analyzer also provides feedback information to the user to assist him in achieving a better problem statement.

SSA also produces a number of networks which record the interrelationships of processes and data and passes the networks on to SGA and SPE.

Each type of input and output is specified in terms of the data involved, the transformation needed to produce output from input and stored data. Time and volume requirements are also stated. SSA analyzes the statement of the problem to determine whether the required output can be produced from the available inputs. The problem statement stored in machine readable form is processed by SSA which:

1. checks for consistency in the Problem Statement (PS) and checks syntax in accordance with SSL; i.e., verifies that the PS satisfies SSL rules and is consistent, unambiguous, and complete.
2. prepares summary analyses and error comments to aid the problem definer in correcting, modifying and extending his PS.
3. prepares data to pass the PS onto SGA, and
4. prepares a number of matrices that express the interrelationship of Processes and Data Sets.

SGA is a procedure for the selection of a Computer System (cpu, core size, auxiliary memory devices) and the specification of alternative designs of program structure and file structure. SGA constructs a configuration of equipment in order to evaluate performance of the system. A number of models are used (to compute timing estimates) that select timing factors for alternative hard-

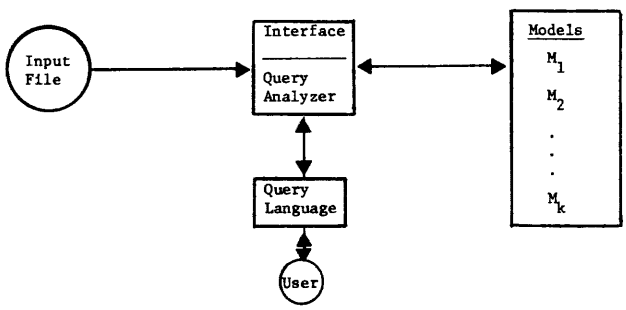


Figure 1b NAPSS GPLAN terminology

ware/software configurations from a data file. SGA simulates the jobstream as it would be processed on the selected configuration, and, using the factors from the hardware/software library, SGA and SPE produce detailed cost/performance projection reports so that the user can evaluate the final design.

There are a number of systems similar to some aspects of SODA, such as SCERT[14,15] and CASE.[16]

In Figures 2a and 2b it is shown how SODA fits into a more general structure.

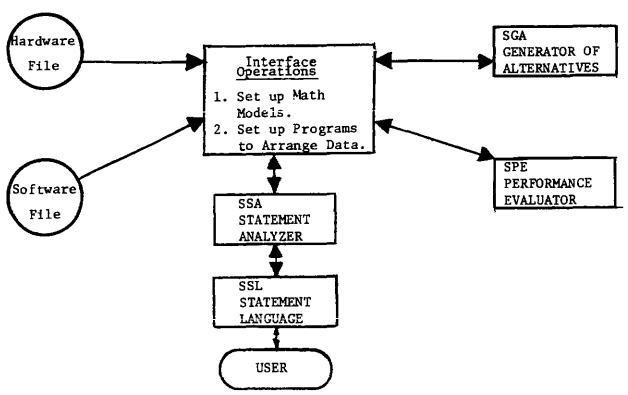


FIGURE 2a—SODA: Systems optimization and design algorithm

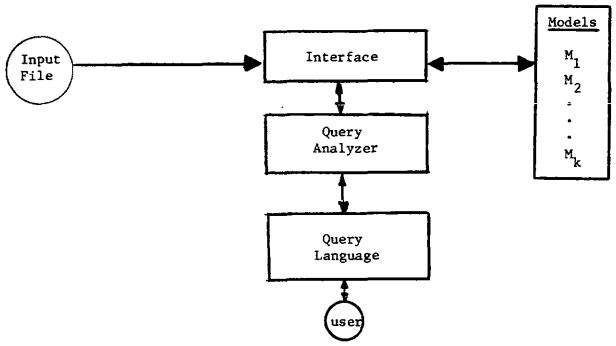


Figure 2b—SODA—GPLAN terminology

GENERALIZED DATA MANAGEMENT SYSTEMS

SYSTEM 2000

SYSTEM 2000, developed by MRI Systems Corporation, is a general-purpose data base management system. The basic system provides a comprehensive set of data base management capabilities, including the ability to define new data bases, modify the definition of existing data bases, and retrieve and update values in these data bases.

In SYSTEM 2000, the basic components of data base definitions are data elements and repeating groups. Values are stored in data elements. Repeating groups describe a structure for storing multiple sets of data values (data sets) and also serve to link hierarchical levels of the definition.

Values for each element and logical entry (record) may vary in length. The user may specify without restriction which elements in the data base are to be inverted and become key fields, and what hierarchical relationship an element will have with other elements in the data base. Data security is maintained by password control to the data base and additional password control to each component.

The Procedural Language feature of SYSTEM 2000 enables users to manipulate data in a SYSTEM 2000 data base from COBOL or FORTRAN. This feature provides the mechanism to address any part of the data base of interest to the procedural program, to retrieve data in a sequence and format suitable for procedural processing, and to update the data base from the program.

RAMIS: RANDOM ACCESS MANAGEMENT INFORMATION SYSTEM

RAMIS, developed by Mathematica, Inc., is a data base management system which permits a user to describe and build data bases, maintain the data in the data bases through updates, additions, and deletions, retrieve information from the data bases and display it in meaningful report formats, or pass the information to other processing programs.[4]

RAMIS is both a report generator and a data management system, since it has a simple and logical English-like language, which permits the user to both request information from data bases, and, at the same time, process it into finished reports.

RAMIS organizes the physical placement of data into tree structures on random access devices by exploiting the hierarchical relationships of the data fields. The user has to supply only some minimal information about these relationships. User written programs in Fortran, Cobol, Assembler, or PL/1 can also be linked directly into RAMIS.

IMS: INFORMATION MANAGEMENT SYSTEM

The Information Management System (IMS) is a system designed to facilitate the implementation of medium to large common data bases in a multi-application environment.[5] This environment is created to accommodate both online message processing and conventional batch processing, either separately or concurrently. The system permits the evolutionary expansion of data processing applications from a batch-only to a teleprocessing environment.

The data base processing capabilities of IMS are provided by a facility called Data Language/I. The data base functions supported are definition, creation, access, and maintenance. The full data base capabilities of Data Language/I can be used in the IMS batch processing or teleprocessing environment.

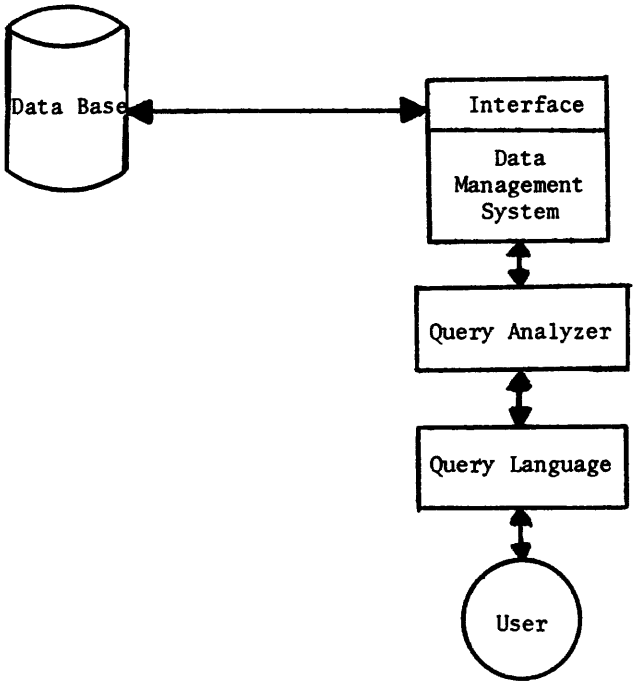


Figure 3—Generalized data management system

Data communication capabilities are characterized by the use of input/output terminals in remote and local environments, connected to the computer, which provide the user with access to the data base. IMS also has extensive message scheduling, checkpoint, and restart facilities.

DISK FORTE

Disk FORTE, the Burroughs manufacturer system, is programmer-oriented at the most basic level. Nearly all features and capabilities of other data management systems must be programmed in Disk FORTE. Yet, it permits both hierarchic and network data structures (user-programmed, of course) which make possible more complex associations among data.

Disk FORTE makes its data management capabilities available through extensions to COBOL which are handled by a pre-compiler.

Figure 3 shows the generalized structure of SYSTEM 2000, RAMIS, IMS and DISK FORTE.

OPTIMA

OPTIMA is an advanced mathematical programming system for the CDC 6000 series computers. It includes advanced algorithms and techniques in addition to algorithms for standard linear programming formulations. User-controlled data and storage management features are also provided.

"The basis of OPTIMA is a revised product-form, composite, bounded variable, separable, multipricing, simplex, linear programming algorithm." Some of the advanced features that OPTIMA provides are: the capability to form a nontrivial starting basis; the ability to start a solution using a previous basis; dynamic control of the frequency of inversion of the basis matrix; provision for partial and multiple pricing; the use of maps to exclude or include specified vectors in the basis; and elaborate recovery procedures. A dual optimization algorithm is available for those problems in which its use might be advantageous; and postoptimal analysis of a problem can be accomplished as an integral part of OPTIMA.

Through the use of the Applications Control Language (ACL), OPTIMA allows dynamic control of the progress and execution of the program. ACL has logic and computational capability and provides verbs and phrases for modifying various parameters and controlling the progress of the solution.

An ACL program must be written for any study. This program defines the data files to be used and the operations to be performed on these files, sets any parameters and controls necessary, and calls various routines required to carry out the study.

Two other languages, the Matrix Generator Language (MGL) and the Report Generator Language (RGL) operate within control of the ACL. MGL provides capabilities for generating a problem matrix automatically. RGL provides the capability for generating reports in any desired format, and permits computer-generated solutions to be used for further arithmetic and logical computation.

SUMMARY OF COMPARISONS

Figure 4 shows the structures of GPLAN. In considering the four information processing systems (1) NAPSS,

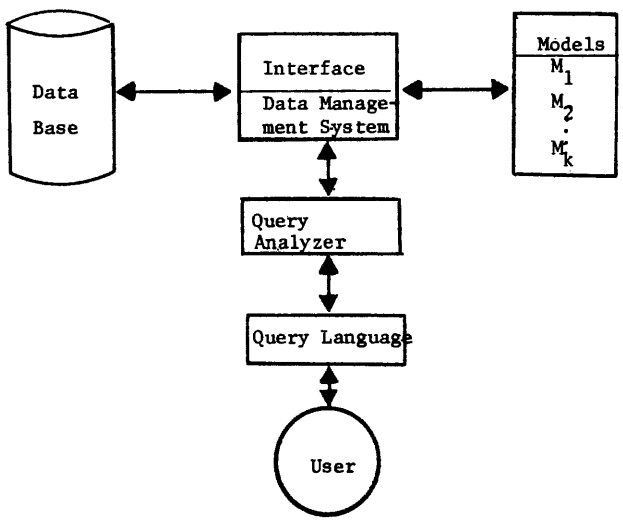


Figure 4—GPLAN—Generalized data base planning system

TABLE 1—Comparison of information processing system

CHARACTERISTICS	DISK FORTE	GPLAN	IMS	SODA	NAPSS	OPTIMA	RAMIS	SYSTEM 2000
Mnemonic value of name	DISK File ORgani-zation TEchniques	Generalized Data Base Planning System	Infor-mation Manage-ment System	Systems Optimi-zation Design Algorithm	Numerical Analysis Problem Solving System	.....	Rapid Access Management Information System	.....
Implementers	Burroughs		IBM	Purdue U.; University of Michigan	Purdue University	Control Data Corporation	Mathematica, Inc.	MRI Systems Corporation
Hardware	B2500, B3500	CDC 6500 IBM 370	IBM 360, 370	UNIVAC 1108 IBM 360/67 CDC 6500	CDC 6500	CDC 6000 series	IBM 360,370	IBM 360,370 UNIVAC 1106,1108 CDC 6000,Cyber 70
What is the system?	Manufac-turer's DBMS	Planning System Generator	Manufac-turer's DBMS	Information System Design	Numerical Analysis System	Mathemati-cal Program-ming System	Nonmanufac-turer's DBMS	Nonmanufacturer's DBMS
System Imple-mentation Language	COBOL	FORTRAN BAL/COMPASS	BAL	FORTRAN	FORTRAN COMPASS		FORTRAN BAL	FORTRAN BAL
User Language	COBOL extensions	OWN	COBOL, PL/1,BAL	OWN	OWN	OWN	OWN, COBOL, FORTRAN, PL/1, BAL	Assembler, COBOL, FORTRAN
File Organization (Storage Struc-tures - under user control if visible to user)	Unordered Random Index-Ran-dom, Index-Sequential (visible to user)	Special File Struc-ture	Sequential Index-Sequential Direct access (visible to user)	Sequential	Direct Access	Sequential	Special File Structure	Special File Structure
Data Structures Allowed	Hierarchic or Network	Hierarchic or Network	Hierarchic	Sequential	Network	Sequential	Trees	Hierarchic
Users: Nonprogramming		X		X	X		X	X
Programming	X	X	X		X	X		X
User Language Classifications:								
Tabular	X			X				
Command			X	X	X	X		X
Free		X					X	
Procedural	X		X		X	X		X
Nonprocedural		X		X	X		X	
Query Language and Analyzer:								
Data Base Queries Allowed?	X (user programmed)	X	X				X	X
Application Pack-age Queries Allowed		X		X	X	X		
Problem Oriented Language		X		X	X	X		
Application Pack-ages or Models:								
Mathematical and/ or optimization models		X		X	X	X		
Statistical Packages		X						
Simulations		X		X				
Any Program or Group of Intercon-nected Programs	COBOL (user programmed)	X						
Comparison with GDBPS Components:								
Data Management System	X	X	X				X	X
Data Base	X	X	X	X			X	X
Models		X		X	X	X		

(2) SODA, (3) GDMS, and (4) OPTIMA, it is observed that we must have at least a query language, a query analyzer and a Generalized Data Management System. NAPSS and SODA are missing the data management capabilities; the GDMS's are missing those components needed to readily interface models; and OPTIMA is missing the data management and query capabilities. These differences are elaborated on in Table I.

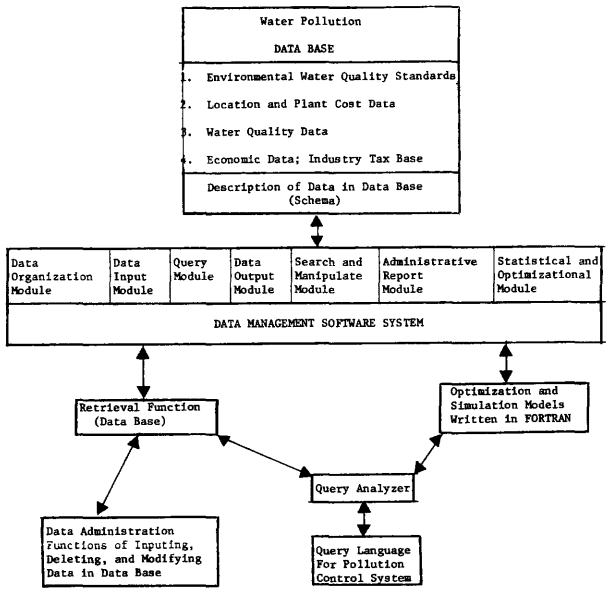


Figure 5—GPLAN for water pollution control

COMPONENTS OF A GENERALIZED DATA BASE PLANNING SYSTEM

A synthesis of the previous systems results in the definition of the following nine components of a Generalized Data Base Planning System:

- A Generalized Data Management System (GDMS)
- Raw Data for the Data Base
- A Query Language
- A Query Analyzer
- A Collection of Application Packages or Models
- Administrative Report Module
- User's Interface
- Extraction Files
- Users

Each of these components is discussed in more detail in the following sections. An overview of GPLAN for Water Pollution Control is shown in Figure 5 and Figure 6.

A generalized data management system (GDMS)

A Generalized Data Base Management System that is implemented at a particular installation under a specified operating system must be available. The GDMS must meet minimum requirements as to data structure definition, data base loading, data base updating, and data base retrieval, and it must satisfy some minimum set of queries, as specified in the following section.

Six general functions must be provided by the GDMS:

Input—the system accepts data values or information about data structures.

Search—the system searches the data base by examining the descriptions of data structures and storage structures to ascertain the existence and location of certain data values.

Storage—the system accesses a data base to add, insert, modify, or delete data values.

Maintenance—the system generates or modifies descriptions of data, data structures, and storage structures to adapt to change.

Retrieval—the system accesses a data base to obtain data values previously stored.

Output—the system exhibits data values or information about data structures and storage structures.

Raw data for a data base

The raw data for the planning system must be available in whatever form it can be collected. A Data Input Module is used to convert the raw data into a form necessary to be loaded by the GDMS into the data base.

To refer to the data in the data base, the following terms, adapted from the CODASYL Data Base Task Group Report, are defined:

DDL—Data Description Language. A language for defining data and their relationships. The DDL is divided into schema and sub-schema.

Schema—That part of the DDL which defines the "universal" data base.

Sub-schema—That part of the DDL which describes the data known to each application program or model.

A schema describing the data on the data base must be prepared. (Note that the DDL is not necessarily the one defined in the DBTG report.)

While preliminary Data Input Modules would be dependent on the format of the raw data and dependent on the GDMS, it is hoped that some measure of independence can be achieved in the same manner as the data transformations used in implementing the user interface mentioned later.

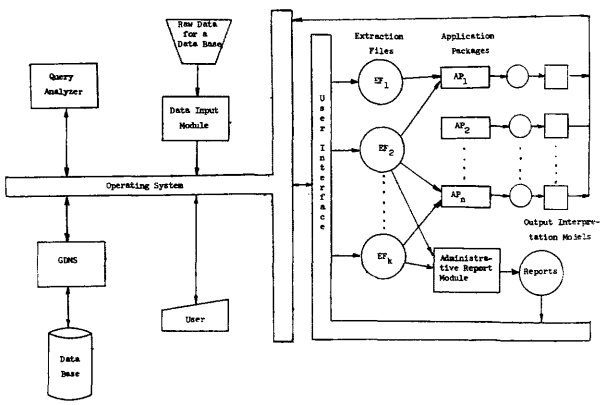


Figure 6—GPLAN—Generalized data base planning system

A query language

A Basic Query Language (BQL) defines all those queries that can be handled by the GDMS alone and allows the user to request that the DBMS display certain data or answer questions about the data. The BQL is a comparative and computationally oriented language used to compare item values with other item values, or with constants, or with results of computations. Arithmetic operators define the computations to be performed, and logical operators combine simple expressions into compound expressions. The BQL can be extended as application packages are added to include questions that are answered by the new application packages, i.e., each package adds a set of new query components to the BQL. The BQL, together with all the query components from the applications packages, makes up the Query Language (QL).

General capabilities provided by the QL are:

Selective retrieval—the user specifies the selection conditions to be satisfied in retrieving the desired data.
Nonselective retrieval—the user specifies unconditional retrieval of data.
Conditional retrieval—the user employs verbs such as IF ... ELSE to test items for some qualifying values in determining alternative courses of action.
Statistical retrieval—the user may query the system about data. Statistical computations for all the instances of one item, for example, would include maximum value, minimum value, mean value, median value, mode value, standard deviation, and total number of instances.

The QL thus should provide the capability for easily asking questions of the data base and asking questions that can be handled by the various application packages. By including optimization models in the application packages, the policymaker is able to move efficiently beyond the "What if" to the "What's best" question. Moreover, much additional research needs to be done on the query language (and its associated analyzer), since its enhancement adds much to GPLAN.

As an example of the power needed in the query language, consider the added components of the planning model from the Regional Water Pollution Control Planning System. It is able to handle two distinct types of planning problems. First, it is able to select a least-cost combination of treatment methods, given water quality goals and economical, political, and water quality information from the river basin. The second type of question which can be handled is directed toward individual projects. Types of individual planning problems that could be analyzed are:

a. What is the least cost solution for towns X and Y to handle their effluent? Should they combine to construct and operate a joint treatment plant?
b. What would be the least cost solution if consideration is given to the political constraints that may become operative?
c. What is the optimal plan for capacity expansion giving consideration to the growth and shift of population and industrial growth in the basin?
d. What are the least cost and optimal treatment plans that correspond to the task of providing water of high enough quality for certain recreation activities?
e. What is the sensitivity of the optimal pollution control plan to costs and constraints?
f. What is the optimal tradeoff between water quality, flow and alternative costs? For example, what would the difference in costs be if a plan were to permit the violation of a water quality standard once in 25 years as compared to once in 50 years?

GPLAN is a methodology for obtaining answers and responses to the above type of questions.

A query analyzer (QA)

A Query Language Analyzer must be able to analyze the BQL and as many application package query components as are available. A user enters his query in the QL, and the Query Analyzer analyzes the question and provides the user diagnostics to help him reformulate his question, if necessary. The query stored in machine readable form is processed by the QA which:

1. Checks for consistency in the Query and checks syntax in accordance with the Query Language; i.e., verifies the QL rules and is consistent, unambiguous, and complete.
2. Prepares error comments to aid the user in correcting, modifying and extending his Query.
3. Decides whether to pass the Query to the GDMS or one of the application packages.
4. May request additional information from the user if the action to be initiated requires it.

A collection of application packages or models

Application packages are simulation and optimization models, statistical packages, and other self-contained systems currently functioning under a specific computer and operating system.

We want to make it as easy as possible to add application packages, so we generalize the process by describing how we add a specific package.

We will make several assumptions about application packages: First, they are already running as batch jobs rather than interactive jobs. They may have quite long running times and making them interactive may simply mean waiting at the terminal; Second, they require user preparation to get the data ready; and third, they require other programs to run before the input data is complete.

We must know certain characteristics of an application package or model before we can consider tying it into GPLAN:

A. Input
   1. For each data input, we must know what kind it is (pure data or commands) and the associated types and formats.
   2. For each data input, we must know what kind of device it is assigned to (sequential or random-access).
   3. We must know the passes and correct logic steps and transformations to go from the data base to each input.
   4. For each data input, what must be included in system queries to the GDMS for 3 above?

B. Output
   1. Is the output self-explanatory or does it require minor explanation in the form of good documentation in the Query Language description?
   2. Does the output require much technical know-how to interpret the results? If yes, an output interpretation module is required (e.g., nonlinear river basin model solution).

C. What query components can it add to the Basic Query Language?

D. Minimal Documentation to be used by:
   1. Systems personnel
   2. Non-programmer researcher

Administrative report module

The Administrative Report Module will produce standard reports that must be completed and filed on a routine basis. These reports may be automatically triggered by queries or specifically requested from the console.

The standard reports will be supplied with data from the extraction file structure.

User's interface

Each interface component required for each part of an applications package must be defined:

A. Simple input linkage—direct to the Data Base.
B. Phased inputs—including self-started and analyzed Data Base retrievals.
C. Any combination of A. and B.
D. Simple output—direct from package.
E. Output interpretation module needed.

Extraction files

Between the data base and the set of application packages and the Administrative Report Module is a set of extraction files containing those items from the data base that are used for the packages and reports. Questions to be answered on extraction files are:

A. How many should there be?
B. What items should they contain?
C. What should their data structure and storage structure be?
D. How often should they be updated?
E. If a new application package or report is added, what changes should be made in the extraction file structure?
F. Should some application packages bypass extraction files entirely?

Users

There are two types of users connected with GPLAN: the technical systems personnel and the nontechnical administrator or manager.

A data administrator and his systems staff are responsible for: all original data input; updating of data; restructuring the Data Base and extraction files as necessary; changing machines and operating systems; adding new packages, standard reports (Administrative Report Module), and other additions or improvements. These systems users, taken as a group, must understand fairly well every component of GPLAN. They possibly could get by with not being familiar with some of the application packages, but then would have to get consultation to patch up or improve on these.

The major group of users are non-programming administrators. These are user's who don't know how to program, probably don't want to learn, and definitely shouldn't have to learn. They have a good understanding of the area for which the planning system was designed, or will have to have some training in this area before using GPLAN. Most of the details of the GPLAN implementation should be transparent to these users, and they should not notice any changes in the system, except the addition of new capabilities (possibly requested by them), new efficiency, or new packages. The success or failure of GPLAN depends on how well these users are able to carry out their querying of the data base and interaction with the application packages using only the query language and its documentation.

RFMS and RAMIS easily meet and/or exceed the minimal requirements for a GDMS as specified above. Thus all software being developed for GPLAN is being implemented on the CDC 6500 and the IBM 370/155.

Research is proceeding on the Query Language—Query Analyzer components in three areas. One area of investigation is the relationship between the QL and QA and the SODA Statement Language and SODA Statement Analyzer as used in the SODA project. Second, research on QL and QA is proceeding as a result of the development of the water pollution control models. Finally, the state of the art in artificial intelligence is being investigated for incorporation into the query language and analyzer.

STATUS OF GPLAN

There are two major efforts under way with respect to the development of GPLAN:

- Development and construction of the software for GPLAN.
- Work on a real world planning system (Water Pollution Control) and development of user training aids.

Development of software for GPLAN

Two different GDMS systems are being evaluated in parallel with the construction of GPLAN. Development is proceeding using RAMIS and RFMS (Remote File Management System). RFMS is a version of SYSTEM 2000 that was originally developed at the University of Texas at Austin. RFMS has been converted to run under the Purdue MACE Operating System, and substantial improvements have been added to the original version.

The two most difficult areas in the development of software for GPLAN are the User's Interface and the Extraction Files. Thus, a Data Description Language schema for data base description, and the Data Description Language subschema for the description of application package and administrative report data requirements, have been defined. Research is proceeding on the automatic mapping between the data base schema and an application package's subschema. Included in this mapping is a set of extraction files to be composed of subsets of items from the data base. An integer programming model has been defined which relates the data items of the data base to those on the extraction files as required by the application programs. Also, a cost function representing the cost of operating GPLAN has been defined. An important aspect of the problem is the optimization of the extraction files by solving for the extraction file arrangement which minimizes operating costs.

Work on a real application

Data is presently available to us from a previous study on the West Fork of the White River in Indiana for the development of a demonstration project concerning the Water Pollution Data Base Planning System. The insight achieved through the development of a specific planning system has already proved to be a tremendous aid in the accomplishment of the major goal of having a truly easy-to-use system.

The query system is being implemented in two modes:

1. Standard 80 column teletype
2. Graphics Terminal

The usefulness of GPLAN is enhanced considerably through the effective use of a graphical display. The graphics terminal offers the obvious advantage of being able to output designs, graphs, etc., in a more visible and appealing form. But the main advantage of interactive graphics is that it offers the user the capability of complete user interaction with the planning system.

Consider, for example, the river basin planning system. The optimization models output a diagram of the optimal solution to a specific water pollution control problem, showing the actual location of treatment plants and cooling towers on a computerized representation of the river basin, i.e., the actual solution is illustrated on a map of the river basin. The user may not have much confidence in the results, but he can at least relate to the output in this form. However, if he is given the opportunity to improve the solution by making adjustments to the design, or by changing the location or capacity of a treatment plant through the graphical query system, he finds that he is part of the decision making or planning process. Now, we can let the user input his own design and then compare the value of the objective function for his design with the optimal design. GPLAN can then indicate whether or not his design is even feasible. The user can also be given the opportunity to experiment with the values of the constraints and try different water quality goals. The result of this interaction is that the user has increased confidence in the planning system with a unique appreciation of the special talents and capabilities of man and machine.

The user can only be convinced that the mathematical solution is "good" if he can't improve on it himself. This experience was also supported by observations from a project concerned with the location of a major highway in southern California.[17]

Interactive graphics allows the user to utilize insight that often can't be built into models. This man-machine interaction enhances the planning system and brings the user into the decision making process.

REFERENCES

1. Pingry, D., Whinston, A., *A Survey of Planning Techniques for Pollution Control*, Krannert School of Industrial Administration, Purdue University, August 1972.
2. Pingry, D., Whinston, A., *A Multi-Goal Water Quality Planning Model*, Krannert School of Industrial Administration, Purdue University, August 1972.
3. Groner, Leo, Goel, Amrit, *Generalized Data Management Systems for Structured Information Retrieval*, Working Paper, Department of Systems and Information Science, Syracuse University, August 1972.
4. Minker, Jack, *General Data Management Systems—Some Perspectives*, Computer Science Center, University of Maryland, December 1969.
5. *Feature Analysis of Generalized Data Base Management Systems*, CODASYL Committee, Association for Computing Machinery, May 1971.
6. *Data Base Task Group Report*, CODASYL Committee, Association for Computing Machinery, April 1971.
7. Rice, J. R., Rosen, S., "NAPSS: A Numerical Analysis Problem Solving System," *Proceedings of the ACM National Conference*, 1966.
8. Nunamaker, J. F. Jr., "A Methodology for the Design and Optimization of Information Processing Systems," *AFIPS Proceedings*, Volume 38, AFIPS Press, May 1971.
9. *System 2000 General Information Manual*, MRI Systems, 1972.

10. *RAMIS Information Bulletin—Description and Specifications*, Mathematica, Inc., Princeton, New Jersey, 1972.
11. *Information Management System/360, Version 2 General Information Manual*, IBM, GH 20-0765-1, Second Edition, February 1971.
12. *DISK FORTE Manual*, Burroughs Corporation, 1969.
13. *OPTIMA Version 3.0 Reference Manual*, Control Data Corporation, Publication Number 60207000, Revision B, 1969.
14. Herman, D. J., Ihrer, F. C., "The Use of a Computer to Evaluate Computers," *AFIPS Proceedings*, Volume 25, May 1964.
15. Huesman, L. R., Goldberg, R. P., "Evaluating Computer Systems through Simulation," *Computer Journal*, August 1967.
16. *The Case for CASE: Computer Aided System Evaluation*, Computer Learning and Systems Corporation, 1971.
17. Liggett, Robin, Computer Graphics Staff, U.C.L.A. School of Architecture, Private Communication, July 1972.

Database sharing—An efficient mechanism for supporting concurrent processes

by PAUL F. KING and ARTHUR J. COLLMEYER

Xerox Corporation
El Segundo, California

INTRODUCTION

The advent of transaction-oriented data processing systems has offered a number of new challenges to designers of database management systems. Requisites for efficient transaction processing include (1) a multiprogramming system oriented toward maximizing throughput subject to the response-time requirement of the interactive environment, and (2) an integrated database with centralized access control. An integrated database implies the elimination of redundant data processing. Such is necessary (though not sufficient) to achieve acceptable performance in transaction processing. The necessity of an efficient, responsive multiprogramming system is, of course, obvious. But efficiency in the transaction environment necessitates certain system provisions peculiar to the environment. One of these is the provision for the shared use of data. Time-sharing systems, while they generally provide for shared procedures, do not generally provide elaborate facilities for data sharing, since users typically do not require access to files other than their own. In the transaction environment, typified by a number of users operating on a single integrated database, elaborate provisions for database sharing are required.

In the sense that the use of a database is a privilege frequently extended to different users, shared databases are quite common. This kind of sharing is done primarily to avoid the proliferation of redundant data. But for the purposes of this paper, database sharing refers to the granting of *simultaneous* access to a database. This kind of sharing is motivated by the necessity for efficient, responsive multiprogramming in the transaction environment. By making a database available to several programs simultaneously, the mean number of programs eligible for execution increases, as does the mean occupancy of the disk queue, thereby increasing the potential for effective utilization of resources. The potential for improved performance depends, of course, on the manner in which the database is organized on the storage medium. Random (physical) organizations are more appropriate in transaction processing than the sequential organizations of batch processing systems. The potential for improved performance likewise depends on the extent to which the data base can be shared.

The most sophisticated approach to database sharing admits concurrent WRITERS as well as READERS. This approach necessitates the most elaborate mechanism for regulating access to data within the database. Such mechanisms have been described for numerous applications.[1–9] Indeed the CODASYL DBTG Report,[10] in its proposed specification for a database management systems, details a KEEP-FREE mechanism to support concurrent update operations. However, the KEEP-FREE mechanism is not without shortcomings. The programmer using KEEP-FREE is, in essence, availing himself of a facility designed to inform him of any untoward interactions with other update programs. KEEP-FREE is not a mechanism for "locking" and "unlocking" data. While it avoids the "deadlock" problem, it does place a significant (perhaps excessive) burden on the application programmer insofar as the integrity of the database is concerned.

A LOCK-UNLOCK mechanism enabling the locking and unlocking of data for update purposes is a preferable solution. In general, however, a LOCK-UNLOCK mechanism is substantially more complex, as it introduces the potential for deadlock. Where databases are constructed solely on hierarchical structures it may be appropriate to insist that consecutive LOCK requests be separated by an UNLOCK request, so as to eliminate the possibility of deadlock. However, where network structures are exploited, such a limitation may be unreasonable.

In this paper, a LOCK-UNLOCK Mechanism enabling the incremental allocation of data elements to processes is described, along with an efficient method for detecting potential deadlocks. The mechanism prevents inter-process interference and permits simple automatic recovery from deadlocked processes as well.

LOCK-UNLOCK

An environment not unlike that of transaction processing is assumed. A number of programs, WRITERS as well as READERS, operate concurrently on a single database. In this mode of operation, READERS are presumed impervious to the effects of concurrent WRITERS. In other words, READERS are fully aware of the possibility that the database may be altered during the course of their task and that information extracted from the database may be out-of-date by the time the task is completed. If such is unacceptable, an inquiry program may use the LOCK-UN-

LOCK facility, in which case its behavior is likened to that of a WRITER who locks records one at a time until he has acquired exclusive control over all records affected by the intended update. When the update has been accomplished, the records are unlocked via the UNLOCK Function. For the sake of simplicity, we will classify a user program as either (1) a READER, impervious to changes in data, or (2) a WRITER, who allocates records for exclusive use via the LOCK Function. It is users in the second category who make database sharing a difficult proposition.

WRITERS, using the LOCK-UNLOCK Mechanism, lock records one at a time until all the records involved in the intended operation have been acquired. Hence, an operational characteristic peculiar to a WRITER is that set of records allocated for exclusive use via the LOCK Function. For the purposes of this paper this set is known as the *lock list*. Taken as a set, the lock lists of all the WRITERS currently active in the system constitutes a listing of all the records which cannot be allocated for exclusive use. If a WRITER happens to request one of these records for his exclusive use, he must be placed in a queue for said record. Under these circumstances, the WRITER is said to be *blocked*. The blocked WRITER cannot proceed until the record in question has been unlocked.

Two (or more) WRITERS incrementally allocating records to themselves pose the problem of deadlock. A blocked WRITER is *deadlocked* when there is no way through normal operations for him to become not blocked. To illustrate, suppose WRITER 1 has locked record j and wants record k, and WRITER 2 has locked record k, and wants record j. Neither can proceed; the update of WRITER 1, as well as that of WRITER 2 is deadlocked. It is necessary to "backtrack" one of the stymied programs under system control and then restart it.

The detection of incipient deadlocks can be a more or less complex operation, depending on the detection algorithm. The computational overhead of a mechanism to solve for necessary and sufficient conditions has to date been considered intolerable. Simpler mechanisms based on necessary (but not sufficient) conditions have been proposed, at the cost of increased detection frequency. In this paper, a simple algorithm is proposed for deadlock detection based on necessary and sufficient conditions.

THE MODEL

The algorithm that has been developed for deadlock detection is best presented by means of a graphical model of data element access under the LOCK-UNLOCK Mechanism. To introduce the model, some graph-theoretic definitions are first required.

A *directed graph* is a pair $<N, A>$, where N is an abstract set and $A \subseteq N \times N$. Each element in N is called a *node* and each pair (a, b) in A is termed an *arc*. The arc (a, b) is said to be *directed from* node b. A *path* is a sequence of two or more nodes $(n_1, n_2, \cdots, n_m)$ with each connected to the next by an arc. That is, for each n_i, $1 \leq i \leq m-1$, $(n_i, n_{i+1}) \in A$. A path is a *loop* if its first and last nodes are the same.

From the discussion in the previous section, it should be clear that the state of all accesses with respect to a given database can be defined by describing:

(1) the allocatable data elements (e.g., records) in the database,
(2) the active processes (WRITERS), and
(3) the lock list associated with each process.

Therefore, the state of all accesses of a database can be defined by a *database access state graph*, $<P \cup E, L>$. The set of nodes within each of these graphs consists of the union of the set of active processes, P, and the set of allocatable data elements, E. Each lock list is represented by a path beginning at an active process node and connecting it with each data element allocated to that process. Thus, the set of arcs within the lock lists comprises the set L.

More formally, a database access state graph is a directed graph $<P \cup E, L>$ where

$P = \{p_i \mid p_i$ is the i-th oldest process$\}$,
$E = \{e \mid e$ is an allocatable data element$\}$, and
$L = A \cup B$.

The set of lock lists, L, is composed of the set of allocated elements,

$A = \{(a, b) \mid a = p_i$ and b is the oldest data element allocated to p_i, or
$A = e_{ij}$, the j-th oldest data element allocated to p_i and $(e_{ij-1}, e_{ij}) \in A\}$,

and the set of blocked allocation requests,

$B = \{(a, b) \mid a = p_i$ or $a = e_{ij-1}$ and $(e_{ij-2}, e_{ij-1}) \in A$ with process p_i being blocked when requesting allocation of data element $b = e_{ij}$. That is, $b = e_{kl}$ for some $k = i$ and I such that either $(p_k, b) \in A$ or $(e_{kl-1}, b) \in A\}$.

Since each access state of a database is represented by an access state graph, operation of the LOCK-UNLOCK Mechanism can be modeled by state transition functions that map access state graphs to access state graphs. The four required functions are:

(1) The LOCK Function. If database access state $s = <P \cup E, L>$ then LOCK(s) $= s' = <P' \cup E, L>$. If $P = \{p_i \mid p_i$ is a process, $1 \leq i \leq n\}$ then $p' = p \cup \{p_{n+1}\}$. That is, the LOCK Function adds a process node to the graph.
(2) The UNLOCK Function. This function is the inverse of the LOCK Function, and its application deletes an isolated process node from a graph.
(3) The ALLOCATE Function. If database access state $s = <P \cup E, L>$, then ALLOCATE $(s, p_i, e_{ij}) = s'$. If $L = A \cup B$ then $s' = <P \cup E, L'>$ and $L' = A \cup B \cup \{(p_i, e_{ij})$ or $(e_{ij-1}, e_{ij})\}$. This function adds an arc to

the graph and thereby models the allocation of a data element to a process.

(4) The DEALLOCATE Function. This function is the inverse of the ALLOCATE Function. Its application deletes an arc from the graph and thus represents the release of a data element from a lock list.

Figure 1 illustrates the application of the LOCK and UNLOCK Functions to a simple database access state. In the first graph shown, $P=\{p_1, p_2, p_3\}$, $E=\{d_1, d_2, \cdots, d_9\}$, and L = the set of paths, $\{(p_1, d_7, d_8, d_9), (p_2, d_4, d_2), (p_3, d_4)\}$. The arc $(p_3, d_4) \in B$ indicates p_3 is blocked, while all other arcs are elements of A. The figure shows the LOCK Function adding process node p_4, and the UNLOCK Function is shown deleting it. Figure 2 gives an example of the application of the ALLOCATE and DEALLOCATE Functions.

In terms of the model, the normal access sequence for a given process consists first of an application of the LOCK

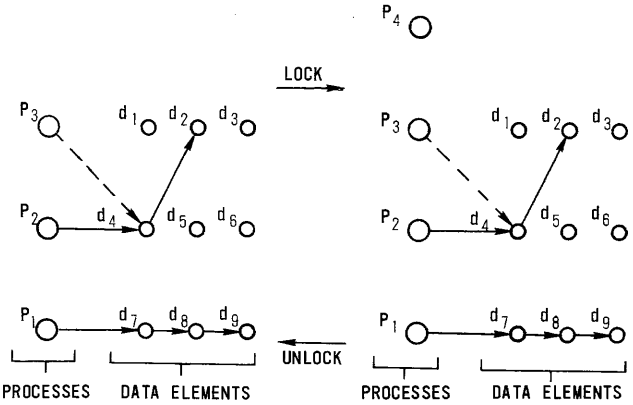


Figure 1—The LOCK and LOCK functions

Function. This is followed by some number of applications of the ALLOCATE and DEALLOCATE Functions. At some point, the number of applications of ALLOCATE is equaled by applications of DEALLOCATE, and the access sequence ends with an UNLOCK.

DEADLOCK DETECTION

From the discussion above, it should be obvious that the ALLOCATE Function is the only function defined that can precipitate a deadlock. This is clearly the case, for ALLOCATE is the only function capable of blocking a process.

It is now possible to describe a simple and efficient deadlock detection algorithm in terms of the model just presented. The following theorem provides the theoretical basis for the detection procedure.

THEOREM: If a valid database access state s is not a deadlocked state, then

ALLOCATE $(s, p_i, e)=s'$ is a deadlocked state if and only if

(1) process p_i is blocked on attempting to allocate data element e, and

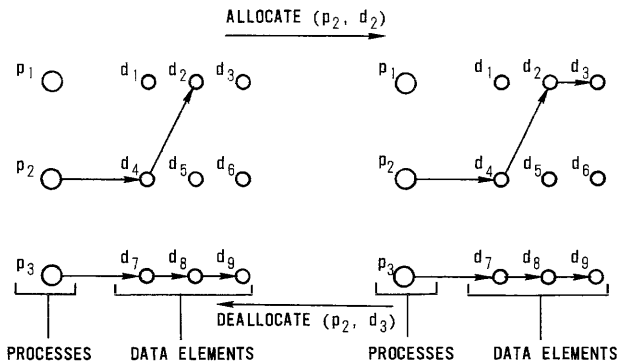


Figure 2—The ALLOCATE and DEALLOCATE functions

(2) the database access state graph representing s' contains a loop.

PROOF: To establish these conditions as necessary for s' to be a deadlocked state, notice first that if p_i is not blocked then, by definition, s' is not a deadlocked state. Now, let $<P \cup E, L>$ be the database access state graph representing s with $P=\{p_i \mid 1\leq i\leq n, \text{ and } n\geq 2\}$. Assume ALLOCATE $(s, p_i, e)=s'$ is a deadlocked state and that $s'=<P \cup E, L'>$ does not contain a loop.

Since s' is a deadlocked state, in s' there is a set of P_d of m deadlocked processes, where $m\leq 2$, and $P_d=\{p_{d1}, p_{d2}\cdots, p_{dm}\} \subseteq P$. By definition, each $p_{di} \in P_d$ is blocked. Furthermore, each p_{di} is blocked by a $p_{dj} \in P_d$, with $i\neq j$. If p_{di} were not blocked by some $p\in P_d$, then p_{di} would have to be blocked by a nondeadlocked process; therefore, p_{di} would not be deadlocked. Thus, if there are m processes in P_d, then p_{di} is blocked by one of the $(m-1)$ processes $\{p_{d2}, p_{d3}, \cdots, p_{dm}\}$.

Assume for convenience that p_{d2} blocks p_{d1}. Now, p_{d2} must in turn be blocked by one of the $(m-2)$ processes $\{p_{d3}, p_{d4}, \cdots, p_{dm}\}$. If not, then p_{d2} would have to be blocked by p_{d1}. If p_{d2} were blocked by p_{d1}, then for some data element c' the path $(p_{d2}, \cdots, c) \in A$ and the arc $(c, c') \in B$ while the path $(p_{d1}, \cdots, c') \in A$. But since p_{d1} is blocked by p_{d2}, this implies that for some data element b' the path $(p_{d1}, \cdots c', \cdots, b) \in A$ and the arc $(b, b') \in B$ while the path $(p_{d2}, \cdots, b', \cdots, c) \in A$. This, however, violates the assumption that no loop is contained in $<P \cup E, L'>$, since the path $(b', \cdots, c, c', \cdots, b, b')$ is a loop (see Figure 3). Hence, p_{d2} must be blocked by one of the processes $\{p_{d3}, p_{d4}, \cdots, p_{dm}\}$.

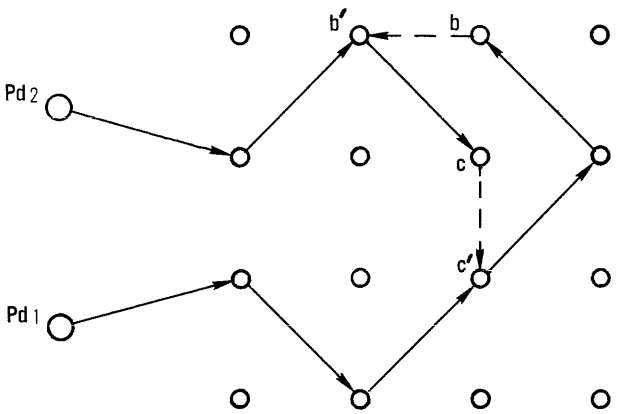


Figure 3—A deadlocked state involving processed p_{d1} and p_{d2}

More generally, note that if a sequence of k processes $p_1, p_2, \cdots, p_k$ exists such that each p_{i-1} is blocked by p_i for all $2 \leq i \leq k$, then a path exists from p_1 through elements of each p_i's lock list to the last element in the lock list of p_k. The lock list of each p_{i-1} is connected to the lock list of p_i by the arc representing a blocked request, which is always the last arc in a lock list.

Now consider the j-th process of the m deadlocked processes of state s'. Assume for convenience that for $2 \leq i \leq j$, p_{di-1} is blocked by p_{di}. Then, p_{dj} must be blocked by one of the $(m-j)$ processes $\{p_{dj}, p_{dj+1}, \cdots, p_{dm}\}$. For if p_{dj} were blocked on allocating b' by some p_{di} with $1 \leq i < j$, then the path $(p_{di}, \cdots, b', \cdots, b)$ exists and the arc (b, b') from the lock list of p_{dj} to that of p_{di} creates the loop $(b', \cdots, b, b')$—contrary to the assumption that no loop exists. Hence, p_{di} must be blocked by one of the processes $\{p_{di}, p_{di+1}, \cdots, P_{dm}\}$.

However, for $j = m$ the above implies that $(m-m)$ or zero choices remain for the process that blocks p_{dm}. Therefore, it was false to assume that if s' is a deadlocked state, then a loop did not exist in $<P \mathbf{U} E, L'>$.

To establish the sufficiency of the conditions in the theorem, suppose that the blocking of p_i creates a loop in the access state graph. Since a single lock list cannot contain a loop, elements of j lock lists, where $j \leq 2$, must participate in the loop. Since the elements of one lock list can be connected to those of another lock list only by an arc representing a blocked allocation request, the existence of a loop implies that a sequence of processes $p_1, p_2, \cdots, p_j$ exists with each p_{i-1}, $? \leq i \leq j$, being blocked by p_i and p_j being blocked by p_1 In this case a set of processes exist such that each process is blocked by a member of that same set; thus, no chance remains for them to become not blocked. Therefore, the state s' is a deadlocked state, and the theorem is established.

Straightforward application of this theorem results in a deadlock detection procedure that is both simple and efficient. Since a deadlock can occur only when an allocation request results in a process being blocked, which is assumed to be an infrequent event in a transaction processing environment,[11] only infrequently will an examination of the database access state be necessary. In those instances when a process is blocked and it becomes necessary to test for a loop in the access state graph, the computation required for the test is nearly trivial since (1) the data element requested by the blocked process must be in the loop, and (2) the out-degree of every node in a database access state graph is one. Thus, deadlocked states of arbitrary complexity are easily and efficiently detected.

This detection method has yet another useful characteristic —it directly identifies those processes that are responsible for the deadlock. The processes that are responsible are, of course, those which are blocking each other. In general, however, it is possible to encounter a deadlocked access state in which an arbitrary number of processes participate, but only a small fraction of these are responsible for that state. This condition can exist since any number of processes can themselves be blocked while either not blocking other processes or not blocking others in a deadlocked manner (i.e. processes participating in the deadlock whose lock lists can be removed from the database access state graph without removing the deadlock condition). However, it is obviously those processes whose lock lists participate in the loop that cause a deadlock condition to exist. By detecting the existence of a loop, the algorithm has also isolated the processes responsible for the deadlock; and, thus, the method has also accomplished the first essential step in the recovery process.

RECOVERY

In the context of shared databases, Recovery is the procedure by which the effects of a (necessarily) aborted process on the object database* are reversed so that the process can be restarted. On the detection of a deadlock a Recovery Procedure must be invoked. The first element of the Recovery Procedure is the determination of which process to abort. This decision might be based on any of several criteria. For example, it might be advisible to abort the most recent of the offending processes, or the process with the fewest allocated data elements. In any event, the information required for this decision is readily available (see above). This decision is, however, a small part of the recovery problem. To recover efficiently, the LOCK-UNLOCK Mechanism requires features beyond an efficient detection algorithm. One such feature is a checkpoint facility—a facility that records the state of a process and thus enables it to be restarted. Clearly, a *checkpoint* must be performed at each LOCK. Furthermore, to enable efficient restoration of the database, utilization of a *process map* is appropriate.

A process map is basically a map from a virtual addressing space to a real addressing space, maintained for each process in the database access state graph. Database management systems are typically characterized by three levels of addressing: content addressing, logical addressing, and physical addressing. Associative references are processed in two steps: a content-to-logical address transformation, followed by a logical-to-physical address transformation. This "virtual" secondary storage characterized by a logical-to-physical storage map provides device independence and facilitates the efficient use of storage hierarchies. The process map is similarly a logical-to-physical storage map. A process map is created and associated with process at the execution of a LOCK Function. With each execution of an ALLOCATE Function, a (physical) copy of the allocated data element is created and an entry is made in the associated process map. Subsequent references by the process to the database are routed through its process map; hence, incremental updates are performed on the copies of the data elements. The DEALLOCATE Function effects a modification** of the database storage map and the deletion of the associated entry in the process map. ALLOCATE therefore has the effect of creating a physical copy of the object data element accessible only to the allocator, and DEALLOCATE has the effect of making the physical copy available to all processes and the

* The term database as used here includes all data files affected by the process, auxiliary files as well as the principal data files.

** The database storage map is modified so that references to the object data element are mapped to the physical copy created at the execution of ALLOCATE and subsequently modified by the allocator.

original available to the database manager as allocatable space.

A process map makes the recovery from a deadlocked access state a relatively simple matter. Once a decision is reached as to which process to abort, that process is merely restarted at the checkpoint performed by the LOCK Function. Implicit in this action is, of course, the restoration of the lock list of that process to an empty state. That is, each data element that was allocated to the process is released, and the copies of these elements are discarded. Clearly, priorities must be appropriately arranged to insure that a process blocked by the aborted process is allocated the released data element for which it was previously blocked.

No further action by the Recovery Procedure is required, for due to the process maps, the actual database was unaltered by the aborted process. Note further that the utilization of process maps significantly reduces the probability of WRITERS interfering with READERS, since references to data elements by READERS are always directed to the actual database.

SUMMARY

The above discussion of the LOCK-UNLOCK Mechanism is intended to serve as a functional description of the elements of a database management system that are essential in order to provide an efficient facility for database sharing. In an actual database management system, the LOCK-UNLOCK Mechanism could be manifested in the form of LOCK and UNLOCK commands used by the programmer. Alternatively, the LOCK Function could be assumed implicit in the commonly used OPEN command. Under these schemes, ALLOCATE could be accomplished via a FIND command, and DEALLOCATE could be implicitly invoked by an UNLOCK or CLOSE.

The occurrence of a deadlock can be translated directly into a degradation in system throughput. The work done by a process to the point where it is aborted *plus* the overhead required for recovery represent the computational cost of a deadlock. Thus the justification of a LOCK-UNLOCK mechanism of the type described here is predicated on an acceptably low frequency of occurrence of deadlocked access states. Of course, as tasks become small in terms of computational and other resource requirements, the throughput cost of deadlocks as well as the probability of their occurrences diminishes.

Any database sharing mechanism can significantly contribute to the satisfaction of the requirement for efficient, responsive multiprogramming in the transaction environment. The LOCK-UNLOCK Mechanism not only provides the potential for efficient database sharing, but it also eliminates the requirement for special consideration for sharing from the application program. Moreover, this is accomplished while the integrity of the database is guaranteed.

REFERENCES

1. Bernstein, A. J., Shoshani, A., "Synchronization in a Parallel Accessed Data Base," *CACM*, Vol. 12, No. 11, pp. 604-607, November 1969,
2. Coffman, E. G., Elphick, M. J., Shoshani, A., "System Deadlocks," *Computing Surveys*, Vol. 3, No. 2, pp. 67-77, June 1971.
3. Collmeyer, A. J., "Database Management in a Multi-Access Environment" *Computer*, Vol. 4, No. 6, pp. 36-46, November/December 1971.
4. Dennis, J. B., Van Horn, E. C., "Programming Semantics for Multiprogrammed Computations," *CACM*, Vol. 9, No. 3, pp. 143-155, March 1966.
5. Dijkstra, E. W., "The Structure of THE Multiprogramming System," *CACM*, Vol. 11, No. 5, pp. 341-346, March 1968.
6. Habermann, A. N., "Prevention of System Deadlocks," *CACM*, Vol. 12, No. 7, pp. 373-385, July 1969.
7. Havender, J. W., "Avoiding Deadlock in Multitasking Systems," *IBM Systems Journal*, No. 2, 1968.
8. Murphy, J. E., "Resource Allocation with Interlock Detection in a Multi-Task System," *Proceedings AFIPS Fall Joint Computer Conference*, pp. 1169-1176, 1968.
9. Holt, R. C., "Some Deadlock Properties of Computer Systems," *Computing Surveys*, Vol. 4, No. 3, pp. 179-196, September 1972.
10. CODASYL Data Base Task Group Report, April 1971.
11. Shemer, J. E., Collmeyer, A. J., "Database Sharing—A Study of Interference, Roadblock, and Deadlock, *Proceedings of 1972 ACM-SIGFIDET Workshop.*

Optimal file allocation in multi-level storage systems*

by PETER P. S. CHEN**

Harvard University
Cambridge, Massachusetts

INTRODUCTION

Storage is an important and expensive component of a computer system. Many types of storage such as semiconductor, magnetic core, bulk core, disk, drum, tape, etc. are available today, each having different cost and physical attributes (e.g., access time). To be economical, the storage system of a modern computer generally consists of several different types of storage devices. Such an assemblage is called a *multi-level storage system* (or a *storage hierarchy system*).

Since each type of storage has a different cost/performance ratio, a series of important problems arise in the design and use of multi-level storage systems—for example, how to allocate files within a multi-level storage system in order to achieve the best performance without considering cost, and also when the cost is considered. The purpose of this paper is to study these problems.

For simplicity in designing models, the following assumptions are made:

(a) Statistics of file usage are assumed to be known either by hardware/software measurements in previous runs or by analysis of frequency and type of access to the information structure.
(b) Allocation is done statically (before execution) and not dynamically (during execution).

Although these assumptions are introduced primarily to make analysis more tractable, many practical situations fit into this restricted case. One example is periodical reorganization of the data base for airline reservation systems. Another is the allocation of user files and non-resident system programs to auxilary devices.

These file allocation problems have usually been treated intuitively or by trial and error. Only recently have some analyses been performed in this area.[1,2,3] The work done by Ramamoorthy and Chandy[4] and by Arora and Gallo[5] is particularly interesting; it concludes that the optimal file allocation strategy is to allocate the more frequently used files to faster devices to the extent possible. However, waiting time in the request queues is ignored. Thus, this file allocation strategy may induce undesirably long request queues before some devices.

By considering queueing delay, a more realistic analysis may be performed. We analyze three types of file allocation problem. The first one is to allocate files minimizing the mean overall system response time without considering the storage cost. The second one is to allocate files minimizing the total storage cost and satisfying one mean overall system response time requirement. The last one is to allocate files minimizing the total storage cost and satisfying an individual response time requirement for each file. We propose algorithms for the solutions of the first two problems; the third problem is considered elsewhere.[6]

ANALYSIS

To design models, it is important to identify the significant parameters of the physical systems and to describe the interrelationships among these parameters. In the following, we shall describe the essential characteristics of file allocation problems.

The storage device types concerned in this paper are auxilary devices. It is assumed that the block size is fixed for each device type, exactly one block of information is transfered per input-output request, and a storage cost per block is associated with each device type. The service time for a request generally consists of two components. One is the data transfer time, roughly a constant for each device, and the other is the electromechanical delay time, which may vary from request to request. Thus, the device service time is considered to be a random variable.

Let M denote the total number of devices in the storage hierarchy. For device j ($j=1, 2, \ldots, M$), the cost per block is c_j, and request service time is assumed to be exponentially distributed with mean $1/u_j$ $(u_1 \geq u_2 \geq \cdots \geq u_M > 0)$:

$$\text{Prob [service time} \leq t] = 1 - \exp(-u_j t), \quad t \geq 0$$

A *file* is a set of information to be allocated in the storage hierarchy. The length of a file may vary from a few words to some bound set by the system designer. The file reference frequency is assumed to be known by some means. For

* This work was sponsored in part by the Electronic Systems Division, U.S. Air Force, Hanscom Field, Bedford, Massachusetts under Contract No. F-19628-70-C-0217.

** The work reported here will be included in the author's Ph.D. Thesis, "Optimal File Allocation," to be presented to Harvard University.

simplicity, it is assumed that each block of a file has the same request frequency (for otherwise, we may redefine the files to satisfy this condition).

Let L denote the total number of files. The length of file i ($i=1, 2, \ldots, L$) is denoted by N_i and the per block request frequency by f_i ($f_1 \geq f_2 \geq \cdots \geq f_L$).

We assume that each block of a file can be assigned to any storage device, and there is sufficient storage on each device to allocate files in any manner desired. L files (with N_i blocks each) must be allocated among M storage devices. We are interested in the effect of the allocation pattern on the total storage cost and response time.

Let n_{ij} denote the number of blocks of file i allocated to storage device j. The n_{ij} are integers; however, for simplicity we shall assume the n_{ij} are continuous variables. (A near-optimal integer solution can be obtained by rounding the optimal solution to the nearest integer.) Note that the n_{ij} are nonnegative:

$$n_{ij} \geq 0, \qquad i=1, \ldots, L, \quad j=1, \ldots, M \tag{1}$$

Note also that

$$\sum_{j=1}^{M} n_{ij} = N_i, \qquad i=1, \ldots, L \tag{2}$$

Since c_j is the cost per block of storage using device j, and $\sum_{i=1}^{L} n_{ij}$ is the total number of blocks of storage using device j, the total storage cost is:

$$C = \sum_{j=1}^{M} c_j \sum_{i=1}^{L} n_{ij} \tag{3}$$

Since the reference frequency for blocks in file i ($i=1, 2, \ldots, L$) is f_i, and there are n_{ij} blocks of file i allocated to storage device j, the total input request rate for device j is:

$$\lambda_j = \sum_{i=1}^{L} n_{ij} f_i \tag{4}$$

To prevent the queue lengths from growing without bound, it is required that

$$\lambda_j = \sum_{i=1}^{L} n_{ij} f_i < u_j \tag{5}$$

That is, the overall arrival rate to a device must be less than the device service rate.

The input requests for each device are assumed to be independent and to arrive at random. Thus, the input to each device is generated by a Poisson process with mean rate λ_j. The total mean input rate for the system is the sum of the mean input rate for all devices. (It is also the total request rate for all files.) That is,

$$\lambda = \sum_{j=1}^{M} \lambda_j = \sum_{j=1}^{M} \sum_{i=1}^{L} n_{ij} f_i \tag{6}$$

Since the input is generated by a Poisson process, and the service time is exponentially distributed, the mean response time for a request forwarded to device j is given by[7]

$$R_j = 1/(u_j - \lambda_j) = 1 \Big/ \left(u_j - \sum_{i=1}^{L} n_{ij} f_i\right) \tag{7}$$

The mean system response time is the weighted sum of the mean response time for requests forwarded to each device:

$$R(\lambda_1, \ldots, \lambda_M) = \sum_{j=1}^{M} (\lambda_j/\lambda) R_j = \sum_{j=1}^{M} [\lambda_j/\lambda(u_j - \lambda_j)]$$

or,

$$R(\{n_{ij}\}) = \sum_{j=1}^{M} \left[\left(\sum_{i=1}^{L} n_{ij} f_i\right) \Big/ \left(\sum_{j=1}^{M} \sum_{i=1}^{L} n_{ij} f_i\right) \times \left(u_j - \sum_{i=1}^{L} n_{ij} f_i\right)\right] \tag{8}$$

The probability that the response time for a request forwarded to device j exceeds T can be expressed by[7]

$$P_j(t>T) = \exp[-(1-\lambda_j/u_j)u_j T] = \exp\left[\left(\sum_{i=1}^{L} n_{ij} f_i - u_j\right) T\right] \tag{9}$$

Since n_{ij}/N_i is the proportion of requests for file i forwarded to device j, the probability that the response time for a request for file i exceeds T can be expressed as a weighted sum of the individual probabilities for each device:

$$P^{(i)}(t>T) = \sum_{j=1}^{M} (n_{ij}/N_i) P_j(t>T) = \sum_{j=1}^{M} (n_{ij}/N_i) \exp\left[\left(\sum_{i=1}^{L} n_{ij} f_i - u_j\right) T\right] \tag{10}$$

MINIMIZING THE MEAN SYSTEM RESPONSE TIME

Problem statement

Consider the simple case in which the mean system response time is the only measure of performance, storage cost being ignored. The file allocation problem can be stated: allocate files such that the mean system response time is minimized. This problem is formulated as a nonlinear programming problem as follows:

Minimize

$$R(\{n_{ij}\}) \tag{11a}$$

subject to

$$\sum_{i=1}^{L} n_{ij} f_i < u_j, \qquad j=1, \ldots, M \tag{11b}$$

$$n_{ij} \geq 0, \qquad i=1, \ldots, L, \quad j=1, \ldots, M \tag{11c}$$

$$\sum_{j=1}^{M} n_{ij} = N_i, \qquad i=1, \ldots, L \tag{11d}$$

The interpretations of (11) can be found in the derivations of (8), (5), (1), and (2). The above will be referred to as the *type 1 problem.*

Substituting (4) and (6) into (11) in the type 1 problem and reorganizing, we get the following system:

Minimize

$$R(\lambda_1, \ldots, \lambda_M) \tag{12a}$$

Subject to

$$\lambda_j < u_j, \quad j=1, \ldots, M \tag{12b}$$

$$\lambda_j \geqq 0, \quad j=1, \ldots, M \tag{12c}$$

$$\sum_{j=1}^{M} \lambda_j = \lambda \tag{12d}$$

The above is called the *load partition problem*: partition a given total input load λ into $\lambda_1, \lambda_2, \ldots, \lambda_M$ for each device so that the mean system response time is minimized.

Optimal solution

The optimal solution of the load partition problem is obtained as follows:[8]

For $\Lambda(k+1) > \lambda \geqq \Lambda(k)$ $(k=1, \ldots, M)$, set,

$$\lambda_j^* = u_j - u_j^{1/2}\left(\sum_{s=1}^{k} u_s - \lambda\right) \Big/ \sum_{s=1}^{k} u_s^{1/2}, \quad j=1, \ldots, k$$

$$\lambda_j^* = 0, \quad \text{otherwise} \tag{13}$$

where

$$\Lambda(k) = \sum_{j=1}^{k} u_j - u_k^{1/2} \sum_{j=1}^{k} u_j^{1/2}, \quad k=1, \ldots, M$$

$$\Lambda(M+1) = \sum_{j=1}^{M} u_j$$

Theorem 1. The set of input rates obtained by (13) holds for every optimal solution to the type 1 problem.

Proof of this theorem follows immediately from the manner in which the load partition problem was obtained from the statement of the type 1 problem.

Utilizing (13), we propose the following algorithm for solution of the type 1 problem.

Algorithm 1.

Step 1. Calculate the total mean input rate λ by (6).

Step 2. Use (13) to obtain the optimal solution $\lambda_1^*, \ldots, \lambda_M^*$ for the corresponding load partition problem.

Step 3. Allocate file blocks in any manner desired, but ensure the resulting mean input rate to device j is equal to λ_j^*.

The optimality of Algorithm 1 follows directly from Theorem 1.

In the following, we illustrate the use of Algorithm 1 by a simple example.

Example 1. Suppose that there are three files $(L=3)$ to be allocated to two devices $(M=2)$ in order to minimize the mean system response time.

Given:

$$f_1=3 \quad f_2=2 \quad f_3=1$$
$$N_1=50 \quad N_2=80 \quad N_3=100$$
$$u_1=400 \quad u_2=100$$

Using Algorithm 1, the total input rate is

$$\lambda = 3\times 50 + 2\times 80 + 1\times 100 = 410$$

The optimal solution for the corresponding load partition problem is

$$\lambda_1^* = 400 - 20(400+100-410)/(20+10) = 340$$
$$\lambda_2^* = 70$$

Any file allocation pattern with $\lambda_1^*=340$ and $\lambda_2^*=70$ will be optimal. For instance:

$$n_{11}=50 \quad n_{21}=80 \quad n_{31}=30 \tag{1}$$
$$n_{12}=0 \quad n_{22}=0 \quad n_{32}=70$$

$$n_{11}=40 \quad n_{21}=60 \quad n_{31}=100 \tag{2}$$
$$n_{12}=10 \quad n_{22}=20 \quad n_{32}=0$$

A commonly used file allocation strategy is:

Allocation strategy A. Order files according to their relative frequency of access, and allocate files to devices in order, starting by allocating the file with the highest reference frequency to the faster device, etc. (for example, the first set of solution in Example 1).

The second solution stated in the above example provides a counterexample to the conjecture that allocation strategy A is a necessary condition for the optimal solution of the type 1 problem.

MINIMIZING THE STORAGE COST—ONE MEAN SYSTEM RESPONSE TIME CONSTRAINT

Problem statement

When storage cost is also a factor, the following problem arises: allocate the files such that the storage cost is minimized and the mean system response time is bounded. That is,

Minimize

$$\sum_{j=1}^{M} c_j \sum_{i=1}^{L} n_{ij} \tag{14a}$$

subject to

$$R(\{n_{ij}\}) \leqq V \tag{14b}$$

$$\sum_{i=1}^{L} n_{ij} f_i < u_j, \quad j=1, \ldots, M \tag{14c}$$

$$n_{ij} \geqq 0, \quad i=1, \ldots, L, \quad j=1, \ldots, M \tag{14d}$$

$$\sum_{j=1}^{M} n_{ij} = N_i, \quad i=1, \ldots, \mathrm{L} \tag{14e}$$

(14a) denotes the total storage cost. (14b) is the mean system response time constraint where V is the given bound. The above will be referred to as the *type 2 problem.*

Optimal solution

The following theorems state the properties of the (unique) optimal solution for the type 2 problem. Proofs are supplied in the Appendix.

Theorem 2.1. If $c_j > c_{j+1} (j=1, \ldots, M-1)$, then the optimal solution for the type 2 problem follows allocation strategy A.

In the following theorem, we consider the case in which there are only two devices ($M=2$). Let λ_1', λ_2' denote the mean input rates at the optimum for the type 2 problem with $M=2$, and λ_1^*, λ_2^* denote the mean input rates at the optimum for a corresponding load partition problem. Let

$$S = \{\lambda_1 \mid u_1 > \lambda_1 > \lambda - u_2, \lambda_1 \geqq 0, R(\lambda_1, \lambda - \lambda_1) \leqq V\}$$

where

$$R(\lambda_1, \lambda - \lambda_1) = \lambda_1/\lambda(u_1 - \lambda_1) + (\lambda - \lambda_1)/\lambda(u_2 - \lambda + \lambda_1)$$

It is easy to see that a solution, $\{n_{ij}\}$, satisfies (14b)-(14e) is equivalent to that corresponding input rates (λ_1, λ_2) satisfy $\lambda_1 \in S$.

Theorem 2.2. For a type 2 problem with $M=2$, the mean input rates (λ_1', λ_2') at the optimum have the following property:

$$\lambda_1' = \text{Min}\{\lambda_1 \mid \lambda_1 \in S\}$$

It is easy to see that $R(\lambda_1, \lambda - \lambda_1)$ is convex on λ_1 in $0 \leqq \lambda_1$ and $\lambda - u_2 < \lambda_1 < u_1$. The minimum of $R(\lambda_1, \lambda - \lambda_1)$ occurs at $\lambda_1 = \lambda_1^*$. Thus, λ_1' is easy to obtain. (In fact, in most cases we only need to solve a second order equation.)

Conventional algorithms usually involve matrix manipulations which are time-consuming. In the following, we propose an efficient algorithm utilizing Theorems 2.1 and 2.2 for $M=2$. For $M \geqq 3$, we choose a conventional algorithm, and propose an easy method for obtaining the initial feasible solution.

Aglorithm 2.1. (for $M=2$)

Step 1. Calculate the total input rate λ using (6).

Step 2. Find λ_1' by the method stated in Theorem 2.2.

Step 3. Allocate files according to allocation strategy *A*, ensuring that the mean input rate to the faster device equals to λ_1'.

Algorithm 2.2. (for M 3)

Step 1. Calculate the total input rate λ using (6), and the input rates $\lambda_1^*, \ldots, \lambda_M^*$ using (13).

Step 2. If $R(\lambda_1^*, \lambda_2^*, \ldots, \lambda_M^*) > V$, terminate. (No feasible solution exists.)

Step 3. Use $\lambda_1^*, \ldots, \lambda_M^*$ and allocation strategy *A* to calculate $\{n_{ij}\}$, the initial feasible solution.

Step 4. Use this initial feasible solution and the Sequential Unconstrained Minimization Technique (SUMT)[9] to find the optimal solution.

MINIMIZING THE STORAGE COST—INDIVIDUAL RESPONSE TIME REQUIREMENTS

Files may have individual response time requirements. This means that inequality (14b) is replaced by L individual response time requirements (one for each file). In addition, there are many situations in which the probability of the response time for an individual request exceeding some given bound must be limited. Thus, the cumulative response time probability distribution must enter the analysis.

With inequality (14b) replaced by L individual inequalities (10), we formulate another important type of file allocation problem: allocate files minimizing the storage cost and limiting the probability that the response time for file i exceeds some given bound T to Q_i. That is,

Minimize

$$\sum_{j=1}^{M} c_j \sum_{i=1}^{L} n_{ij} \tag{15a}$$

subject to

$$P^{(i)}(t > T) \leqq Q_i, \qquad i=1, \ldots, L \tag{15b}$$

$$\sum_{i=1}^{L} n_{ij} f_i < u_j, \qquad j=1, \ldots, M \tag{15c}$$

$$n_{ij} \geqq 0, \qquad i=1, \ldots, L, \quad j=1, \ldots, M \tag{15d}$$

$$\sum_{j=1}^{M} n_{ij} = N_i, \qquad i=1, \ldots, L \tag{15e}$$

This will be refered to as the *type 3 problem.*

The type 3 problem is very complicated, exhibiting one or more nonconvex feasible regions. A detailed discussion of this problem can be found elsewhere.[6] We state a theorem concerning the optimal solution; the proof of the theorem is contained in the Appendix to this paper.

Theorem 3. The optimal solution for the type 3 problem does not necessarily obey allocation strategy A.

DISCUSSION

Service time distributions

The assumption that device service times are exponentially distributed is made for simplicity in the analysis. To be more realistic, we can assume that the device service times are generally distributed with appropriate means and variances. The resulting models have similar properties to those presented here, and similar algorithms can be derived to analyze them. Some results for the load partition problem with general service times are discussed by Chen and Buzen.[10]

Remote storage

As use of distributed computer networks becomes more widespread, it will become economical to store some files in cheap storage at remote sites rather than store them locally.

Since retrieval time for files at a remote site may not be acceptable, determining the tradeoff between storage cost and response time will be a problem.

Our models can be easily adjusted to apply to these problems. A simple application of the models is to consider each remote storage as part of the storage hierarchy. The mean service time for retrieving a file block from that remote site is assumed to be exponentially distributed with an appropriate mean.

Note that the situation considered here is conceptually different from the models developed by Chu[11] and Casey.[12] The latter models are optimized from the point of view of a computer network designer. Our model takes the point of view of a computer center manager at one site in the network who stores files at other sites.

SUMMARY

We have analyzed three types of file allocation problem in multi-level storage systems, and proposed algorithms for solving two of them. Since effects of queueing delay are given proper consideration, this analysis is more precise than previous analyses. Considering files with individual response time distribution requirements, we have presented a model (the type 3 problem) which is suitable for real-time environments.

One might expect that the optimal strategy always allocates more frequently used files to faster devices (allocation strategy A). This is true in some situations, such as in the type 2 problem. However, when each file has an individual response time requirement (the type 3 problem), this strategy may not be optimal. Moreover, in the case where storage cost is not a factor (the type 1 problem), use of this strategy is not essential.

Finally, we have briefly discussed extension to general service time distributions, allowing the resulting models to fit the practical situation better, and application of the models to use of remote storage in a distributed computer network.

ACKNOWLEDGMENT

The author is indebted to G. H. Mealy, J. P. Buzen and S. P. Bradley for their comments.

REFERENCES

1. Lowe, T. C., "The Influence of Data Base Characteristics and Usage on Direct Access File Organization," *JACM*, Vol. 15, pp. 535-548, October 1968.
2. Baskett, F., Browne, J. C., Raike, M., "The Management of a Multi-Level Non-Paged Memory System," *AFIPS Proceedings*, SJCC, pp. 459-465, 1970.
3. Collmeyer, A. J., Shemer, J. E., "Analysis of Retrieval Performance for Selected File Organization Techniques," *AFIPS Proceedings*, FJCC, pp. 201-210, 1970.
4. Ramamoorthy, C. V., Chandy, K, M., "Optimization of Memory Hierarchies in Multiprogrammed Systems," *JACM*, July 1970.
5. Arora, S. R., Gallo, A., "Optimal Sizing, Loading and Re-loading in a Multi-Level Memory Hierarchy System," *AFIPS Proceedings*, SJCC, pp. 337-344, 1971.
6. Chen, P. P. S., Mealy, G. H., "Optimal Allocation of Files with Individual Response Time Requirements," *Proceedings of the Seventh Annual Princeton Conference on Information Sciences and Systems*, Princeton University, March 1973.
7. Satty, T. L., *Elements of Queueing Theory*, McGraw-Hill Book Company, 1961.
8. Chen, P. P. S., "Optimal Partition of I-put Load to Parallel Exponential Servers," *Proceedings of the Fifth Southeastern Symposium on System Theory*, North Carolina State University, March 1973.
9. Fiacco, A. V., McCormick, G. P., *Nonlinear Programming Sequential Unconstrained Minimization Techniques*, Wiley, 1968.
10. Chen, P. P. S., Buzen, J. P., "On Optimal Load Partition and Bottlenecks" (abstract), *Computer Science Conference*, Columbus, Ohio, February 1973.
11. Chu, W. W., "Optimal File Allocation in a Multiple Computer System," *IEEE Tran. on Computers*, Vol. C-18, No. 10, October 1969.
12. Casey, R. G., "Allocation of Copies of a File in an Information Network," *AFIPS Proceedings*, SJCC, pp. 617-625, 1972.

APPENDIX

Proof of Theorem 2.1:

Assume that the optimal solution consists of $n_{i'j} \neq 0$ and $n_{ij'} \neq 0$, where $f_i > f_{i'}$ and $u_j > u_{j'}$. That is, some portion of a less frequently referenced file i' is allocated to a faster device j, and some portion of a more frequently used file i is allocated to a slower device j'. Let $a = \text{Min}[n_{i'j} f_{i'}, n_{ij'} f_i]$. Exchanging $a/f_{i'}$ blocks of file i' in device j with a/f_i blocks of file i in device j' will not change the mean request input rate to these two devices, nor will it change the mean response time. The inequality (14b) is still satisfiable, and so are (14c)-(14e). But this exchange reduces the total storage cost by the amount:

$$[(a/f_{i'})c_1 + (a/f_i)c_2] - [(a/f_i)c_1 + (a/f_{i'})c_2]$$
$$= a(f_i - f_{i'})(c_1 - c_2)/(f_{i'} f_i) > 0$$

This contradicts the assumption. Thus, the optimal solution of the type 2 problem obeys allocation strategy A.

Proof of Theorem 2.2:

Assume that at optimum $\lambda_1 = \lambda_1^a \neq \text{Min}\{\lambda_1 \mid \lambda_1 \in S\}$. We can find $\lambda_1^b \in S$ such that $\lambda_1^b < \lambda_1^a$. By Theorem 2.1, the optimal solution must use allocation strategy A. By using allocation strategy A throughout, the allocation pattern with $\lambda_1 = \lambda_1^a$ will cost more than the allocation pattern with $\lambda_1 = \lambda_1^b$ since the latter uses fewer faster storage blocks. This contradicts the optimality assumption. Thus,

$$\lambda_1' = \text{Min}\{\lambda_1 \mid \lambda_1 \in S\}.$$

Proof of Theorem 3:

We state a counterexample to the necessity of the use of allocation strategy A.

Given:

$c_1 = 10$ $\quad c_2 = 5$
$f_1 = 3$ $\quad f_2 = 2$
$N_1 = 5$ $\quad N_2 = 10$
$u_1 = 30$ $\quad u_2 = 28$
$T = 0.1$
$Q_1 = 1$ $\quad Q_2 = 0.438$

$Q_1 = 1$ indicates that any allocation pattern will satisfy the response time probability distribution constraint for file 1. The (unique) optimal solution of this example can be found to be

$n_{11} = 0$ $\quad n_{21} = 5$
$n_{12} = 5$ $\quad n_{22} = 5$

Interaction statistics from a database management system

by J. D. KRINOS

United Aircraft Research Laboratories
East Hartford, Connecticut

INTRODUCTION

Increasing attention is being paid to the measurement and evaluation of computer systems performance. A recent bibliography,[1] laying no claim to completeness, lists roughly 250 items all published in the last decade. While there is still a lack of unified "theory of computer performance" as observed by Johnson,[2] several techniques are being used—primarily analysis, simulation, monitoring, and benchmarking. The choice depends somewhat on the characteristics of the system under examination and on the purposes of the exercise.*

For effective design of new information processing systems or applications it is necessary to predict with accuracy the future behavior of the system under load. To do this, one needs a model (analytical or simulation), and reliable data. The best source of data is from performance statistics gathered during actual operation of a system of a similar type.

Once a system is developed and implemented, there is a continuing need for monitoring and controlling its usage and performance. As in other technological systems, suitable instrumentation for data collection should be a permanent feature of information systems, by being incorporated in the initial design process. The instrumentation can be external (usually a hardware monitor), an internal software data gathering program for continuous monitoring or sampling, or a combination of both approaches.

The major components affecting the performance of an online database management system are:

- its hardware and communications environment,
- its software (applications and systems),
- its users at the man-machine interface.

Historically, hardware performance has received most attention. This paper concentrates on the software and man-machine parameters and presents some empirical results derived from operation and use of the UAIMS real-time system. However, these results are dependent on the actual hardware characteristics, and care should be exercised if they are to be used in predicting performance of any other system environment.

* For a good review of the entire field the reader is referred to Lucas.[3]

ONLINE SYSTEM DESCRIPTION

United Aircraft Information Management System (UAIMS) is an interactive computing utility providing generalized database management and information processing services to various user groups throughout United Aircraft Corporation. UAIMS was developed at the Research Laboratories over a number of years. It is installed on an IBM System/360 Model 50 computer with 512K bytes of core memory and a 2314 disk storage facility. The current system[4] has been operational since 1970 and comprises:

- a teleprocessing executive and task scheduler (TPE),
- the database management system (DMS), and
- application programs (AP), both general-purpose and specialized.

As illustrated in Figure 1 UAIMS runs under IBM's operating system (OS/MVT), often concurrently with other batch-processing jobs. A typical allocation of 2314 disk-pack drives is shown in the figure. This permits a balanced multiprogramming load with one online (swapping) partition and a lower priority batch stream.

The TPE is a version of BEST (Baylor Executive System for Teleprocessing),[5] but extended and modified in-house to provide time-sharing features. It is written in assembly language and requires 80K bytes of core. Maintaining one or more online partitions, it supports interactive execution of multiple jobs from low-speed terminals.

User terminals, Sanders 620 CRT displays and Teletypes (Models 33 and 35), are connected to the computer by a communications network of cable, private line, and dial-up telephone facilities. From any terminal a user may access the computer utility by giving the proper accounting information and signing on a program. He then typically engages in a man-machine conversational interaction to enter data into the system, to formulate and

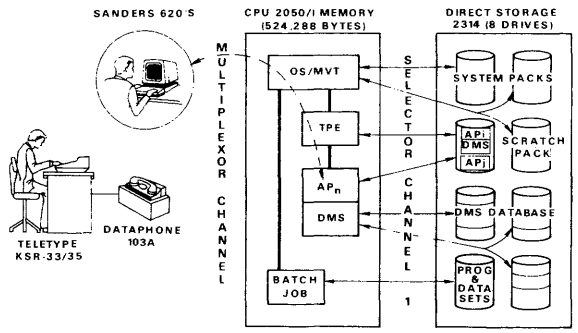


Figure 1—UAIMS schematic

execute simple interrogations or complex queries, to process or modify selected data, to produce output reports, or otherwise initiate, follow and react to programmed instructions.

All user data bases and programs reside on direct storage. The interactive application programs are brought into main memory only between the times at which they are signed on from a terminal and at which they are terminated. User programs are dynamically rolled into main core from secondary storage when requests for processing are made from terminals. A program using DMS attaches it at the time the first call is made. All access to the database is through DMS via its own query or updating language.

Interactive programs do not necessarily remain in main memory for the entire time between sign-on and sign-off. With the TPE online swapping capability, when an application program is waiting for a message from a terminal it is rolled out to secondary storage if some other program is ready for processing. Also, if processing takes too long and there are other programs waiting, a time-slicing feature comes into play to ensure equitable service to all system users. Two queues are maintained for each online partition to better control the scheduling and swapping of application programs. When programs first become ready for processing they are placed in the high priority queue to get a "quick shot" at the CPU. This allows for quick response to short transactions. If, however, the initial shot is not enough, time-slicing occurs and the transaction is then placed on the lower priority queue for the additional processing.

A typical activity sequence for two full interactive cycles is shown as an example in Figure 2.

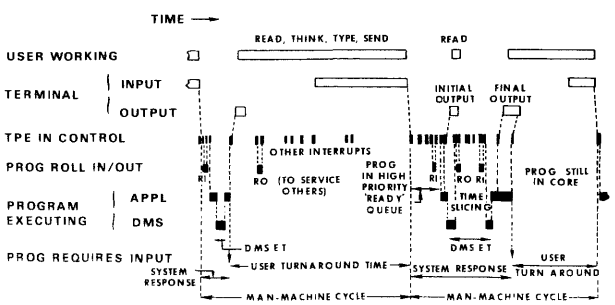


Figure 2—Example of timing sequence

DATABASE MANAGEMENT

Many recent publications discuss the concepts and capabilities of generalized database management systems (GDBMS). Surveys of existing GDBMS have been made by CODASYL[6] and others.[7,8] Fry[9] establishes the distinction between a GDBMS and other information handling system categories such as information storage and retrieval, real-time fixed-transaction systems, and conventional data processing using access methods. Olle[10] reviews current developments and calls attention to the differences between self-contained and host language systems. Early systems were usually of the self-contained variety while

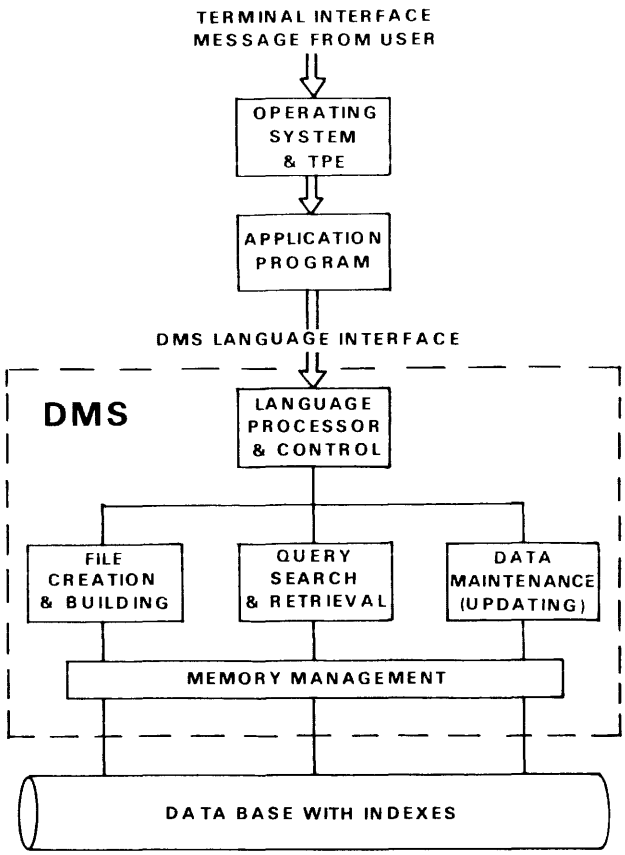


Figure 3—Information flow

later developers favor host language designs such as HIS's IDS, IBM's IMS, Cincom's TOTAL, and CODASYL's DBTG[11] specification. The host language approach is to interface the GDBMS with some other program written in a higher level language such as Cobol. This provides an open-ended flexible environment for programming but usually lacks capabilities oriented toward non-programming users, for example general query and updating.

The latest CODASYL Systems Committee report[12] expressed the wish that the self-contained and host language approaches be unified. This was attempted, to some extent, in UAIMS, which was designed to incorporate desirable features of both types of system organization.

Thus a self-contained, non-procedural, general-purpose language for data definition, query and updating is available to the programmer within a host language environment; furthermore, the same language can be employed by the non-programming user.

The Data Management Service (DMS) part of UAIMS allows a user to define his file structure logically and specify multiple keys for secondary indexing as required for fast access. The query language can be utilized to search and retrieve data from any database. File maintenance may take place at the same time as interrogation from another time-shared terminal. DMS sets up special queues to allow multiple online access and updating of the same file from different terminals. Software protect keys prevent unauthorized access to the database.

DMS acts as an extension of an application program, written in a higher-level procedural host language (Fortran, Cobol or BAL). Interfacing is achieved by means of the CALL statement. A total program is thus formed, tailored to the needs of the end user and the particular database of interest. Such a total package, including DMS, works either online or in the batch mode, and can usually be handled in a core partition of 150K bytes total. DMS itself is coded mainly in Fortran, with a few assembly language routines. Utilizing overlays, it requires 60K out of the total partition.

The functional organization of DMS is depicted in Figure 3, illustrating the information flow generated within the system by a message from a terminal user.

PERFORMANCE DATA COLLECTION

Monitoring is essential to evaluate reliability and performance and to improve the quality of service provided.[13] In UAIMS, both the TPE and DMS gather data on all transactions.

The software monitor incorporated in the TPE captures the time and origin (or destination) of each event as it occurs. Events are:

- the system going on or off the air,
- a terminal logging on or off,
- a program starting or terminating,
- a message being received from or sent to a terminal.

A postprocessor, initiated when UAIMS goes off the air, reads the captured events in sequence, calculates and stores appropriate time intervals, and produces a monitor log. This log contains information on system availability, terminal and program usage, and traffic. Abnormal program terminations and system crashes are indicated. The latter are inferred from the absence of a recorded "system off" event and may have been caused by either software or hardware malfunctions. The time of the last recorded vent provides a good approximation to the actual time of crash.

Weekly and monthly teleprocessing statistical reports are produced from the information generated by the postprocessor. These reports provide figures for operational reliability and utilization as an aid in the proper allocation of resources and to improve operating efficiency. Various timings are presented, aggregated by terminal, by account number, and by individual computer program. Some of these observed statistics will be discussed in the following sections of this paper.

DMS keeps a separate record of all database transactions. This includes elapsed time from call to return, CPU time, number of disk I/O's performed, file accessed, and identification of calling program and terminal location. Error returns and no-returns (incomplete transactions) are flagged so that a determination of the cause can be made later. A summary is prepared of all the files that have been changed by online data entry (building) or maintenance (delete, change, or add functions) during the session, together with the number of updating transactions for each and whether any were incomplete. This information helps the data administrator maintain file integrity, decide the frequency of backup copying, and recover from failures.

Statistical reports from DMS online transaction monitoring are produced monthly. They include a transaction breakdown (query, input, maintenance, etc.), and frequency for the period. Distributions of execution time, CPU time and number of disk I/O's per transaction are tabulated and mean values computed.

MAN-MACHINE CONVERSATIONAL BEHAVIOR

Online time-sharing systems have been in existence for several years. Their performance has been measured and reported in the literature. A body of knowledge exists and predictions can be made with confidence, as long as the mode of application remains unchanged.

Information management systems, as a special class of interactive systems, are a more recent occurrence. Their performance characteristics have not yet been extensively analyzed. The question is, to what extent can predictions for the terminal interface behavior be made based on general time-sharing results?

During its operational life UAIMS has served as a testbed for the study of man-machine conversational behavior within a database management environment. The timing example in Figure 2 serves to illustrate the process and to define some of the more important terms used in the analysis.

System response is the elapsed time between receiving the last character of an input message from a terminal and sending the final processor output to the channel for transmission to the terminal in response to the message. With this conservative definition, initial or intermediate responses to the user are not counted even though they can make a big difference in user satisfaction. System response time* as well as *DMS execution time (E.T.)*, as

* This is called "processor elapsed time to complete" by Stimler[14] who reserves the term "response time" to transactions requiring less than one time slice. We make no such distinction here.

PROGRAM CHARACTERISTICS (MEAN VALUES)	NON-DMS USERS	GENERAL QUERY + OTHER DMS USERS	TOOL SELECTION (DMS)	ALL PROGRAMS
CPU TIME (SEC)	0.2	1.6	3.7	1.3
NON - CPU TIME (SEC)	3.1	5.3	29.1	6.4
SYSTEM RESPONSE (SEC)	3.3	6.9	32.8	7.7
USER TURNAROUND (SEC)	14.0	17.3	26.3	17.9
CYCLE TIME (SEC)	17.3	24.2	59.1	25.6
NO. OF CYCLES/SESSION	31	37	8	26
SESSION TIME (MIN)	8.9	14.9	7.8	10.9
ONLINE USAGE LOAD	33%	50%	17%	100%

Figure 4—Program usage statistics

measured here, may also include inactive periods due to time slicing. In all cases, only a fraction of the total response time is spent for processing, in the CPU. This fraction is measured and included in the results but it is not shown diagrammatically in Figure 2.

The *user turnaround time* corresponds to what is commonly referred to as "think time;"[15,16,17,18] roughly it is the time a user requires to complete a new message after being requested to provide one by the system.

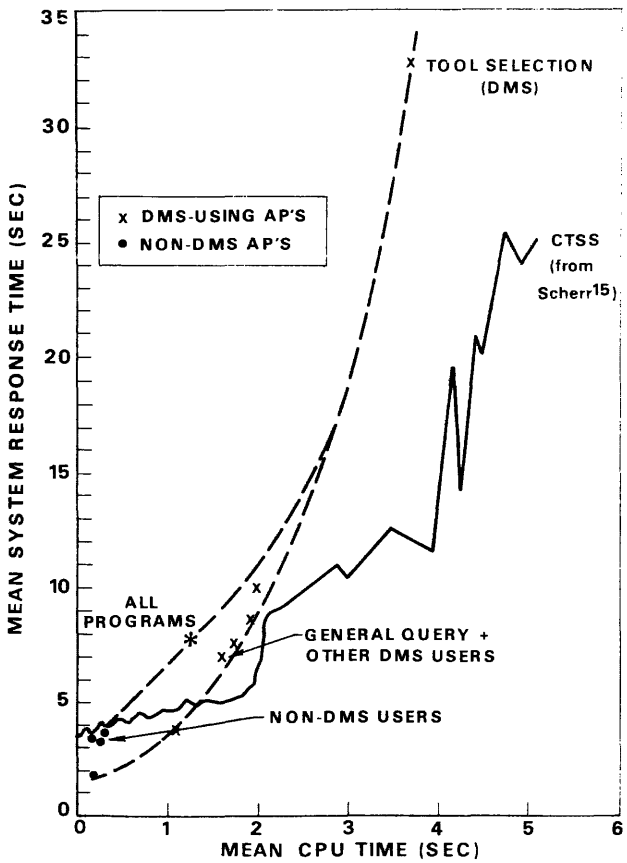


Figure 5—System response versus CPU time

Observed performance results showed considerable variation among application programs. It was possible, however, to group them into three distinct classes exhibiting significant and meaningful differences in their characteristics.

Figure 4 shows the mean values of the statistics obtained from all measurements covering a six-month period between October 1971 and April 1972. The total sign-on or active usage time for all programs was 700 hours.

The class of non-DMS users in Figure 4 comprises several online programs for programming, editing, data entry, and calculation purposes. This class is representative of the conventional time-sharing applications.

General query and other DMS users are made up of a variety of database application programs. Typically they

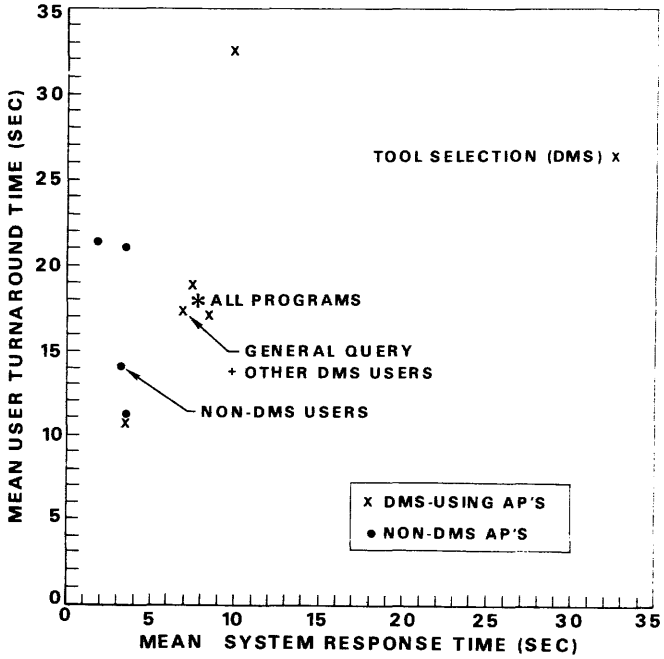


Figure 6—User turnaround versus system response

permit a terminal user to interrogate DMS-structured files using the DMS query language. After opening a file with its protection key, questions can be asked with any search conditions and their logical combinations by means of Boolean"and", "or" operations. Search criteria associate any field name with desired values using relational operators "equal", "not equal", "greater than", "less than", "between". Searches can also be made for data that start with certain characters or that contain the given string within the field. All searches take advantage of the indexing available within a file whenever possible. However, it is also possible to request sequential searches or other lengthy processing. Thus, system response times can occasionally be quite long.

A special DMS-calling application for tool selection enables production engineering designers to choose drills

and other manufacturing tools. The program first asks for certain inputs from the user and then goes through an iterative algorithm to compare the user's needs with available tools and come up with an optimum selection. During this process the program formulates and sends several queries to DMS and analyzes the answers. This results in a mean system response time to the terminal user of 32.8 seconds, the longest of any program. However, very few interactive cycles are required to solve a particular design problem. This can be seen in Figure 4 by the small number of cycles per session and consequent short terminal session time.

The tabulated results show the decisive importance of application type in predicting the system performance. The relation between mean system response and mean CPU time for various AP's is also shown graphically in Figure 5; for illustration it is compared to simulation results given by Scherr.[15] The average user turnaround time versus the mean system response time for the AP's is plotted in Figure 6. No obvious correlation seems to exist in this case. Both Figures 5 and 6 include average results for individual application programs belonging to the program classes discussed previously.

No frequency distributions of the timings at the man-machine interface are available. The longest system response times were recorded and these reached between 10 and 20 minutes on a month-by-month basis. It is believed that the distributions would follow the hyperexponential pattern, which commonly applies to time-sharing statistics. This pattern was also found empirically at the AP-DMS interface, as discussed later in this paper.

OPERATING CHARACTERISTICS

The man-machine statistics were generated during a six-month period in which the workload was light and the operating conditions remained relatively stable. The UAIMS utility was "up" four hours per working day, two in the morning and two in the afternoon. Out of the total available uptime the system was in use 76 percent of the time on the average. "In use" is defined as having one or more programs signed on for processing, and may include periods of no system activity due to user thinking, etc. The mean system response time and terminal loading are plotted on a month-by-month basis in Figure 7. As expected, there is a correlation between loading and response time. But, due to the light workload, the effects of secondary storage I/O contention, investigated by Atwood,[19] were not pronounced and did not cause appreciable delays in response.

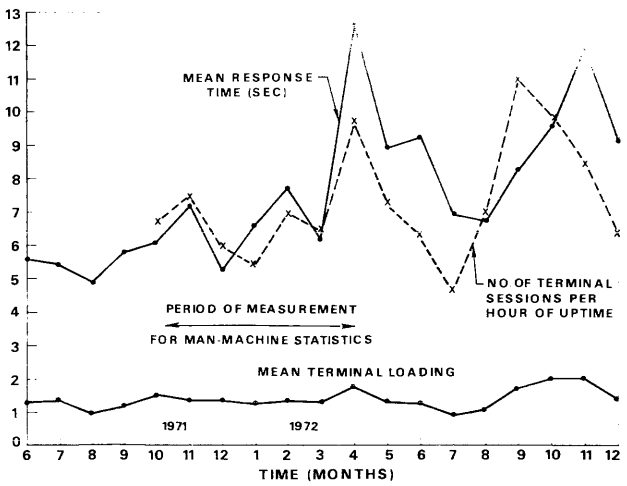


Figure 7—UAIMS workload characteristics

Average loading and traffic figures per hour of UAIMS availability (uptime) for the six month period of measurement are summarized below:

Occupancy	- of CPU	3.9 minutes
	- of computer system	23.3 minutes
	- of terminals	1.4 hours
Number of programs signed on (terminal sessions)		7
Messages received form all terminals		182
Number of DMS transactions		73
DMS utilization	- Execution time	7.4 minutes
	- Time in CPU	2.2 minutes

As explained earlier, the numbers for computer system occupancy and DMS execution time come from elapsed time measurements within programs, i.e., by summing all response times. They include, besides the computing (CPU) time shown, all secondary storage input/output I/O) and waiting periods when another program may be in control. The concept corresponds to what Stevens[20] and Yourdon[21] call "user time."

The general-purpose database management system, DMS, is by far the most important element in UAIMS from a load and performance standpoint. It accounts for 60 percent of total CPU utilization, the rest being distributed among all application programs. The DMS utilization per hour of UAIMS "in use" (as defined previously) is plotted in Figure 8 on a month-by-month basis. Also shown are two reliability measures. One is the percentage of calls to DMS that were not completed, for any reason at all. The non-completion ratio diminished as improvements and corrections were made to DMS during the time period. The average tends to be about 0.2 percent of transactions, and includes the effects of application program debugging, system failures, and cancellations made by the computer operator. System crashes (for all reasons including hardware failures) are also plotted and presently average around 5 per month.

DATA MANAGEMENT INTERFACE STATISTICS

The discussion so far has centered on the man-machine interface and its traffic pattern. It was just noted, however, that within this type of real-time information sys-

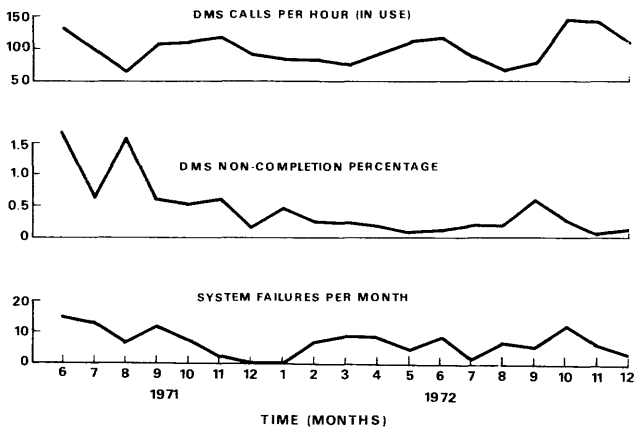


Figure 8—Utilization and reliability

tem another important interface exists. It is the one shown in Figure 3 between application programs and the DMS software.

An analysis of 70,000 DMS transactions originating from all application programs during the one-year period June 1971 through May 1972 yielded the mean response statistics and the observed relative loadings tabulated in Figure 9. Overall, the average time between call and return was 5.6 seconds. Of this time 1.8 seconds were devoted to computer processing. Data transfer between DMS and the disk files required 23.7 I/O's which amount to about 2.6 seconds.* The remaining 1.2 second represents interruptions from OS and the TPE and includes time-slicing waits (see Figure 2).

When broken down by function performed, the average response statistics vary by a factor of ten or more. In data management technology there is a performance trade-off between query and updating which poses a design choice. Data structures providing multiple access for associative searching and fast retrieval respond well to unanticipated queries but are time-consuming to build and maintain. This is illustrated in Figure 9. For the system described, the design choice proved correct since queries represent the greatest load on the system. The penalty is paid for data entry and modification transactions which take much more time. This is largely due to the many I/O's required to update the file, its indexes and directories

TYPE OF TRANSACTION	CONTROL FUNCTIONS		QUERY	NEW DATA ENTRY	MODIFICATION (UPDATING)	ALL TRANSACTIONS
	VARIOUS	OPEN FILE				
EXECUTION TIME (SEC)	1.3	3.6	5.9	6.1	13.2	5.6
CPU OCCUPANCY (SEC)	0.3	0.7	2.3	1.0	3.8	1.8
NUMBER OF DISK I/O'S	3.3	6.5	27.8	38.3	62.9	23.7
RELATIVE LOADING	5.4%	27.7%	55.3%	3.7%	7.9%	100%

Figure 9—DMS transaction responses

* Each I/O to the 2314 disk storage requires 110 ms on the average, for seek, rotational delay, and one track of data transfer.

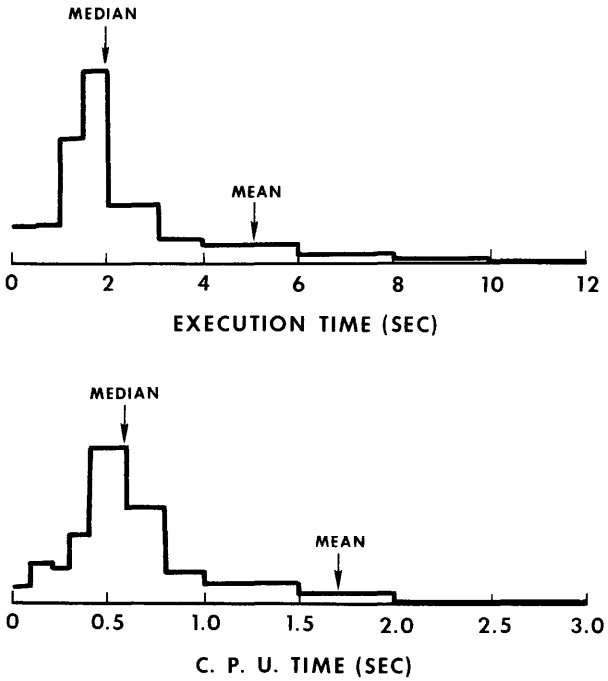


Figure 10—Observed frequency distribution for DMS responses

which facilitate the query process. At the other extreme we have control functions to open and close files, to view the data definition tables, etc. These make up a third of all transactions and have the fastest response.

A frequency distribution for all transactions, regardless of function, was derived and plotted from measurements covering the eight-month period June 1971 - January 1972 (45,000 transactions). The density and cumulative probability are shown in Figures 10 and 11 respectively. The distribution of responses follows the familiar shape which fits interarrival and service time distributions in time-sharing systems.[22,23,16,24] These distributions are close to exponential but contain too few data near the origin and too many in the tail, producing a characteristic skewness. This is seen more clearly in Figure 12 which compares the observed data with an exponential distribution by plotting the complement of the cumulative probability on a semi-log scale. The empirical curve could be mathematically approximated by fitting a hyperexponential distribution to the data, that is a linear combination of two or more ordinary exponential distributions.

	CUMULATIVE PROBABILITY								
	5%	10%	20%	50%	80%	90%	95%	99%	99.9%
EXECUTION TIME (SEC)	0.6	1.0	1.3	2.0	5.0	10	18	45	220
CPU OCCUPANCY (SEC)	0.18	0.30	0.45	0.6	1.2	2	6	20	90
NO OF DISK I/O S	0.9	1.3	3.2	5.4	17	40	80	260	600

Figure 11—Cumulative frequency distribution of all DMS transactions

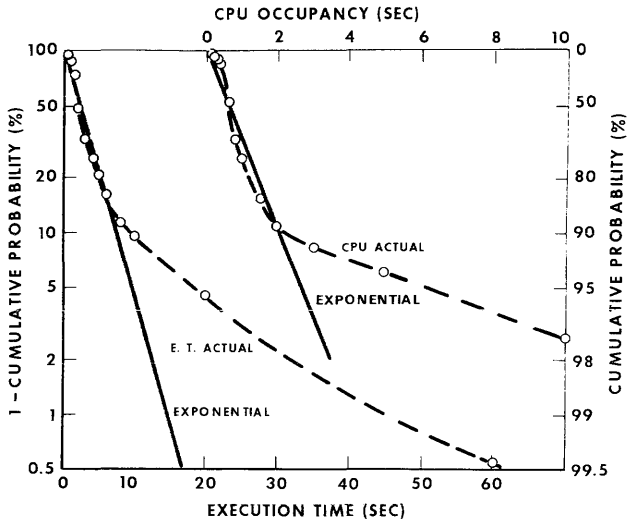


Figure 12—Comparison to the exponential distribution

CONCLUSIONS

The man-machine interactive characteristics of a database management system can be substantially different from those of general purpose time-shared systems. In this paper the timings were compared by separating UAIMS application programs into groups of non-DMS and DMS users. Data management applications on the average required more time per interactive cycle for both user turnaround to think, etc., (17 vs. 14 sec) and for system response (7 vs. 3 sec). They were also far more computer-bound (1.6 vs. 0.2 sec CPU time per cycle).

In order to predict the behavior of a new application in a real-time environment it is important to know the type of program and its expected transaction mix. This was highlighted by one particular DMS application, for tool selection, which had to be considered separately because of its totally distinct man-machine performance.

Numerically, the results obtained here are comparable to those reported by Scherr[15], Bryant[23], and others.[17,24] Depending on the application, mean "think" times range between 11 and 32 seconds. Response times, which depend on both the hardware and software, average between 2 and 33 seconds at the terminal, and between 1 and 13 seconds at the AP-DMS interface, depending on the function requested. At the latter interface, the shape of the frequency distribution conforms to the "hyperexponential" pattern described by Coffman & Wood,[22] and found by all investigators. We may infer that the same pattern holds for the man-machine parameters of system response and user turnaround, making the median values considerably less than the means (around half). Some researchers, including Parupudi & Winograd,[24] have doubted the validity of such results and attempted to "normalize" the observed data by discarding the largest 10 percent. This practice has the effect of artificially reducing the mean values and making the distributions more like exponential. We would suggest that if the hyperexponential pattern continues to be empirically confirmed for all interactive environments, then it should be accepted at its face value and further investigated by the theoreticians so that it may be better explained and understood.

REFERENCES

1. Miller, E. F., "Bibliography on Techniques of Computer Performance Analysis," *Computer (IEEE)*, Vol. 5, No. 5, pp. 39-47, September/October 1972.
2. Johnson, R. R., "Needed - A Measure for Measure," *Datamation*, Vol. 16, No. 17, pp. 20-30, December 15, 1970.
3. Lucas, H. C., Jr., "Performance Evaluation and Monitoring," *ACM Computing Surveys*, Vol. 3, No. 3, pp. 79-90, September 1971.
4. Kronos, J. D., *United Aircraft Information Management Systems (UAIMS) User's Guide - General Information*, United Aircraft Research Laboratories Report K-032131-21, July 1971.
5. Hobbs, W. F., Levy, A. H. McBride, J., "The Baylor Medical School Teleprocessing System," *AFIPS Conference Proceedings*, Vol. 32, SJCC, pp. 31-36, 1968.
6. *A Survey of Generalized Data Base Management Systems*, CODASYL Systems Committee Technical Report, ACM, New York, May 1969.
7. Angell, T., Randell, T. M., "Generalized Data Management Systems," *IEEE Computer Group News*, Vol. 2, No. 12, pp. 5-12, November 1969.
8. Prendergast, S. L., "Selecting a Data Management System," *Computer Decisions*, Vol. 4, No. 8, pp. 12-15, August 1972.
9. Fry, J. P., "Managing Data is the Key to MIS," *Computer Decisions*, Vol. 3, No. 1, pp. 6-10, January 1971.
10. Olle, T.W., "MIS Data Bases," *Datamation*, Vol. 16, No. 15, pp. 47-50, November 15, 1970.
11. CODASYL Data Base Task Group Report, ACM New York, April 1971.
12. *Feature Analysis of Generalized Data Base Management Systems*, CODASYL Systems Committee Technical Report, ACM, New York, May 1971.
13. Shemer, J. E., Robertson, J. B., "Instrumentation of Time Shared Systems", *Computer (IEEE)*, Vol. 5, No. 4, pp. 39-48, July/August 1972.
14. Stimler, S., "Some Criteria for Time Sharing System Performance," *Communications ACM*, Vol. 12, No. 1, pp. 47-52, January 1969.
15. Scherr, A. L., *An Analysis of Time Shared Computer Systems*, The MIT Press, Cambridge, Massachusetts, 1967.
16. Schrage, L., *The Modeling of Man Machine Interactive Systems*, Department of Economics and Graduate School of Business Report 6942, University of Chicago, September 1969.
17. Schwetman, H. D., Deline, J. R., "An Operational Analysis of Remote Console System," *AFIPS Proceedings*, Vol. 34, SJCC, pp. 257-264, 1969.
18. Sharpe, W. F., *The Economics of Computers*, Columbia University Press, New York, 1969.
19. Atwood, R. C., "Effects of Secondary Storage I/O Contention on the Performance of an Interactive Information Management System" *Proceedings ACM Annual Conference*, pp. 670-679, August 1972.
20. Stevens, M. E., *Problems of Network Accounting, Monitoring and Performance Measurement*, National Bureau of Standards Report PB 198 048, U. S. Department of Commerce, September 1970.

21. Yourdon, E., "An Approach to Measuring a Time Sharing System," *Datamation*, Vol. 15, No. 4, pp. 124-126, April 1969.

22. Coffman, E. G., Jr., Wood, R. C., "Interarrival Statistics for Time Sharing Systems," *Communications ACM*, Vol. 9, No. 7, pp. 500-503, July 1966.

23. Bryan, G. E., "JOSS - 20,000 Hours at a Console - A Statistical Summary," *AFIPS Proceedings*, Vol. 33, SJCC, pp. 1019-1032, 1968.

24. Parupudi, M., Winograd, J., "Interactive Task Behavior in a Time Sharing Environment," *Proceedings ACM Annual Conference*, pp. 680-692, August 1972.

EDP conversion consideration

by WILLIAM E. HANNA, JR.

Social Security Administration
Baltimore, Maryland

ABSTRACT

Conversion from one manufacturer to another is a simple phrase that embodies a myriad of changes. There are large changes in the obvious. That is, changes in hardware and software. This, however, is only the beginning. There are sweeping changes to be made in concept, DP management, machine operation, systems programming, forms and forms control, methods and procedures to mention a few. The changes in this case are not analogous at all to a change from one automobile manufacturer to another. Rather, the change is analogous to a change from an automobile to a helicopter.

The conversion, if it is successfully done, then has a sweeping effect on all operations. Special purpose leased or written software packages will not work. Extensive Systems software that allows the unity of processing on multiple machines will not work. Systems and systems programmer groups will no longer be backed up to each other nor can their efforts be coordinated and shared. This will create multiple problems on systems instead of duplicate problems for the same equipment. One for one conversion will not be a satisfactory method of program change. A complete redesign of application program systems would be necessary to best utilize the hardware and software capabilities of a new system.

The evolution of virtual machine architecture*

by J. P. BUZEN and U. O. GAGLIARDI

Honeywell Information Systems, Inc.
Billerica, Massachusetts
and
Harvard University
Cambridge, Massachusetts

INTRODUCTION

In the early 1960's two major evolutionary steps were taken with regard to computing systems architecture. These were the emergence of I/O processors and the use of multiprogramming to improve resource utilization and overall performance. As a consequence of the first step computing systems became multiprocessor configurations where nonidentical processors could have access to the common main memory of the system. The second step resulted in several computational processes sharing a single processor on a time-multiplexed basis while vying for a common pool of resources.

Both these developments introduced very serious potential problems for system integrity. An I/O processor executing an "incorrect" channel program could alter areas of main memory that belonged to other computations or to the nucleus of the software system. A computational process executing an "incorrect" procedure could cause similar problems to arise. Since abundant experience had demonstrated that it was not possible to rely on the "correctness" of all software, the multi-processing/multiprogramming architectures of the third generation had to rely on a completely new approach.

DUAL STATE ARCHITECTURE

The approach chosen was to separate the software into two classes: the first containing a relatively small amount of code which was presumed to be logically correct, the second containing all the rest. At the same time the system architecture was defined so that all functionality which could cause undesirable interference between processes was strictly denied to the second class of software.

Essentially, third generation architectures created two distinct modes of system operation (privileged/non-privileged, master/slave, system/user, etc.) and permitted certain critical operations to be performed only in the more privileged state. The critical operations restricted to privileged state typically include such functions as channel program initiation, modification of address mapping mechanisms, direct monitoring of external interrupts, etc. Experience has shown that this solution can be quite effective if the privileged software is limited in quantity, is stable in the sense that few changes are made over long periods of time, and is written by skilled professional programmers.

While this architectural principle has proven its value by fostering the development of computing systems with true simultaneity of I/O operations and high overall resource utilization, it has generated a whole host of problems of its own. These problems arise from the fact that the only software which has complete access to and control of all the functional capabilities of the hardware is the privileged software nucleus.

Probably the most serious difficulty arises in the area of program transportability since non-privileged programs are actually written for the extended machine formed by the privileged software nucleus plus the non-privileged functions of the hardware. These extended machines are more difficult to standardize than hardware machines since it is relatively easy to modify or extend a system whose primitives are in part implemented in software. This has frequently resulted in a multiplicity of extended machines running on what would otherwise be compatible hardware machines. A user who wishes to run programs from another installation which were written for a different extended machine is faced with either scheduling his installation to run the "foreign" software nucleus for some period of time or converting the programs to his installation's extended machine. Neither of these alternatives is particularly attractive in the majority of cases.

Another problem is that it is impossible to run two versions of the privileged software nucleus at the same time. This makes continued development and modification of the nucleus difficult since system programmers often have to work odd hours in order to have a dedicated machine at their disposal. In addition to the inconvenience this may cause, such procedures do not result in

* This work was sponsored in part by the Electronic Systems Division, U.S. Air Force, Hanscom Field, Bedford, Massachusetts under Contract Number F19628-70-C-0217.

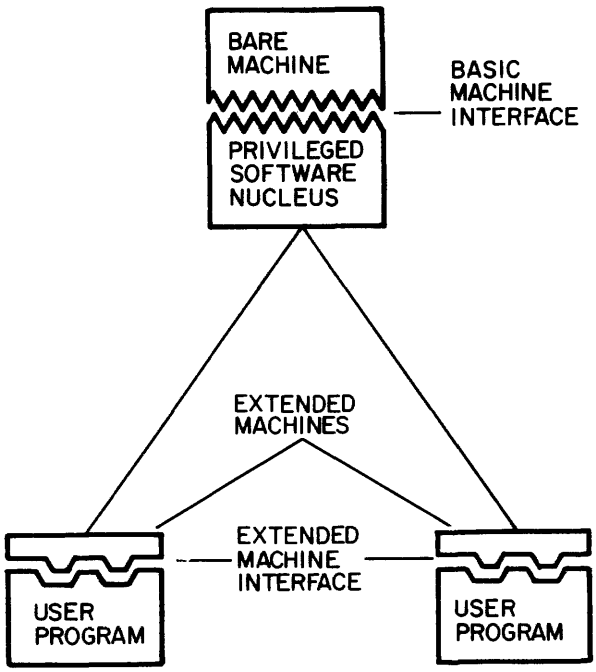


Figure 1—Conventional extended machine organization

very efficient utilization of resources since a single programmer who is modifying or debugging a system from a console does not normally generate a very heavy load.

A final problem is that test and diagnostic software has to have access to and control of all the functional capabilities of the hardware and thus cannot be run simultaneously with the privileged software nucleus. This in turn severely curtails the amount of testing and diagnosis that can be performed without interfering with normal production schedules. The ever increasing emphasis on computer system reliability will tend to make this an even more serious problem in the future.

THE VIRTUAL MACHINE CONCEPT

Figure 1 illustrates the conventional dual state extended machine architecture which is responsible for all the difficulties that were cited in the preceding section. As can be seen in the Figure, the crux of the problem is that conventional systems contain only one basic machine interface* and thus are only capable of running one privileged software nucleus at any given time. Note, however, that conventional systems are capable of running a number of user programs at the same time since the privileged software nucleus can support several extended machine interfaces. If it were possible to construct a privileged software nucleus which supported several copies of the basic machine interface rather than the extended machine interface, then a different privileged software nucleus could be run on each of the additional basic machine interfaces and the problems mentioned in the proceding section could be eliminated.

A basic machine interface which is not supported directly on a bare machine but is instead supported in a manner similar to an extended machine interface is known as a virtual machine. As illustrated in Figure 2, the program which supports the additional basic machine interfaces is known as a virtual machine monitor or VMM. Since a basic machine interface supported by a VMM is functionally identical to the basic machine interface of the corresponding real machine, any privileged software nucleus which runs on the bare machine will run on the virtual machine as well. Furthermore, a privileged software nucleus will have no way of determining whether it is running on a bare machine or on a virtual machine. Thus a virtual machine is, in a very fundamental sense, equivalent to and functionally indistinguishable from its real machine counterpart.

In practice no virtual machine is completely equivalent to its real machine counterpart. For example, when several virtual machines share a single processor on a time-multiplexed basis, the time dependent characteristics of the virtual and real machine are likely to differ significantly. The overhead created by the VMM is also apt to cause timing differences. A more significant factor is that virtual machines sometimes lack certain minor functional capabilities of their real machine counterparts such as the ability to execute self-modifying channel programs. Thus the characterization of virtual machines presented in the preceding paragraph must be slightly modified in many cases to encompass all entities which are conventionally referred to as virtual machines.

Perhaps the most significant aspect of virtual machine monitors is the manner in which programs running on a virtual machine are executed. The VMM does not perform instruction-by-instruction interpretation of these

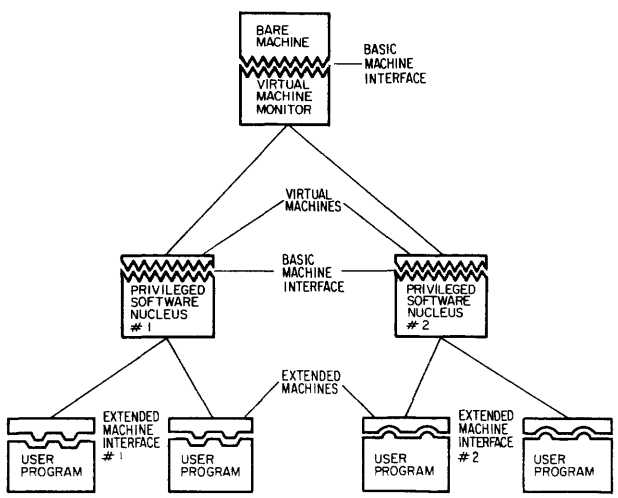


Figure 2—Virtual machine organization

* A basic machine interface is the set of all software visible objects and instructions that are directly supported by the hardware and firmware of a particular system.

programs but rather allows them to run directly on the bare machine for much of the time. However, the VMM will occasionally trap certain instructions and execute them interpretively in order to insure the integrity of the system as a whole. Control is returned to the executing program after the interpretive phase is completed. Thus program execution on a virtual machine is quite similar to program execution on an extended machine: the majority of the instructions execute directly without software intervention, but occasionally the controlling software will seize control in order to perform a necessary interpretive operation.

VIRTUAL MACHINES AND EMULATORS

Figure 2 is not intended to imply that the basic machine interface supported by the VMM must be identical to the interface of the bare machine that the VMM runs on. However, these interfaces often are identical in practice. When they are not, they are usually members of the same computer family as in the case of the original version of CP-67,[1] a VMM which runs on an IBM 360 Model 67 (with paging) and supports a virtual IBM 360 Model 65 (without paging) beneath it.

When the two interfaces are distinctly different the program which supports the virtual interface is usually called an emulator[2] rather than a virtual machine monitor. Aside from this comparatively minor difference, virtual machines and emulators are quite similar in both structure and function. However, because they are not implemented with the same objectives in mind, the two concepts often give the appearance of being markedly different.

Virtual machine monitors are usually implemented without adding special order code translation firmware to the bare machine. Thus, most VMM's project either the same basic machine interface or a restricted subset of the basic machines interface that they themselves run on. In addition, VMM's are usually capable of supporting several independent virtual machines beneath them since many of the most important VMM applications involve concurrent processing of more than one privileged software nucleus. Finally, VMM's which do project the same interface as the one they run on must deal with the problem of recursion (i.e., running a virtual machine monitor under itself). In fact, proper handling of exception conditions under recursion is one of the more challenging problems of virtual machine design.

Emulators, by contrast, map the basic machine interface of one machine onto the basic machine interface of another and thus never need be concerned with the problem of recursion. Another point of difference is that an emulator normally supports only one copy of a basic machine interface and thus does not have to deal with the scheduling and resource allocation problems which arise when multiple independent copies are supported. Still another implementation difference is that emulators must frequently deal with more complex I/O problems than virtual machine monitors do since the emulated system and the system that the emulator is running on may have very different I/O devices and channel architecture.

Modern integrated emulators[3] exhibit another difference from the virtual machine monitor illustrated in Figure 2 in that an integrated emulator runs on an extended machine rather than running directly on a bare machine. However, it is possible to create virtual machine monitors which also run on extended machines as indicated in Figure 3. Goldberg[4] refers to such systems as Type II virtual machines. Systems of the type depicted in Figure 2 are referred to as Type I virtual machines.

It should be apparent from this discussion that virtual machines and emulators have a great deal in common and that significant interchange of ideas is possible. For a further discussion of this point, see Mallach.[5]

ADDITIONAL APPLICATIONS

It has already been indicated that virtual machine systems can be used to resolve a number of problems in program portability, software development, and "test and diagnostic" scheduling. These are not the only situations in which virtual machines are of interest, and in fact virtual machine systems can be applied to a number of equally significant problems in the areas of security, reliability and measurement.

From the standpoint of reliability one of the most important aspects of virtual machine systems is the high degree of isolation that a virtual machine monitor provides for each basic machine interface operating under its control. In particular, a programming error in one privileged software nucleus will not affect the operation of

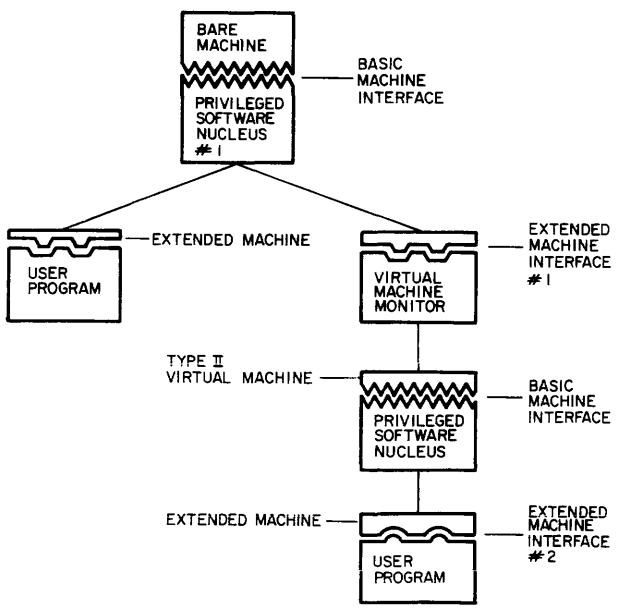


Figure 3—Type II virtual machine organization

another privileged software nucleus running on an independent virtual machine controlled by the same monitor. Thus virtual machine monitors can localize and control the impact of operating system errors in much the same way that conventional systems localize and control the impact of user program errors. In multiprogramming applications where both high availability and graceful degradation in the midst of failures are required, virtual machine systems can, for a large class of utility functions, be shown to have a quantifiable advantage over conventionally organized systems.[6]

The high degree of isolation that exists between independent virtual machines also makes these systems important in certain privacy and security applications.[7,8] Since a privileged software nucleus has, in principle, no way of determining whether it is running on a virtual or a real machine, it has no way of spying on or altering any other virtual machine that may be coexisting with it in the same system. Thus the isolation of independent virtual machines is important for privacy and security as well as system reliability.

Another consideration of interest in this context is that virtual machine monitors typically do not require a large amount of code or a high degree of logical complexity. This makes it feasible to carry out comprehensive checkout procedures and thus insure high overall reliability as well as the integrity of any special privacy and security features that may be present.

The applications of virtual machines to the measurement of system behavior are somewhat different in nature. It has already been noted that existing virtual machine monitors intercept certain instructions for interpretive execution rather than allowing them to execute directly on the bare machine. These intercepted instructions typically include I/O requests and most other supervisory calls. Hence, if it is desired to measure the frequency of I/O operations or the amount of supervisory overhead in a system, it is possible to modify the virtual machine monitor to collect these statistics and then run the system under that modified monitor. In this way no changes have to be made to the system itself. A large body of experimental data has been collected by using virtual machine monitors in this fashion.[9,10,11]

EARLY VIRTUAL MACHINES

Virtual machine monitors for computers with dual state architecture first appeared in the mid 1960's. Early VMM's[12,13] were most noteworthy for the manner in which they controlled the processor state, main memory and I/O operations of the virtual machines which ran under their control. This section presents a brief description and analysis of the special mapping techniques that were employed in these early systems.

Processor state mapping

The mapping of processor state was probably the most unusual feature of early virtual machine monitors. If a VMM did not maintain proper control over the actual state of the processor, a privileged software nucleus executing on a virtual machine could conceivably enter privileged mode and gain unrestricted access to the entire system. It would then be able to interfere at will with the VMM itself or with any other virtual machine present in the system. Since this is obviously an unacceptable situation, some mapping of virtual processor state to actual processor state was required.

The solution that was adopted involved running all virtual machine processes in the non-privileged state and having the virtual machine monitor maintain a virtual state indicator which was set to either privileged or non-privileged mode, depending on the state the process would be in if it were executing directly on the bare machine. Instructions which were insensitive to the actual state of the machine were then allowed to execute directly on the bare machine with no intervention on the part of the VMM. All other instructions were trapped by the VMM and executed interpretively, using the virtual system state indicator to determine the appropriate action in each case.

The particular instructions which have to be trapped for interpretive execution vary from machine to machine, but general guidelines for determining the types of instructions which require trapping can be identified.[14] First and most obvious is any instruction which can change the state of the machine. Such instructions must be trapped to allow the virtual state indicator to be properly maintained. A second type is any instruction which directly queries the state of the machine, or any instruction which is executed differently in privileged and non-privileged state. These instructions have to be executed interpretively since the virtual and actual states of the system are not always the same.

Memory mapping

Early virtual machine monitors also mapped the main memory addresses generated by processes running on virtual machines. This was necessary because each virtual machine running under a VMM normally has an address space consisting of a single linear sequence that begins at zero. Since physical memory contains only one true zero and one linear addressing sequence, some form of address mapping is required in order to run several virtual machines at the same time.

Another reason for address mapping is that certain locations in main memory are normally used by the hardware to determine where to transfer control when an interrupt is received. Since most processors automatically enter privileged mode following an interrupt generated transfer of control, it is necessary to prevent a process executing on a virtual machine from obtaining access to these locations. By mapping these special locations in virtual address space into ordinary locations in real memory, the VMM can retain complete control over the

actual locations used by the hardware and thus safeguard the integrity of the entire system.

Early VMM's relied on conventional paging techniques to solve their memory mapping problems. Faults generated by references to pages that were not in memory were handled entirely by the VMM's and were totally invisible to processes running on the virtual machines. VMM's also gained control after faults caused by references to addresses that exceeded the limits of a virtual machine's memory, but in this case all the VMM had to do was set the virtual state indicator to privileged mode and transfer control to the section of the virtual machine's privileged software nucleus which normally handles out-of-bounds memory exceptions. These traps were thus completely visible to the software running on the virtual machine, and in a sense they should not have been directed to the VMM at all. More advanced virtual machine architectures permit these traps to be handled directly by the appropriate level of control.[15,16]

It should be noted that the virtual machines supported by early VMM's did not include paging mechanisms within their basic machine interfaces. In other words, only privileged software nuclei which were designed to run on non-paged machines could be run under these early virtual machine monitors. Thus these VMM's could not be run recursively.

I/O mapping

The final problem which early VMM's had to resolve was the mapping of I/O operations. As in the case of main memory addresses, there are a number of reasons why I/O operations have to be mapped. The primary reason is that the only addresses which appear in programs running on virtual machines are virtual (mapped) addresses. However, existing I/O channels require absolute (real) addresses for proper operation since timing considerations make it extremely difficult for channels to dynamically look up addresses in page tables as central processors do. Thus all channel programs created within a particular virtual machine must have their addresses "absolutized" before they can be executed.

The VMM performs this mapping function by trapping the instruction which initiates channel program execution, copying the channel program into a private work area, absolutizing the addresses in the copied program, and then initiating the absolutized copy. When the channel program terminates, the VMM again gains control since all special memory locations which govern interrupt generated transfers are maintained by the VMM. After receiving the interrupt, the VMM transfers control to the address which appears in the corresponding interrupt dispatching location of the appropriate virtual machine's memory. Thus I/O completion interrupts are "reflected back" to the virtual machine in the same manner that out-of-bounds memory exceptions are.

One of the drawbacks of copying channel programs into private work areas and executing the absolutized copies is that channel programs which dynamically modify themselves during execution sometimes do not operate correctly. Hence it was not possible to execute certain self-modifying channel programs in early VMM's. However, since the majority of commonly used channel programs are not self-modifying, this lack of functionality could frequently be tolerated without serious inconvenience.

Channel program absolutization is not the only reason for VMM intervention in I/O operations. Intervention is also needed to maintain system integrity since an improperly written channel program can interfere with other virtual machines or with the VMM itself. The need for intervention also arises in the case of communication with the operator's console. This communication must clearly be mapped to some other device since there is normally only one real operator's console in a system.

A final point is that VMM intervention in I/O operations makes it possible to transform requests for one device into requests for another (e.g., tape requests to disk requests) and to provide a virtual machine with devices which have no real counterpart (e.g., a disk with only five cylinders). These features are not essential to VMM operation, but they have proven to be extremely valuable by-products in certain applications.

Summary

In summary, early VMM's ran all programs in non-privileged mode, mapped main memory through paging techniques, and performed all I/O operations interpretively. Thus they could only be implemented on paged computer systems which had the ability to trap all instructions that could change or query processor state, initiate I/O operations, or in some manner be "sensitive" to the state of the processor.[14] Note that paging *per se* is not really necessary for virtual machine implementation, and in fact any memory relocation mechanism which can be made invisible to non-privileged processes will suffice. However, the trapping of all sensitive instructions in non-privileged mode is an absolute requirement for this type of virtual machine architecture. Since very few systems provide all the necessary traps, only a limited number of these VMM's have actually been constructed.[12,13,17,19]

PAGED VIRTUAL MACHINES

It has already been noted that early VMM's did not support paged virtual machines and thus could not be run on the virtual machines they created. This lack of a recursive capability implied that VMM testing and development had to be carried out on a dedicated processor. In order to overcome this difficulty and to achieve a more satisfying degree of logical completeness, CP-67 was modified so that it could be run recursively.[18]

The major problem which had to be overcome was the efficient handling of the additional paging operation that took place within the VMM itself.[18,20] To put the problem in perspective, note that early VMM's used their page tables to map addresses in the virtual machine's memory into addresses in the real machine's memory. For example, virtual memory address A′ might be mapped into real memory address A″. However, processes running on paged virtual machines do not deal with addresses which refer directly to the virtual machine's memory the way address A′ does. Rather, an address A used by such a process must be mapped into an address such as A′ by the page table of the virtual machine. Thus, in order to run a process on a paged virtual machine, a process generated address A must first be mapped into a virtual machine memory address A′ by the virtual machine's page table, and then A′ must be mapped into a real address A″ by the VMM's page table.

In order to carry out this double mapping efficiently, the VMM constructs a composed page table (in which virtual process address A is mapped into real address A″) and executes with this map controlling the address translation hardware. When the VMM transfers a page out of memory, it must first change its own page table and then recompute the composed map. Similarly, if the privileged software nucleus changes the virtual machine's page table, the VMM must be notified so that the composed map can be recomputed.

This second consideration poses some difficulties. Since the virtual machine's page tables are stored in ordinary (virtual) memory locations, instructions which reference the tables are not necessarily trapped by the VMM. Thus changes could theoretically go undetected by the VMM. However, any change to a page table must in practice be followed by an instruction to clear the associative memory since the processor might otherwise use an out of date associative memory entry in a subsequent reference. Fortunately, the instruction which clears the associative memory will cause a trap when executed in non-privileged mode and thus allow the VMM to recompute the composed page table. Therefore, as long as the privileged software nucleus is correctly written, the operation of a virtual machine will be identical to the operation of the corresponding real machine. If the privileged software nucleus fails to clear the associative memory after changing a page table entry, proper operation cannot be guaranteed in either case.

TYPE II VIRTUAL MACHINES

VMM's which run on an extended machine interface are generally easier to construct than VMM's which run directly on a bare machine. This is because Type II VMM's can utilize the extended machine's instruction repertoire when carrying out complex operations such as I/O. In addition, the VMM can take advantage of the extended machine's memory management facilities (which may include paging) and its file system. Thus Type II virtual machines offer a number of implementation advantages.

Processor state mapping

Type II virtual machines have been constructed for the extended machine interface projected by the UMMPS operating system.[21] UMMPS runs on an IBM 360 Model 67, and thus the VMM which runs under UMMPS is able to utilize the same processor state mapping that CP-67 does. However, the instruction in the VMM which initiates operation of a virtual machine must inform UMMPS that subsequent privileged instruction traps generated by the virtual machine should not be acted on directly but should instead be referred to the VMM for appropriate interpretation.

Memory mapping

The instruction which initiates operation of a virtual machine also instructs UMMPS to alter its page tables to reflect the fact that a new address space has been activated. The memory of the virtual machine created by the VMM is required to occupy a contiguous region beginning at a known address in the VMM's address space. Thus UMMPS creates the page table for the virtual machine simply by deleting certain entries from the page table used for the VMM and then subtracting a constant from the remaining virtual addresses so the new address space begins at zero. If the virtual machine being created is paged, it is then necessary to compose the resulting table with the page table that appears in the memory of the virtual machine. This latter operation is completely analogous to the creation of paged virtual machines under CP-67.

I/O mapping

I/O operations in the original UMMPS Type II virtual machine were handled by having UMMPS transfer control to the VMM after trapping the instruction which initiated channel program execution. The VMM translated the channel program into its address space by applying the virtual machine's page map if necessary and then adding a constant relocation factor to each address. After performing this translation the VMM called upon UMMPS to execute the channel program. UMMPS then absolutized the channel program and initiated its execution.

In addition to the overhead it entailed, this mapping procedure made it impossible for the virtual machine to execute a self-modifying channel program. A recent modification to the UMMPS virtual machine monitor has been able to alleviate this situation.[22] This modification involves positioning the virtual machine's memory in real

memory so that the virtual and real address of each location is identical. This eliminates the need for channel program absolutization and thus improves efficiency while at the same time making self-modification of channel programs possible.

One of the difficulties that had to be overcome when making this change to the VMM was that the real counterparts of certain virtual machine memory locations were already being used by UMMPS. The solution that was adopted was to simply re-write the virtual machine's privileged software nucleus so that most of these locations were never used. A more detailed discussion of this point is provided by Srodawa and Bates.[22] Parmelee[11] describes a similar modification that has been made to CP-67.

SINGLE STATE ARCHITECTURE

One of the more unusual approaches to the problem of creating virtual machine architectures is based on the idea of eliminating privileged state entirely.[15,23] The proponents of this approach argue that the primary—and in fact only essential—function of privileged state is to protect the processor's address mapping mechanism. If the address mapping mechanism were removed from the basic machine interface and thereby made totally invisible to software, there would be no need to protect the mechanism and therefore no need for privileged state.

In these single state architectures all software visible addresses are relative addresses and the mechanism for translating these relative addresses to absolute addresses always concealed. That is, each software level operates in an address space of some given size and structure but has no way of determining whether its addresses correspond literally to real memory addresses or whether they are mapped in some fashion. Since all addressing including I/O is done in this relative context, there is really no need for software to know absolute address and thus no generality is lost.

The central feature of this architecture is the manner in which software level N creates the address space of software level $N+1$. Basically, level N allocates a portion of its own address space for use by level $N+1$. The location of the address space of level $N+1$ is thus specified in terms of its relative address within level N. After defining the new address space, the level N software executes a special transfer of control instruction which changes the address mapping mechanism so that addresses will be translated relative to the new address space. At the same time, control passes to some location within that new space.

Note that this special instruction need not be privileged since by its nature it may only allocate a subset of the resources it already has access to. Thus it cannot cause interference with superior levels. Level N can protect itself from level $N+1$ by defining the address space of level $N+1$ so that it does not encompass any information which level N wishes to keep secure. In particular, the address map that level N sets up for level $N+1$ is excluded from level $N+1$'s address space.

When an addressing fault occurs, the architecture traps back to the next lower level and adjusts the address map accordingly. Thus the system must retain a complete catalog of all active maps and must be able to compose and decompose them when necessary. This is relatively easy to do when only relocation/bounds maps are permitted[15] but more difficult when segmentation is involved.[23]

Since each level sees the same bare machine interface except for a smaller address space, each level corresponds to a new virtual machine. Mapping of processor state is unnecessary, mapping of memory is defined by the level N VMM relative to its own address space and is completely invisible to level $N+1$, and mapping of I/O is treated as a special case of mapping of memory. The two published reports on this architecture are essentially preliminary documents. More details have to be worked out before a complete system can be defined.

THE VIRTUAL MACHINE FAULT

The single state architecture discussed in the preceding section provides a highly efficient environment for the creation of recursive virtual machine systems. However, the basic machine interface associated with this architecture lacks a number of features which are useful when writing a privileged software nucleus. These features, which are present to varying degrees in several late third generation computer systems, include descriptor based memory addressing, multi-layered rings of protection and process synchronization primitives.

A recent analysis[24] of virtual machine architectures for these more complex systems is based on an important distinction between two different types of faults. The first type is associated with software visible features of a basic machine interface such as privileged/nonprivileged status, address mapping tables, etc. These faults are handled by the privileged software nucleus which runs that interface. The second type of fault appears only in virtual machine systems and is generated when a process attempts to alter a resource map that the VMM is maintaining or attempts to reference a resource which is available on a virtual machine but not the real system (e.g., a virtual machine memory location that is not in real memory). These faults are handled solely by the VMM and are completely invisible to the virtual machine itself.*

Since conventional architectures support only the former type of fault, conventional VMM's are forced to map both fault types onto a single mechanism. As already noted, this is done by running all virtual machine proc-

* Faults caused by references to unavailable real resources were not clearly identified in this paper. The distinctions being drawn here are based on a later analysis by Goldberg.[16]

esses in non-privileged mode, directing all faults to the VMM, and having the VMM "reflect" all faults of the first type back to the privileged software nucleus of the virtual machine.

An obvious improvement to this situation can be realized by creating an architecture which recognizes and supports both types of faults. A preliminary VMM design for a machine with this type of architecture has been proposed.[24] The design relies on static composition of all resource maps and thus requires a trap to the VMM each time a privileged process attempts to alter a software visible map. However, the privileged/non-privileged distinction within a virtual machine is supported directly by the bare machine and a privileged process is allowed to read all software visible constructs (e.g., processor state) without generating any type of fault. The major value of this design is that it can be implemented on an existing system by making only a relatively small number of hardware/firmware modifications.

DYNAMIC MAP COMPOSITION—THE HARDWARE VIRTUALIZER

The clear distinction between virtual machine faults (handled by the VMM) and process exceptions (handled by the privileged software nucleus of the virtual machine) first appeared in a Ph.D. thesis by Goldberg.[16] One of the essential ideas of the thesis is that the various resource maps which have to be invoked in order to run a process on a virtual machine should be automatically composed by the hardware and firmware of the system. Since map composition takes place dynamically, this proposal eliminates the need to generate a virtual machine fault each time a privileged process running on a virtual machine alters a software visible map. Thus the only cause of a virtual machine fault is a reference to a resource that is not present in a higher level virtual or real machine.

The thesis contains a detailed description of a "hardware virtualizer" which performs the map composition function. It includes a description of the virtualizer itself, the supporting control mechanisms, the instructions used for recursive virtual machine creation, and the various fault handling mechanisms. These details will not be considered here since they are treated in a companion paper.[25]

It is interesting to note that the work on single state architecture[15,23] can be regarded as a special case of the preceding analysis in which process exceptions caused by privileged state are completely eliminated and only virtual machine faults remain. Similarly, the earlier work of Gagliardi and Goldberg[24] represents another special case in which map composition is carried out statically by the VMM and where additional virtual machine faults are generated each time a component of the composite map is modified. By carefully identifying the appropriate functionality and visibility of all the maps involved in virtual machine operation, Goldberg's later analysis provides a highly valuable model for the design of virtual machine architectures and for the analysis of additional problems in this area.

CONCLUSION

A number of issues related to the architecture and implementation of virtual machine systems remain to be resolved. These include the design of efficient I/O control mechanisms, the development of techniques for sharing resources among independent virtual machines, and the formulation of resource allocation policies that provide efficient virtual machine operation. Many of these issues were addressed at the ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems held recently at Harvard University's Center for Research in Computing Technology.*

In view of the major commitment of at least one large computer manufacturer to the support of virtual machine systems,[27] the emergence of powerful new theoretical insights, and the rapidly expanding list of applications, one can confidently predict a continuing succession of virtual machine implementations and theoretical advances in the future.

ACKNOWLEDGMENT

We would like to express our appreciation to Dr. R. P. Goldberg for generously providing us with much of the source material that was used in the preparation of this paper.

REFERENCES

1. *Control Program-67 Cambridge Monitor System*, IBM Corporation, IBM Type III Release No. 360D-05.2.005, IBM Program Information Department, Hawthorne, New York.
2. Mallach, E. G., "Emulation—A Survey," *Honeywell Computer Journal*, Vol. 6, No. 4, 1973.
3. Allred, G., "System/370 Integrated Emulation under OS and DOS," *Proceedings AFIPS SJCC*, 1971.
4. Goldberg, R. P., "Virtual Machines—Semantics and Examples," *Proceedings IEEE International Computer Society Conference*, Boston, Massachusetts, 1971.
5. Mallach, E. G., "On the Relationship between Emulators and Virtual Machines," *Proceedings ACM SIGOPS-SIGARCH Workshop on Virtual Computer Systems*, Boston, Massachusetts, 1971.
6. Buzen, J. P., Chen, P. P., Goldberg, R. P., "Virtual Machine Techniques for Improving Software Reliability," *Proceedings IEEE Symposium on Computer Software Reliability*, New York, 1973.
7. Attansio, C. R., "Virtual Machines and Data Security," *Proceedings ACM SIGOPS-SIGARCH Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
8. Madnick, S. E., Donovan, J. J., "Virtual Machine Approach to Information System Security and Isolation," *Proceedings ACM SIGOPS-SIGARCH Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.

* Proceedings[26] may be ordered from ACM Headquarters in New York City.

9. Casarosa, V., "VHM—A Virtual Hardware Monitor," *Proceedings ACM SIGOPS-SIGARCH Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
10. Bard, Y., "Performance Criteria and Measurement for a Time-Sharing System," *IBM Systems Journal*, Vol. 10, No. 3, 1971.
11. Parmelee, R. P., *Preferred Virtual Machines for CP-67*, IBM Cambridge Scientific Center Report No. G320-2068.
12. Adair, R., Bayles, R. U., Comeau, L. W., Creasy, R. J., *A Virtual Machine System for the 360/40*, IBM Cambridge Scientific Center Report No. G320-2007, 1966.
13. Meyer, R. A., Seawright, L. H., "A Virtual Machine Time-Sharing System," *IBM Systems Journal*, Vol. 9, No. 3, 1970.
14. Goldberg, R. P., "Hardware Requirements for Virtual Computer Systems," *Proceedings Hawaii International Conference on System Sciences*, Honolulu, Hawaii, 1971.
15. Lauer, H. C., Snow, C. R., "Is Supervisor-State Necessary?," *Proceedings ACM AICA International Computing Symposium*, Venice, Italy, 1972.
16. Goldberg, R. P., *Architectural Principles for Virtual Computer Systems*, Ph.D. Thesis, Division of Engineering and Applied Physics, Harvard University, Cambridge, Massachusetts, 1972.
17. Sayre, D., *On Virtual Systems*, IBM T. J. Watson Research Laboratory, Yorktown Heights, 1966.
18. Parmelee, R. P., Peterson T. I., Tillman, C. C., Hatfield, D. J., "Virtual Storage and Virtual Machine Concepts," *IBM Systems Journal*, Vol. 11, No. 2, 1972.
19. Auroux, A., Hans, C., "Le Concept de Machines Virtuelles," *Revue Francaise d'Informatique et de Recherche Operationelle*, Vol. 15, No. B3, 1968.
20. Goldberg, R. P., *Virtual Machine Systems*, MIT Lincoln Laboratory Report No. MS-2687 (also 28L-0036), Lexington, Massachusetts, 1969.
21. Hogg, J., Madderom, P., *The Virtual Machine Facility—How to Fake a 360*, University of British Columbia, University of Michigan Computer Center Internal Note.
22. Srodawa, R. J., Bates, L. A., "An Efficient Virtual Machine Implementation" *Proceedings AFIPS National Computer Conference*, 1973.
23. Lauer, H. C., Wyeth, D., "A Recursive Virtual Machine Architecture," *Proceedings ACM SIGOPS-SIGARCH Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
24. Gagliardi, U. O., Goldberg, R. P., "Virtualizeable Architectures," *Proceedings ACM AICA International Computing Symposium*, Venice, Italy, 1972.
25. Goldberg, R. P., "Architecture of Virtual Machines," *Proceedings AFIPS National Computer Conference*, 1973.
26. Goldberg, R. P. (ed.), *Proceedings ACM SIGOPS-SIGARCH Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
27. *IBM Virtual Machine Facility/370—Planning Guide*, IBM Corporation, Publication No. GC20-1801-0, 1972.

An efficient virtual machine implementation*

by RONALD J. SRODAWA and LEE A. BATES

Wayne State University
Detroit, Michigan

INTRODUCTION

Wayne State University has traditionally combined all computational facilities, for administrative as well as research and educational uses, in one central center. At times all services have been provided under a single hardware and software system. At other times administrative services have been provided on a hardware and software system distinct from that used for research and educational services.

In recent past, these services were provided by two similar, but distinct hardware systems and two distinct operating systems. The administrative services were provided by an on-line teleprocessing system developed by Wayne State University running under the IBM OS/360 using MVT.[1] This system (called the Administrative Data Systems Teleprocessing System—ADS-TP) was run on an IBM System/360 Model 50. On the other hand, the research and educational services were provided by the WRAP system running under IBM OS/360 using MFT. (WRAP was an antecedent to the IBM TSC for OS/360 and was developed at Wayne State University.) WRAP was run on a System/360 Model 65. Two independent hardware systems were used to assure the security of the administrative data base which was on-line to the ADS-TP system.

The above configuration did not provide sufficient services for research and education. This situation was alleviated by exchanging the System/360 Model 65 running WRAP for a System/360 Model 67 half-duplex running the Michigan Terminal System (MTS). (MTS is a time-sharing system developed at the University of Michigan for the IBM System/360 Model 67. It utilizes the address translation and multi-processor features of that hardware system.)

It was decided to consolidate the above hardware configuration (a Model 50 and a Model 67 half-duplex) into a single hardware system—a System/360 Model 67 full-duplex. (A half-duplex system has a single central processor while a full-duplex system possesses two central processors.) One consideration in this decision was availability—even if several hardware components fail simultaneously a full-duplex system can generally be configured into a usable subsystem. The other consideration was utilization of the central processors—MTS was approaching saturation of its single central processor while OS/360 generally utilized very little of its central processor. As an interim measure the hardware was configured as two disjoint subsystems with one software system assigned to each subsystem. The singular advantage to this scheme was that the consolidation could be achieved with no changes to software. The goal of additional hardware availability was achieved immediately. The second goal of enhanced central processor utilization, of course, could not be attained until the two software systems could be integrated into a single system. The security of the administrative data base was still assured by the configuration of the hardware as two disjoint subsystems.

The final goal was to run MTS and OS/360 within a single software system. This was not an easy task to accomplish because of the large amount of code contained in the ADS-TP system and its heavy reliance on many of the features of OS/360. Much of the code in ADS-TP interfaced at a low level with the Supervisor and Data Management services of OS/360. The terminal access method was an original package written to interface with OS/360 at the EXCP (execute channel program) level.[2] The indexed sequential access method[3] (ISAM), partitioned datasets, the ability to catalog magnetic tape datasets, and conditional jobstep control were other features of OS/360 which were utilized by the administrative data base applications.

Three alternatives were proposed for supporting ADS-TP and MTS within a single operating system. These were:

(1) MTS and OS/360 as co-equal systems.
(2) Required OS/360 features installed into MTS.
(3) Virtual Machine support in MTS. (A virtual machine is a simulation of a hardware system upon a similar hardware system. A virtual machine does not have the poor performance typical of simulation because most of the instruction set is interpreted by the host hardware system. The most

* A preliminary version of this paper was presented at the limited attendance Workshop on Virtual Computer Systems, sponsored by ACM SIGARCH-SIGOPS and held at Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, March 26-27, 1973.

well-known virtual machine implementation is CP-67.[4]

The third alternative was chosen for several reasons:

(1) Least coding effort. OS/360 would be left unperturbed and MTS changes would be minimal.
(2) Software Reliability. OS/360 code (considered less reliable than MTS code) would not be incorporated into MTS. Most new code and all of OS/360 would operate within a single MTS task.
(3) Demonstrated feasibility. A virtual machine existed in MTS which supported OS/360 with some restrictions.
(4) Isolation. OS/360 would be isolated within a single MTS task. In addition to reliability considerations, this assures the security of the administrative data base, since input/output devices cannot be shared between tasks in MTS.

Certain performance goals were required from the resulting system. The ADS-TP system was to perform overall as if it were running on an independent System/360 Model 50. It would be quite easy to measure the central processor degradation against this goal. However, the measure of adequate teleprocessing response is much more subjective. Here a degradation of 30 percent as compared to response on the System/360 Model 67 half-duplex subsystem was considered the maximum acceptable degradation. Standard teleprocessing scripts were developed for the measurement of this degradation. These scripts originate from an MTS task running on one Model 67 subsystem while the system under test (either OS/360 under a virtual machine or OS/360 on the real machine) is run on the other subsystem. This is accomplished by connecting teleprocessing line adapters between the two subsystems. Degradation is measured in terms of the total elapsed time to complete the scripts.

IBM SYSTEM/360 MODEL 67 FEATURES

It is necessary to understand the special features of the IBM System/360 Model 67 before describing the implementation of MTS and the virtual machine. It is assumed that the reader is familiar with the basic architecture of the IBM System/360.[5] The Model 67 features are described in detail in the IBM Functional Characteristics Manual.[6]

The pertinent hardware features are:

(1) The Model 67 possesses two level address translation—segmentation and pagination. The segment is the natural unit of memory to share, since two segment table entries may point to the same page table.
(2) Channel programs must contain real memory addresses, not virtual addresses. A supervisor must translate the addresses contained in channel programs presented to it by tasks.
(3) The Model 67 does not incorporate memory protection into the segment and page tables, but rather uses the standard System/360 scheme.

MTS ARCHITECTURE

This section describes those elements of the MTS architecture which must be understood in order to read the remainder of the paper. Alexander[7,8] contains more information on this topic.

UMMPS

The heart of the MTS system is UMMPS—the supervisor. Every active MTS user, terminal or batch, is serviced by a single task independent of any other task. There are several additional tasks which provide basic system services, such as spooling. The concept of a task in MTS is similar to a task in OS/360 or a process in Multics. Tasks are always executed in problem state. That is, they cannot execute the System/360 privileged instructions. Tasks are usually executed with a non-zero protection key. This allows a storage key of zero to be used to protect memory regions from change by tasks.

The resident system is that portion of the MTS system which remains in real memory at all times—it is never paged. The major component of the resident system is UMMPS and its various tables. The resident system is assigned to the beginning of real memory, starting with the Prefix Storage Area (PSA).

Task address space

A task possesses an address space consisting of nine segments. Table I describes the contents of these segments. Shared segments are protected from task programs by a storage key of zero. Private segments are generally assigned a storage key of one. Inter-task protection of private storage is achieved by the address translation mappings.

Task input/output

An input/output operation is started on a device by means of an SVC instruction similar to Execute Channel

TABLE I—MTS Segment Usage

Segment	Attributes	Contents
0–1	not paged, shared	Resident System
2	paged, shared	Initial Virtual Memory
3	paged, private	Virtual Machine Memory
4–8	paged, private	User program and data

Program (EXCP) in OS/360. The identification of the device (a task may own more than one) and a channel program are passed as arguments to the SVC instruction. A task may either wait for the end of the operation (synchronous) or enable a task interrupt for the end of the operation on the device (asynchronous). In either case the request is made by means of a second SVC instruction.

The channel program presented to UMMPS is written in terms of virtual memory addresses. UMMPS then generates an equivalent channel program which references the data areas by their real memory addresses. Channel commands referencing data areas which straddle page boundaries may require translation into two or more chained channel commands.

Task interrupts

UMMPS provides a facility through which tasks may enable interrupts which are taken when certain asynchronous conditions are sensed by UMMPS. A task may enable end of operation, attention, and PCI (Program-Controlled Interrupts) for specific input/output devices. Tasks may also enable interrupts for abnormal events such as timer interrupts and program interrupts. The general processing at the time of the interrupt consists of pushing the current state of the task onto a stack and changing the state of the task so that it continues with the first instruction of the interrupt routine. The interrupt routine returns by means of an SVC instruction which restores the previous state of the task. A task may be interrupted by a different condition while still processing a previous interrupt, to any practical level.

THE MADDEROM VIRTUAL MACHINE

The first virtual machine for MTS was developed by Peter Madderom at the University of British Columbia. The particular implementation was unsuitable for the support of a production operating system. However, the basic architecture of all succeeding virtual machines has remained the same. This virtual machine was used to run restricted versions of OS/360, stand-alone direct access device initialization and restoration programs (DASDI and Dump/Restore), and test versions of MTS.

The SWAPTRA SVC instruction

The crux of the virtual machine in MTS is an UMMPS SVC instruction which changes the address space of the task which issues it. The arguments to this SVC instruction are an Interrupt Control Block, the right-hand half of a PSW, and a vector of sixteen full-words. In response to the SWAPTRA SVC instruction, UMMPS changes the segment table for the task so that what normally is Segment Three becomes Segment Zero and all of the other segments disappear. UMMPS then sets the general purpose registers of the task to the contents of the vector argument. The right-hand half of the task's PSW is set to the contents specified as an argument. The task is then scheduled for use of a processor in the normal fashion. These changes are depicted in Figure 1. Processing continues in this manner until the task program either issues an SVC instruction or causes a program interrupt. At that time the address space of the task reverts back to normal and the task is interrupted.

The utility of this mechanism should be obvious. Segment Three of a task's address space is used as an image of the virtual machine's address space. The SWAPTRA SVC instruction is executed by a program to enter the mode in which the virtual machine's program is run. An interrupt to the original program will be generated at just precisely that point where some function of the virtual machine must be simulated (e.g., the execution of a privileged instruction by the virtual machine program). Thus, the problem state instructions of the virtual machine's program will be executed by the Model 67 processor while privileged instructions will cause an interrupt to the program which invoked the virtual machine mode.

The virtual machine monitor

The virtual machine is initiated by loading and executing a program called the Virtual Machine Monitor. This program is a typical MTS program, except that it issues

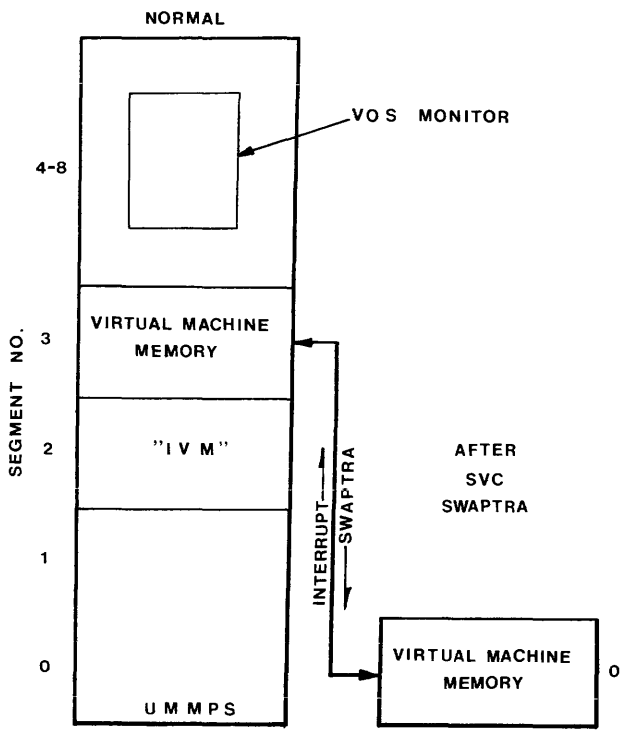


Figure 1—MTS task address space

the SWAPTRA SVC instruction. The Monitor program accepts commands from the operator which allow him to specify the virtual machine configuration and control the virtual machine. The control functions provided by the Virtual Machine Monitor parallel those available at the operator's console of a System/360 (e.g., the Initial Program Load (IPL) button, the Start and Stop buttons, the ability to display and alter the virtual machine's memory, registers, and PSW).

Input/output simulation

Input/output simulation is handled in two ways by the Virtual Machine Monitor.

Several devices (card readers, printers, and punches) may be completely simulated by the Virtual Machine Monitor. In this case the Monitor interprets the channel program and transfers data by means of the standard MTS input/output functions. Input/output done in this manner is synchronous.

Input/output operations may also be performed by acquiring a real physical device for use by the virtual machine. When the virtual machine's program executes a Start I/O (SIO) instruction for a real device, the Virtual Machine Monitor issues an UMMPS SVC instruction to request the start of the input/output operation, enables the end of operation condition for that device, and returns control to the virtual machine's program. When the Virtual Machine Monitor is entered by the interrupt signalling the end of the operation, it either simulates an input/output interrupt immediately or posts the interrupt until the virtual machine enables the channel for interrupts.

It is necessary for the Virtual Machine Monitor to generate a new channel program equivalent to the virtual machine's channel program, since the virtual machine's channel program is in terms of its restricted address space while the Monitor must use the full nine segment address space.

Interval timer

The Madderom Virtual Machine Monitor decremented the virtual machine's interval timer by one second after each second of task CPU time (including overhead). The timer had, therefore, a low resolution in comparison to the standard System/360 interval timer.

VOS VERSION 1

A new virtual machine, called VOS, for virtual machine support of OS/360, was developed at Wayne State University to support a production OS/360 system. The development of VOS-1 required 18 manmonths. VOS-1 is described in this section.

Self-modifying channel programs

The Madderom virtual machine did not allow self-modifying channel programs. Since self-modifying channel programs are used by OS/360 for the Indexed Sequential Access Method (ISAM), it was essential that they be supported, at least for direct access devices.

The major difficulty to self-modifying channel programs is that the channel program executed by the channel is not the same channel program set up by the virtual machine's program—it is a new channel program containing real memory addresses. That portion of UMMPS which translates channel programs was modified to recognize when a channel command modifies the channel program. This command, when translated, would be followed by an invalid command. When the channel program abnormally terminated, UMMPS would reprocess the virtual machine's channel program starting at the point just after the command which modified the channel program. Thus the channel program would be executed in small segments, from one self-modification to the next.

Improved channel program translation

In the original virtual machine, each channel program was translated twice, once by the Virtual Machine Monitor to addresses in Segment Three, and again by UMMPS to real addresses. Both of these steps were incorporated into the UMMPS algorithm, thus removing the translation performed by the Monitor.

Storage protection

The Madderom virtual machine did not support the System/360 storage protection mechanism. UMMPS was changed to allow a task to specify the storage keys of its storage and the protection key with which it desired to execute. The storage keys of the virtual machine's memory were then set in accordance with the Set Storage Key instructions executed by OS/360. The protection key of the task was also set in accordance with the protection key of the virtual machine's simulated PSW.

There were ramifications to the security of MTS caused by this. The Virtual Machine Monitor was forced to execute with a protection key of zero so that it could reference the virtual machine's PSA in Segment Three (OS/360 protects its PSA with a storage key of zero). Since the Monitor executes with the standard address space, it was trusted to not disturb the contents of segments 0, 1, and 2.

Interval timer

The support of a production OS/360 system required that: (1) the CPU time charged to OS/360 tasks not be increased by any overhead in the Monitor or UMMPS, (2) the time of day as computed by OS/360 remain accurate, and (3) the precision of the interval timer be increased.

The major difficulty is caused by the first two constraints. The general technique used in VOS-1 was to record the processor time charged to the MTS task in two

categories: that used by the virtual machine's program and that used by the Monitor and UMMPS. Normally the Monitor would decrement the simulated interval timer by only the former figure. However, whenever the virtual machine entered wait state, the Monitor would decrement the simulated interval timer by the elapsed time spent in wait state and the sum of the two categories of processor time charged to the MTS task. Thus the time of day as computed by OS/360 would lag during periods of sustained processor activity in the virtual machine, but would catch up when the virtual machine entered wait state.

Virtual model 67

The Madderom virtual machine supported a virtual Model 67 in addition to a standard System/360. Since OS/360 did not use these features they were removed to improve the performance of the virtual machine.

Combined SVC instructions

It was found that the Virtual Machine Monitor frequently issued a sequence of two or three UMMPS SVC instructions one after another. Since each SVC instruction is a source of overhead, due primarily to the saving

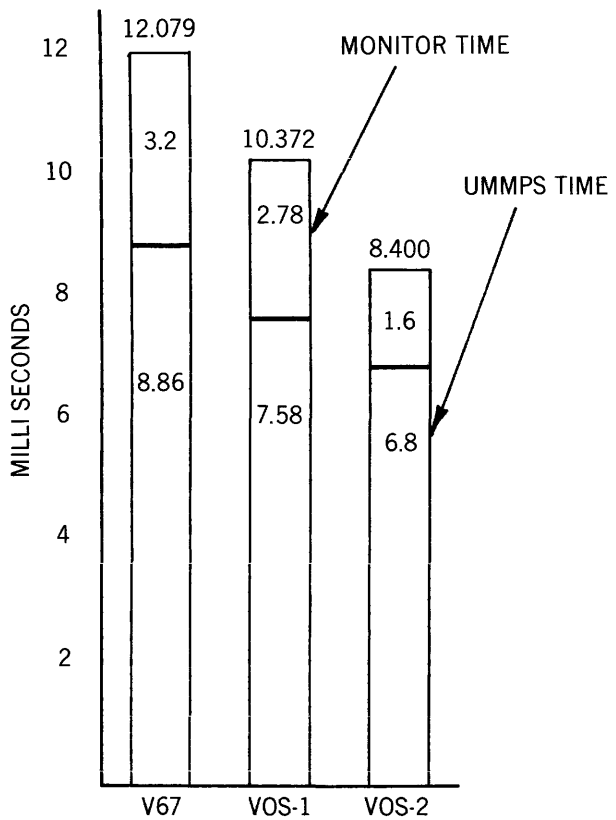


Figure 2—Overhead for simulating OS/360 I/O requests and interrupts

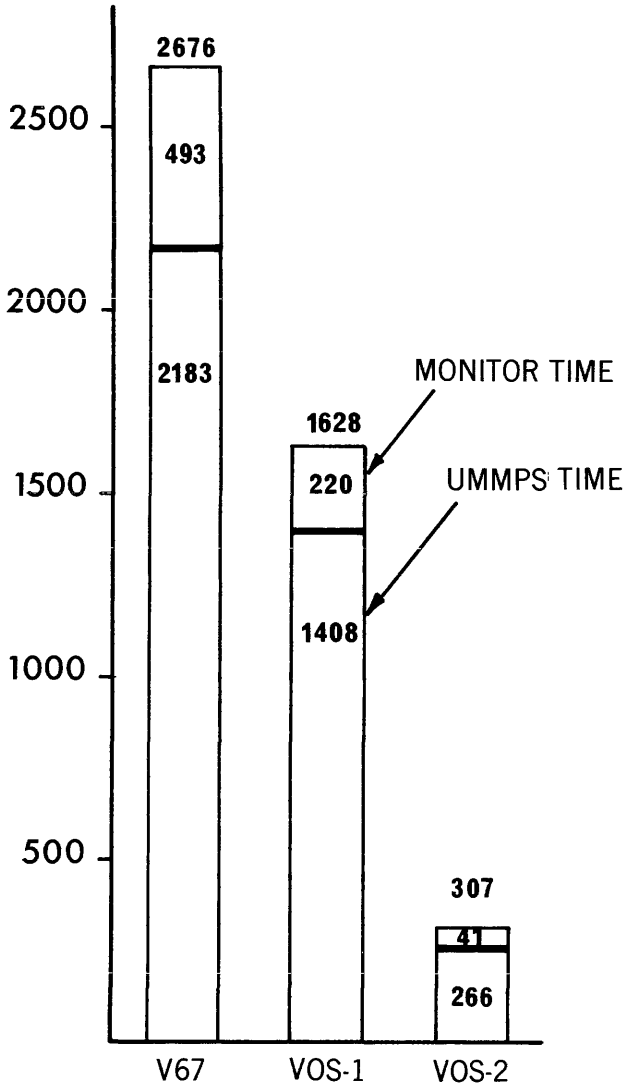


Figure 3—Overhead for simulating OS/360 SVC interrupts and privileged instructions in micro seconds

and restoring of task status, several SVC instructions were added to UMMPS which incorporated the function of popular combinations of SVC instructions. This yielded an appreciable reduction in the processor time required to perform the combined functions.

VOS VERSION 2

The second development period was initiated to respond to two shortcomings of VOS-1. The first was that performance, although noticeably improved over the Madderom Virtual Machine, was still inadequate. Figures 2 and 3 give timing comparisons between the Madderom Virtual Machine (labeled V67) and VOS-1. The second was the degree of uncertainty about the rather inelegant support of self-modifying channel programs. These design efforts required 24 manmonths and were focused in four major areas.

Allocation of OS/360 virtual memory

The first, and most complex change was the installation of what we call OS/360 with a hole. Earlier measurements showed that the majority of overhead was associated with input/output. A large amount of processor time is spent relocating channel programs. Additional time is spent relating the ending status of an input/output operation to the original channel program. Another consideration was that the method chosen to handle self-modifying channel programs was designed for the specific types of channel programs used by OS/360 and could conceivable fail with later releases of ISAM. The solution, simply stated, was to assure that OS virtual memory addresses would be identical to their assigned real memory addresses so that the relocation of channel programs would not be necessary.

The MTS system has been designed under the assumption that UMMPS and other real memory programs are allocated at the beginning of real memory. This, coupled with the fixed locations of primary and alternate Prefix Storage Areas (PSA's) dictated that no portion of OS/360 could be assigned to a real memory address less than X'052000' (We will write hexadecimal values in the notation used by the System/360 Assembler language for self-defining terms). Any area above X'052000' could be reserved for OS use.

OS/360 has its own set of constraints upon its address space. It requires that its first pages (five at WSU) begin at address X'000000' due to its use of halfword address constants and its addressing of its PSA with no base register. Therefore, it is necessary for these five pages of OS/360 to be relocated.

The entire OS/360 address space is treated in the following manner:

(1) The first five pages (X'000000' through X'004FFF') are relocated to an arbitrary set of five contiguous pages in real memory. These pages are not paged from real memory.
(2) Every page in the address space from X'005000' through X'051FFF' is mapped into a single real memory page. Steps are taken to assure that OS/360 does not use this region for any useful purpose.
(3) The remaining OS/360 address space (X'052000' through X'0FFFFF') is mapped into real memory addresses X'052000' through X'0FFFFF'. (We refer to this as virtual = real.)

The above address mapping eliminates the relocation of channel programs referencing input/output areas whose address is above X'052000'. Channel program relocation is simplified for those channel programs referencing areas below X'052000'. Since all self-modifying channel programs and their data areas are located above X'052000', they need not be relocated and require no special processing. The resulting useful OS/360 address space (parts 1 and 3 from above) is 716K in length. Figure 4 depicts the OS/360 memory allocation and Figure 2 illustrates the nineteen percent reduction in I/O overhead due to this approach.

Modifying OS/360 to conform to this address space was trivial. An ORG pseudo instruction was added to the SYSGEN macros used to generate the OS/360 I/O Supervisor and related control blocks. This change forces the last section of control blocks to start at address X'052000'. The nucleus was then reassembled and link-edited into an alternate nucleus.

Code was added to the Virtual Machine Monitor to permit selection of either the normal nucleus or the alternate nucleus when OS/360 is initially loaded.

An SVC was added to UMMPS to initialize the page table and reserve the appropriate real memory regions for the OS/360 virtual memory. The size of the region to be allocated to OS/360 is specified as an argument to the SVC—it cannot exceed a segment (one megabyte) but may be smaller if so desired.

UMMPS virtual machine support

The second design change incorporated some of the support of the virtual machine in UMMPS. The UMMPS first level interrupt handlers were modified to reduce the overhead associated with simulating SVC interrupts and non-I/O privileged instructions.

SVC interrupts and storage key instructions are always simulated in the UMMPS interrupt handlers. Instructions enabling OS to receive interrupts (LPSW and SSM) are simulated in UMMPS unless the Virtual Machine

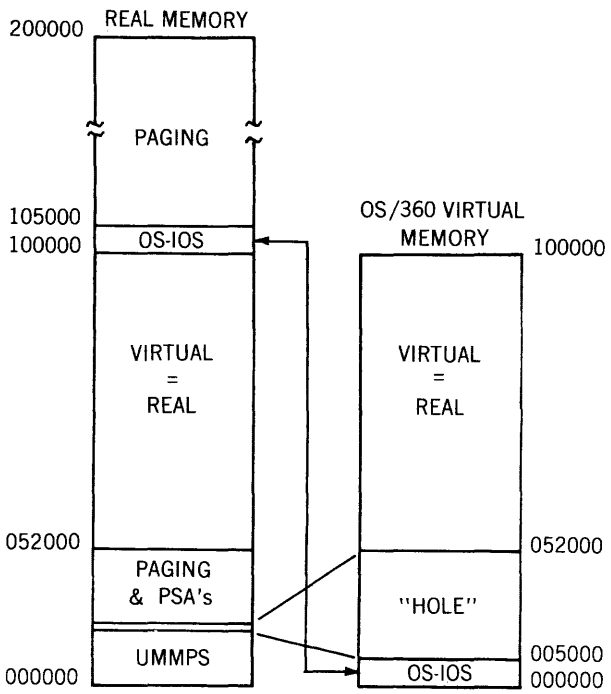


Figure 4—Allocation of OS virtual memory

Monitor indicated that an interrupt was pending. In those instances, an asynchronous exit to the Monitor is effected and the pending External or I/O Interrupt is simulated. The resulting reduction in overhead is shown in Figure 3. Whenever the UMMPS interrupt handlers encounter an event that could result in an OS/360 MVT task switch, the virtual interval timer is decremented to accurately reflect the amount of CPU time spent while the Virtual OS/360 was running. This results in accurate CPU charges for OS/360 jobs.

Memory protection alterations

The third area of design incorporated a hardware extension to functions provided by main storage. While the model 67 virtual memory addressing mechanism serves to prevent OS/360 programs (and OS) from accessing non-OS memory, the channels on a System/360 have only storage keys to limit the damage incurred by an errant channel program. In order to insure the integrity of the MTS tasks, all OS/360 protect key zero channel programs would have to be translated in their entirety to insure that only OS/360 memory would be referenced. This, of course, would result in giving up most of the performance improvements gained via the virtual=real memory arrangement.

This situation arose at the same time we were negotiating the specifications for the replacement of main storage with a Fairchild LSI Modular Main Memory. The addition of a rather simple feature enabled us to eschew complete channel program translations. Protect key eight was implemented as a sub-master protect key. This feature, which is activated by a switch on the memory controller, allows the key zero user (UMMPS) to continue accessing all of memory, and a key eight user (OS supervisor) to access all memory having a protect key of eight or higher.

MTS tasks run under a protect key of one and OS/360 assigns protect keys from 15 downward. As long as we permit no more than seven concurrent OS/360 problem programs, the OS/360 memory integrity is assured and the isolation of the two operating systems is accomplished.

Modification of OS/360

The fourth effort was directed toward the OS/360 I/O supervisor itself. A performance oriented modification was made which is not required for the virtual system to work. The change involved the simulation of a channel-end condition after OS/360 issues a stand-alone seek command to a disk drive. OS/360 normally executes a Test-I/O (TIO) privileged instruction a short time after issuing a stand-alone seek to determine if the channel is free. This change to the I/O Supervisor deletes the TIO instruction and assumes the channel is free. This removes the overhead of simulating the TIO instruction in this situation.

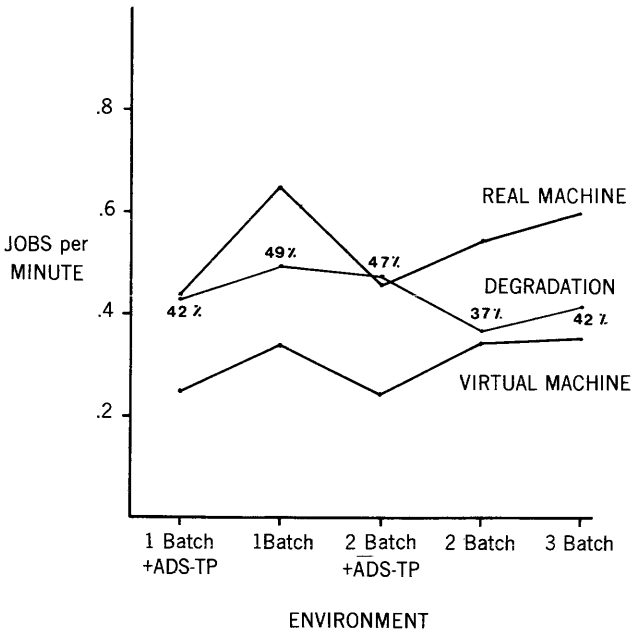


Figure 5—Cobol batch performance and degradation

CONCLUSIONS

Figures 5 and 6 summarize performance measurements taken in January, 1973. They attempt to show how the Virtual OS/360 system performs in comparison with the real OS/360 system, and the percentage of degradation. ADS-TP can be supported on a virtual system while running an OS/360 output writer and batch job stream within the 30 percent degradation permitted.

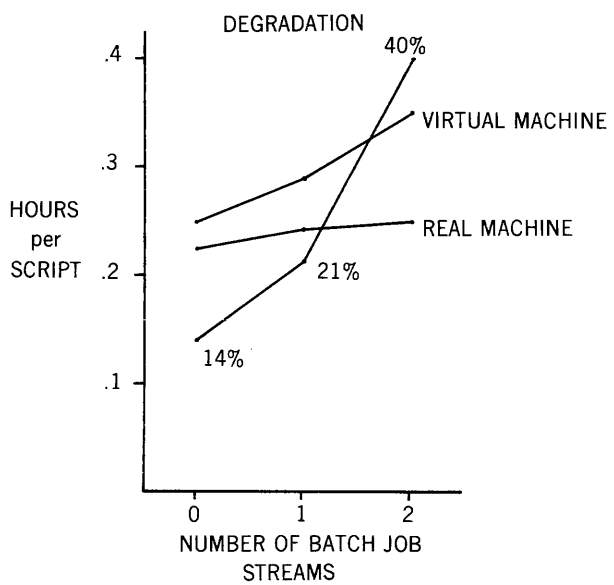


Figure 6—ADS-TP performance and degradation

The batch job streams, consisting of COBOL compilations which are primarily I/O bound, show high levels of degradation. The reduced batch throughput was considered secondary when compared to the increased CPU availability to MTS users during periods of high time-sharing activity.

ACKNOWLEDGMENTS

Henry Bodzin, Donald Cislo, Lucy Kulpa, Leonard Langkam, John Novak and Jeffrey Raben, all staff members of the Wayne State University Computing and Data Processing Center, made major contributions to the design, implementation and success of the VOS Project.

Peter Madderom of the University of British Columbia Computing Centre developed the basic architecture and software used for virtual machines in MTS.

Franklin Westervelt, the Director of the Wayne State University Computing and Data Processing Center, provided the impetus which initiated and sustained the VOS project.

REFERENCES

1. *Operating System/360 Concepts and Facilities*, IBM Systems Reference Library, Form No. GC28-6535, *IBM* Library Services Dept., White Plains, New York.
2. *Operating System/360 System Programmer's Guide*, IBM Systems Reference Library, Form No. GC28-6550, IBM Library Services Dept., White Plains, New York.
3. *Operating System/360 Data Management Services*, IBM Systems Reference Library, Form No. GC26-3746, Library Services Dept., White Plains, New York.
4. Bayles, R., et al., *Control Program-67/Cambridge Monitor System (CP-67/CMS)*, Program No. 360D05.2.005, Cambridge, Mass.
5. *System/360 Principles of Operation*, IBM Systems Reference Library, Form No. GA22-6821, IBM Library Services Dept., White Plains, New York.
6. *System/360 Model 67 Functional Characteristics*, IBM Systems Reference Library, Form No. GA27-2719, IBM Library Services Dept., White Plains, New York.
7. Alexander, M., *Timesharing Supervisor Programs*, Summer Conference Series Notes On Advanced Topics in Systems Programming, University of Michigan, Ann Arbor, Michigan, 1970.
8. Alexander, M., "Organization and Features of the Michigan Terminal System," *AFIPS Proceedings*, Vol. 40, 1972.
9. *Operating System/360 Assembler Language*, IBM Systems Reference Library, Form No. GC28-6514, IBM Library Services Dept., White Plains, New York.

Architecture of virtual machines*

by R. P. GOLDBERG

Honeywell Information Systems, Inc.
Billerica, Massachusetts

and

Harvard University
Cambridge, Massachusetts

INTRODUCTION

Virtual machine (VM) systems are a major development in computer systems design.[1] By providing an efficient facsimile of one or more complete computer systems, virtual machines have extended the multi-access, multi-programming, multi-processing systems of the past decade to be multi-environment systems as well. Thus, many of the advantages in ease of system use previously enjoyed only by application programmers have been made available to systems programmers.

Some of these advantages include support of the following activities concurrently with production uses of the system:

- improving and testing the operating system software[2]
- running hardware diagnostic check-out software[1]
- running different operating systems or versions of an operating system[3,4]
- running with a virtual configuration which is different from the real system, e.g., more memory or processors, different I/O devices[5]
- measuring operating systems[6,7]
- adding hardware enhancements to a configuration without requiring a recoding of the existing operating system(s)[3]
- providing a high degree of reliability and security/privacy for those applications which demand it.[8,9,10]

While several virtual machine systems have been constructed on contemporary machines,[3,7,11,12,13,14] the majority of today's computer systems do not and cannot support virtual machines.[15] The few virtual machine systems currently operational, e.g., CP-67, utilize awkward and inadequate techniques because of unsuitable architectures.

Recent proposals of computer architectures specifically designed for virtual machines, i.e., *virtualizable architectures*, have suffered from two weaknesses. Either they have been unable to support modern complex operating systems directly on the virtual machines[16,17] or they have been unable to avoid all of the traditional awkwardness associated with virtual machine support.[18]

A new proposal[19] called the *Hardware Virtualizer*, avoids the weaknesses of the previous designs while at the same time incorporating their strong points. Thus, the Hardware Virtualizer applies to the complete range of conventional computer systems and eliminates the awkwardness and overhead of significant software intervention. The Hardware Virtualizer may either be added to an existing computer system design or incorporated directly into a future system design.

In this paper, we develop a model which represents the mapping and addressing of resources by a process executing on a virtual machine. By deriving properties of the model, we can clarify and contrast existing virtual machine systems. However, the most important result of the model is that its proper interpretation implies the Hardware Virtualizer as the direct natural implementation of the virtual machine model. We develop some of the characteristics of the Hardware Virtualizer and then illustrate the operation through the use of a concrete example.

MODEL OF A PROCESS RUNNING ON A VIRTUAL MACHINE

In order to derive the underlying architectural principles for virtual machines, we develop a model that represents the execution of a process on a virtual machine. Since we want these principles to be applicable to the complete range of conventional computer systems—from minicomputers, through current general purpose third generation systems, and including certain future (possibly fourth generation) machines—it is necessary to produce a model which reflects the common points of all of these systems. The model should not depend on the particular map structures visible to the software of the machine under discussion. Features such as memory relocation or supervisor state are characteristics of the existing system and occur whether or not we are discussing virtual machines.

* This work was sponsored in part by the Electronic Systems Division, U.S. Air Force, Hanscom Field, Bedford, Massachusetts under Contract Number F19628-70-C-0217. A preliminary version of this paper was presented at the limited attendance Workshop on Virtual Computer Systems, sponsored by ACM SIGARCH-SIGOPS and held at Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, March 26-27, 1973.

To introduce virtual machines we must define a different, independent mapping structure which captures the notions common to all virtual computer systems. The unifying theme is the concept of a virtual machine configuration and a set of *virtual resources*. These resources, e.g., the amount of main memory in the virtual machine, are a feature of all virtual machines regardless of the particular virtual processor's form of memory relocation, etc. Thus, the key point is the relationship between the resources in the configuration of the virtual machine and those in the configuration of the real (host) machine. Only after this relationship has been fully understood need we treat the complexities introduced by the existence of any additional mapping structure.

The resource map f

We develop a model of virtual machine resource mapping by defining the set of resources $V=(v_0, v_1, \ldots, v_m)$ present in the virtual machine configuration and the set of resources $R=(r_0, r_1, \ldots, r_n)$ present in the real (host) configuration. [Resource spaces, both real and virtual, are always represented as squares in the figures.] The sets V and R contain all main memory names, addressable processor registers, I/O devices, etc. However, in the discussion which follows, for simplicity, we treat all resource names as if they are memory names. As Lauer and Snow[16] have observed, memory locations can be used to reference other resource names such as processor registers, e.g., DEC PDP-10, or I/O devices, e.g., DEC PDP-11. Therefore, no generality is lost by treating all resource names as memory names.

Since we assume no a priori correspondence between virtual and real names, we must incorporate a way of associating virtual names with real names during execution of the virtual machine. To this end, we define, for each moment of time, a function

$$f: V \rightarrow RU\{t\}$$

such that if $y \in V$ and $z \in R$ then

$$f(y)=\begin{cases} z \text{ if } z \text{ is the real name for virtual name } y \\ t \text{ if } y \text{ does not have a corresponding real name} \end{cases}$$

The value $f(y)=t$ causes a trap or fault to some fault handling procedure in the machine whose resource set is R, i.e., the machine R. For clarity we always term this event a *VM-fault*, never an exception.

We call the function f a resource map, virtual machine map, or *f-map*. The software on the real machine R which sets up the f-map and (normally) receives control on a *VM*-fault is called the *virtual machine monitor* (*VMM*).

The model imposes no requirement that the f-map be a page map, relocation-bounds (*R-B*) map, or be of any other form. However, when speaking of virtual machines we normally restrict our attention to those cases where both the virtual machine is a faithful replica of the real machine and the performance of the virtual system can be made comparable to the real one.

Recursion

The resource map model developed above extends directly to *recursion* by interpreting V and R as two adjacent *levels* of virtual resources. Then the real physical machine is level 0 and the f-map maps level $n+1$ to leven n.

Recursion for virtual systems is not only a matter of conceptual elegance or a consideration of logical closure,[16,17] it is also a capability of considerable practical interest.[18,20] In its simplest form, the motivation for virtual machine recursion is that although it makes sense to run conventional operating systems on the virtual machine, in order to test the *VMM* software on a *VM*, it is also necessary to be able to run at least a second level virtual machine.

In the discussion which follows, we use a PL/I-style qualified name tree-naming convention in which a virtual machine at level n has n syllables in its name.[18,19] This tree-name is used as a subscript for both the virtual resource space, e.g., $V_{1.1}$, and corresponding f-map, e.g., $f_{1.1}$.

Thus, if

$$f_1: V_1 \rightarrow R$$

$$f_{1.1}: V_{1.1} \rightarrow V_1$$

Then a level 2 virtual resource name y is mapped into $f_1(f_{1.1}(y))$ or f_1 *of* $_{1.1}(y)$.* See Figure 1a.

In this function, f_1 *of* $_{1.1}$, we identify two possible faults:

(1) The level 2 resource (virtual machine) fault to the *VMM* of level, 1, i.e., $f_{1.1}(y)=t$. See Figure 1b.
(2) The level 1 resource (virtual machine) fault to the *VMM* of level 0 (the real machine), i.e., f_1 *of* $_{1.1}(y)=t$. See Figure 1c.

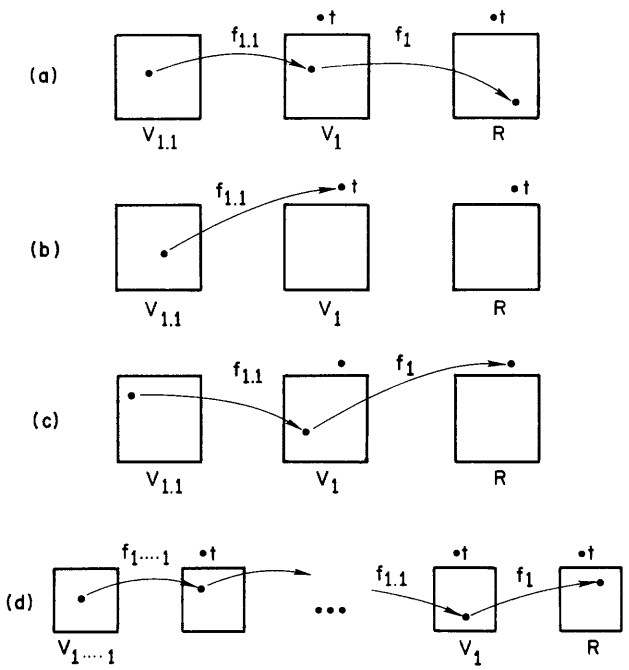


Figure 1—Recursive f-map

* "o" is the conventional function composition operator of mathematics

In general, a composed f-map may cause either fault. However, there exists a class of maps, called *inclusive maps*, which can only cause the first fault (level 2 fault). The relocation-bounds map (R-B map) is inclusive but the page map is not. The inclusive property implies the possibility of simple recursive implementation.[16,19]

For the general case of level n recursion, we have n-level virtual name y being mapped into

$$f_1 \text{ of } {}_{1.1}o \ldots \text{of } {}_{1.\ldots.1}(y).$$

See Figure 1d.

The present model may be used to describe the proposals of Lauer and Snow[16] and of Lauer and Wyeth[17] for single state recursive virtual machines. In the former case, the map is $f=R-B$; in the latter case, it is $f=$segmentation. See discussion of Table I.

The process map ϕ

The model as currently developed represents only the mapping of resources in a computer system. This machinery is sufficient to discuss virtualization of certain mini-computers, e.g., DEC PDP-8, which do not exhibit any local mapping structure. However, most current (third generation) general purpose systems have additional *software-visible hardware maps*. This additional structure may be as simple as supervisor/problem states (IBM System/360) and relocation-bounds registers (DEC PDP-10 and Honeywell 6000), or as complex as segmentation-paging-rings[21] (Multics—Honeywell 6180). In future fourth generation systems, the maps will likely be even more complex and might feature a formal implementation of the process model[22,23] in hardware-firmware.

The crucial point about each of these hardware (supported) maps is that they are software visible. In certain systems, the visibility extends to non-privileged software.[15] However, in all cases the maps are visible to privileged software.[18]

Typically, an operating system on one of these machines will alter the map information before dispatching a user process. The map modification might be as simple as setting the processor mode to problem state or might be as complex as changing the process's address space by switching its segment table. In either case, however, the subsequent execution of the process and access to resources by it will be affected by the current local map. Therefore, in order to faithfully model the running of processes on a virtual machine, we must introduce the local mapping structure into the model.

We develop a model of the software-visible hardware map by defining the set of *process names* $P=\{p_0, p_1, \ldots, p_j\}$ to be the set of names addressable by a process executing on the computer system. [Process spaces are always represented as circles in the figures.] Let $R=\{r_0, r_1, \ldots, r_n\}$ be the set of (real) resource names, as before.

Then, for the active process, we provide a way of associating process names with resource names during process execution. To this end, via all of the software visible hardware mapping structure, e.g., supervisor/problem state, segment table, etc., we define, for each moment of time, a function

$$\phi\colon P \rightarrow RU\{e\}$$

such that if $x \in P$, $y \in R$, then

$$\phi(x) = \begin{cases} y \text{ if } y \text{ is the resource name for process name } x \\ e \text{ if } x \text{ does not have a corresponding resource.} \end{cases}$$

The value $\phi(x)=e$ causes an exception to occur to some exception handling procedure, presumably to a privileged procedure of the operating system on this machine. To avoid confusion with VM-faults (see above), process traps will always be called *exceptions*.

We call the function ϕ a process map or *ϕ-map*. The term process map is applied regardless of what form the ϕ-map takes. In future (fourth generation) systems, ϕ might actually represent the firmware implementation of processes, although this is not necessary. The important point about ϕ is that unlike f, which is an *inter-level* map, ϕ is a local or *intra-level map* and does not cross a level of resource mapping.

Running a virtual machine: f o ϕ

Running a process on a virtual machine means running a process on a configuration with virtual resources. Thus, if a process $P=\{p_0, p_1, \ldots, p_j\}$ runs on the virtual machine $V=\{v_0, v_1, \ldots, v_m\}$ then

$$\phi\colon P \rightarrow VU\{e\}$$

as before, with virtual resource names, V, substituted for real ones in the resource range of the map.

The virtual resource names, in turn, are mapped into their real equivalents by the map, f: $V \rightarrow R$. Thus, a process name x corresponds to a real resource $f(\phi(x))$. In general, process names are mapped into real resource names under the (composed) map

$$f \text{ o } \phi\colon P \rightarrow RU\{t\}U\{e\}.$$

This (composed) map can fail to take a process name into a real resource name in one of two ways. In the event of a process name exception (Figure 2a), control is given, without VMM knowledge or intervention, to the privileged software of the operating system within the same level. A virtual name fault, however, causes control to pass to a process in a lower level virtual machine, without the operating system's knowledge or intervention (Figure 2b). While this fault handling software in the VMM is not subject to an f-map since it is running on the real machine, it is subject to its ϕ-map just as any other process on the machine.

The ϕ-map may be combined with the recursive f-map result to produce the "general" composed map

$$f_1 \text{ of } {}_{1.1} \text{ o } \ldots \text{ of } {}_{1.1 \ldots 1.1} \text{ o } \phi.$$

Thus, for virtual machines, regardless of the level of recursion, there is only one application of the ϕ-map followed by n applications of an f-map. This is an important result that comes out of the formalism of distinguishing the f and ϕ maps. Thus, in a system with a complex ϕ-map but with a

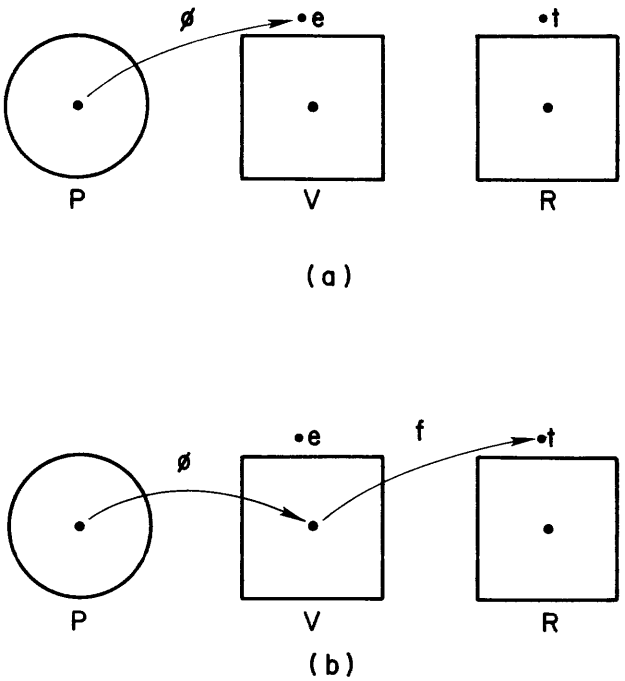


Figure 2—Process exception and VM-fault

simple f-map, n-level recursion may be easy and inexpensive to implement.

In the model presented, f-maps map resources of level $n+1$ into resources of level n. It is equally possible to define an f-map in which resources of level $n+1$ are mapped into process names of level n (which are then mapped into resource names of level n). This new f-map is called a Type II f-map to distinguish it from the Type I f-map which is discussed in this paper.[19,24]

Interpretation of the model

The model is very important for illustrating the existence of two very different basic maps in virtual machines. Previous works have not clearly distinguished the difference or isolated the maps adequately. The key point is that f and ϕ are two totally different maps and serve different functions. There is no *a priori* requirement that f or ϕ be of a particular form or that there be a fixed relationship between them. The ϕ-map is the interface seen by an executing program whereas the f-map is the interface seen by the resources. In order to add virtual machines to an existing computer system, ϕ is already defined and only f must be added. The choice of whether the f-map is *R-B*, paging, etc., depends upon how the resources of the virtual machines are to be used. In any case, the f-map must be made recursive whereas ϕ need not be.

If a new machine is being designed, then neither ϕ nor f is yet defined. ϕ may be chosen to idealize the structures seen by the programmer whereas f may be chosen to optimize the utilization of resources in the system. Such a "decoupled" view of system design might lead to systems with ϕ = segmentation and f = paging.

Another intrinsic distinction between the maps is that the f-map supports *levels* of resource allocation between virtual machines, while the ϕ-map establishes *layers* (rings, master/slave mode) of privilege within a single virtual machine.

The virtual machine model may be used to analyze and characterize different virtual machines and architectures.[19] As can be seen from Table I, none of the existing or previously proposed systems provides direct support of completely general virtual machines. CP-67 has a non-trivial ϕ-map but no direct hardware support of the f-map; the approach of Lauer and Snow provides direct hardware support of the f-map but has a trivial ϕ-map, i.e., ϕ = identity. Therefore, CP-67 must utilize software plus the layer relationship of the ϕ-map to simulate levels, whereas Lauer and Snow must utilize software plus the level relationship of the f-map to simulate layers.*

The Gagliardi-Goldberg "Venice Proposal" (VP)[18] supports both the layer and level relationships explicitly. However, since the VP does not directly provide hardware support for f (it supports ϕ and $f \circ \phi$), certain software intervention is still required.

In the next section, we shall discuss a design, called the *Hardware Virtualizer* (*HV*), which eliminates the weaknesses of the previous designs. As can be seen from Table I, the *HV* is based directly upon the virtual machine model which we have developed.

HARDWARE VIRTUALIZER (HV)

Despite the value of the virtual machine model in providing insight into existing and proposed systems, perhaps its most important result is that it implies a natural means of implementing virtual machines in all conventional computer systems. Since the f-map and ϕ-map are distinct and (possibly) different in a virtual computer system, they should be repre-

TABLE I - COMPARISON OF SYSTEMS USING VIRTUAL MACHINE MODEL

SYSTEM	ø	f	$\phi_E = f \circ \phi$	f ∘ : COMPOSER	DIRECT ACCESS TO : PERMITTED? READ	WRITE	RECURSION AND RELOCATIVE COMPOSTR	RECURSIVE VIRTUAL MACHINE STRUCTURE HARDWARE SUPPORTED
IBM CP-67[3]	Hardware (III generation)	Software	Software support to translated ø	Software	No	No	Software composition	---
Gagliardi-Goldberg[18] VP	Hardware (complex IV generation)	Software	Hardware support to store translated ø	Hardware	Yes	No	Hardware assisted composition	Tree
Lauer-Snow[16]	---	Hardware (relocation-bounds)	---	---	---	---	Direct hardware static composition	Stack
Lauer-Wyeth[17]	---	Hardware (segmentation, paging)	---	---	---	---	Direct hardware dynamic composition	Stack
Goldberg PV[19]	Hardware (completely arbitrary)	Hardware (completely arbitrary)	Evaluated dynamically	Direct hardware dynamic composition	Yes	Yes	Direct hardware dynamic composition	Tree

* This is not to suggest that the Lauer and Snow approach is inferior. It is only less general in that it will not support modern operating systems running directly on the individual virtual machines.

sented by independent constructs. When a process running on a virtual machine references a resource via a process name, the required real resource name should be obtained by a dynamic composition of the f-map and ϕ-map at execution time. Furthermore, the result should hold regardless of recursion or the particular form of f and ϕ. We call a hardware-firmware device which implements the above functionality a *Hardware Virtualizer* (*HV*). The *HV* may be conceptually thought of as either an extension to an existing system or an integral part of the design of a new one.

HV design and requirements

The design of a Hardware Virtualizer must consider the following points:

(1) The database to store f
(2) A mechanism to invoke f
(3) The mechanics of map composition
(4) The action of a *VM*-fault.

In the discussion which follows, we shall develop the basis for a Hardware Virtualizer design somewhat independently of the particular form of the f-map or ϕ-map under consideration. We assume that the ϕ-map is given (it could be the identity map) and we discuss the additional structure associated with the f-map. Although we shall refer to certain particular f-map structures, such as the *R-B* or paging form of memory map, the actual detailed examples are postponed until later.

Database to represent f

The *VMM* at level n must create and maintain a database which represents the f-map relationship between two adjacent levels of virtual machine resources, namely level $n+1$ to level n. This database must be stored so that it is invisible to the virtual machine, i.e., level $n+1$, including the most

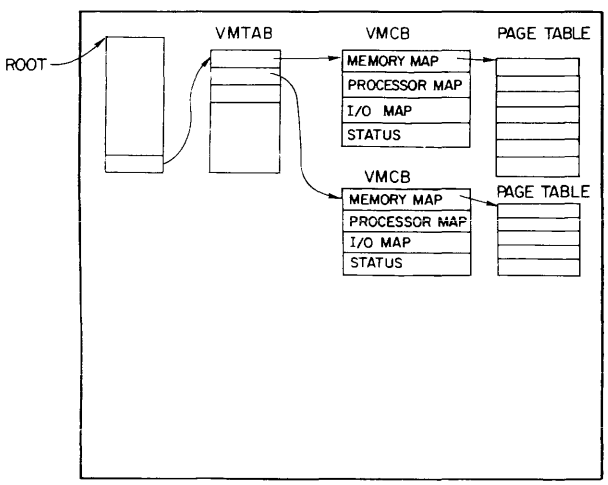


Figure 3—The VMTAB and VMCB's

privileged software. Let us assume that for economic reasons[18] the database must be stored in main memory. Then f may not be in the (virtual) memory of level $n+1$, but it must be in the (virtual) memory of level n.

The only requirement on where the f-map is stored in level n memory is that it be possible for the *HV* to locate it by applying a deterministic algorithm from the beginning (ROOT) of level n memory. The f-maps corresponding to different virtual machines at the same level may be identified either implicitly[16] or explicitly.[18] For explicit identification, we assume a virtual Machine Table (VMTAB), the ith entry of which points to the Virtual Machine Control Block (VMCB) of virtual machine i (supported at level n). See Figure 3.

The VMCB provides the representation of the f-map for the virtual machine. It contains the memory map, processor map, and I/O map. In addition, there may be other status and/or accounting data for the virtual machine.* The specific form of the VMCB is dependent upon the f-map actually used, e.g., *R-B*, paging, etc.

Additional information possibly kept in the VMCB includes capability information for the virtual processor indicating particular features and instructions, present or absent. These capability bits include, for example, scientific instruction set or virtual machine instruction set (recursion). If recursion is supported, then the VMCB must include sufficient information to automatically restart a higher level virtual machine on a lower level *VM*-fault (Figure 1c).

Mechanism to invoke f

In order to invoke the f-map, the *HV* requires an additional register and one instruction for manipulating it. The register is the virtual machine identifier register (VMID) which contains the "tree name" of the virtual machine currently executing. The VMID is a multisyllabic register, whose syllables identify all of the f-maps which must be composed together in order to yield a real resource name. The new instruction is LVMID (load VMID) which appends a new syllable to the VMID register. This instruction should more accurately be called *append* VMID but LVMID is retained for historical reasons.

For the hardware virtualizer design to be successful, the VMID register (and the LVMID instruction) must have four crucial properties.[18,19]

(1) The VMID register *absolute* contents may neither be read nor written by software.
(2) The VMID of the real machine is the null identifier.
(3) Only the LVMID instruction may append syllables to the VMID.
(4) Only a *VM*-fault (or an instruction which terminates the operation of a virtual machine) may remove syllables from the VMID.

* As noted earlier, mapping of I/O and other resources may be treated as a special case of the mapping of memory. Under these circumstances, the VMCB reduces to the memory map component.

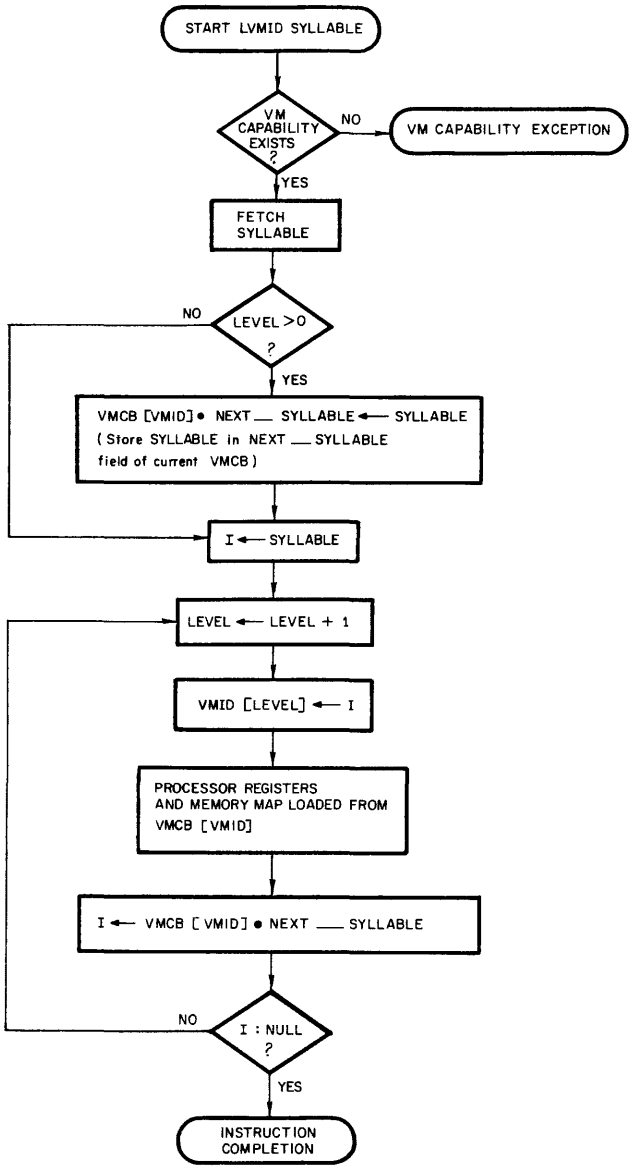


Figure 4—LVMID instruction

Figure 4 sketches the operation of the LVMID instruction while avoiding implementation details related to a specific choice of map. In the flowchart, we use the VMID as a subscript to indicate the current control block, VMCB [VMID]. Thus SYLLABLE, the operand of the LVMID instruction, is stored in the NEXT_SYLLABLE field of the current VMCB. SYLLABLE is appended to the VMID and this new virtual machine is activated. If the NEXT_SYLLABLE field of the new VMCB is NULL, indicating that this level of machine was not previously active, then the LVMID instruction completes and execution continues within this virtual machine. Otherwise, if it is not null, the lower level was previously active and was suspended due to a *VM*-fault at a still lower level. In this case, execution of the LVMID instruction continues by appending the NEXT_SYLLABLE field of the new VMCB to the VMID.

Map composer

A map composer is needed to provide the dynamic composition of the ϕ-map (possibly indentity) and the active f-maps on each access to a resource. The ϕ-map is known and the active f-maps, i.e., the VMCB's, are determined from the VMID register. Figure 5 sketches the map composition mechanism while avoiding implementation details related to specific choice of maps. As can be seen, the composer accepts a process name P and develops a real resource name R or causes a *VM*-fault.

VM-fault

A *VM*-fault occurs when there does not exist a valid mapping between two adjacent levels of resources. As shown in Figure 5, a *VM*-fault causes control to be passed to the *VMM*

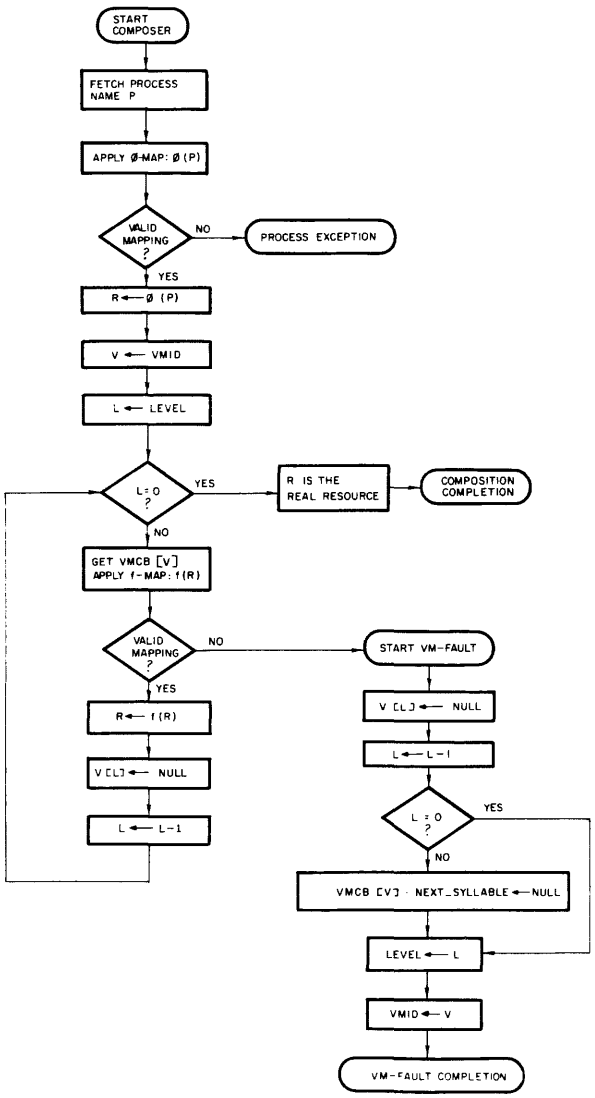


Figure 5—Map composition and VM-fault

superior to the level which caused the fault. This is done by removing the appropriate number of syllables from the VMID.

Preformance assumptions

The performance of the Hardware Virtualizer depends strongly upon the specific f-map, ϕ-map, and HV implementation technique used. However, there are basic reasons why processes can execute on a virtual machine with efficiency approaching that of the real machine. Most current systems which employ memory mapping (in the ϕ-map) make design assumptions concerning program behavior. We will observe that these assumptions are applicable to virtual machines as well.

From the initial notion of "program locality", Madnick[25] has generalized and identified two specific aspects of locality.

(1) Temporal locality
If the logical addresses $\langle a_1, a_2, \ldots \rangle$ are referenced during the time interval $t-T$ to t, there is a high probability that these same logical addresses will be referenced during the time interval t to $t+T$.[25]

(2) Spatial locality
If the logical address a is referenced at time t, there is a high probability that a logical address in the range $a-A$ to $a+A$ will be referenced at time $t+1$.[25]

In modern operating systems, because of the cost to "start up" a process or to change the ϕ-map, it is likely that the scheduler and dispatcher will enforce an additional locality:

(3) Process locality
If the ϕ-map value of the process executing at time t is ϕ_*, then there is a high probability that it will be ϕ_* at time $t+1$.

Virtual machines and the Hardware Virtualizer add a new notion.

(4) Virtual machine locality
If the VMID of the currently executing virtual machine at time t is $x_1.x_2.\ldots.x_{n-1}.x_n$, then there is a high probability that the VMID will be $x_1.x_2.\ldots.x_{n-1}.x_n$ at time $t+1$. Furthermore the VMID may change only on a VM-fault or an LVMID instruction.

Combining all of these locality notions, we determine that with proper implementation, multi-level recursive virtual machines need not have significantly different performance from real machines. Another way of phrasing this observation is:

Temporal and spatial locality are name invariant.

Regardless of what a block of memory is called, or how many times it gets renamed (via composed f-maps) there is still an intrinsic probability of reference to it by an executing program. Thus, a virtual machine supported by a map composer and associative store should enjoy comparable performance to the real machine.[19]

If the f-map and ϕ-map are sufficiently simple then the associator may not be needed. For example, if $f=R\text{-}B$, $\phi=$ identity, then it may be sufficient for the HV to provide "invisible scratchpad registers" to maintain statically composed $R\text{-}B$ values which are altered only on a level change.[16,19]

If ϕ involves paging or segmentation, then the real machine itself probably required an associator for performance reasons.[26] The HV associator will replace it. If the f-map is simple, e.g., $f=R\text{-}B$, then the HV associator will be very similar; if f includes paging it will be somewhat different. The choice of whether to include the VMID or level as part of the search key of the associator can be made for price-performance reasons.

Interpretation of the HV

As indicated earlier, the Hardware Virtualizer can serve as the central mechanism in the design of a new computer system or as an expansion to an existing computer system. In the latter case, we assume a computer system M with a given ϕ-map. The HV construction, i.e., additional data structures, new instruction (LVMID), VM-fault etc., defines a new machine M' with added functionality. The Hardware Virtualizer guarantees that M' is a recursive virtual machine capable of supporting a hierarchy of M' machines with M machines as terminal nodes where desired.

EXAMPLE OF A HARDWARE VIRTUALIZER

In order to clarify the operation of the Hardware Virtualizer, we demonstrate one example of its use. In the example, we present some features of a typical third generation architecture, indicate the extensions introduced by the Hardware Virtualizer, and then illustrate the execution of some instructions. Many other examples have been developed in greater detail, including those for very complex (fourth generation) architectures[19] but the principles involved are the same.

Existing architecture

This example is developed around a canonical third generation computer system, similar to the Honeywell 6000, DEC PDP-10, or IBM System/360. The salient features of the architecture are (1) the privileged/non-privileged mode distinction (master/slave, supervisor/problem, etc.) as part of the instruction counter (IC), (2) a single relocation-bounds register ($R\text{-}B$) whose *absolute* contents may be loaded in privileged mode, and (3) some fixed locations in main memory where the old and new $R\text{-}B$ and IC registers are swapped on a process exception.

To simplify the example we will assume the $R\text{-}B$ register is active, even in privileged mode. Furthermore, all instructions will be assumed to be executing in privileged mode. Since mode violations are local process exceptions and are treated identically to $R\text{-}B$ violations, there is no need to

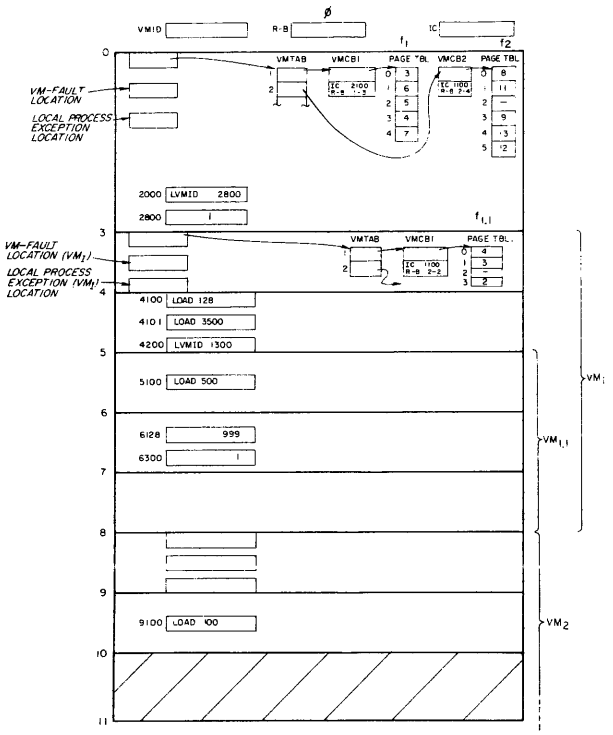


Figure 6—Hardware virtualizer example: Main memory

illustrate them both. The example illustrates execution of central processor instructions only. The extension of the example to include a homogeneous treatment of I/O is possible [17,19] but introduces additional issues of both mechanisms and policies that are best treated in a subsequent paper. Thus, in this example, the *R-B* map is the ϕ-map.

Extensions to architecture

The Hardware Virtualizer requires extensions to the third generation architecture. We will illustrate the modifications introduced by the addition of a page *f*-map (in the memory domain). We will assume 1000-word pages. (See Figure 6.) The modifications include:

(1) database to store *f*—Some fixed known location, say 0, in the memory of level n points to the virtual machine table (VMTAB) which describes the virtual machines of level $n+1$. In this example, each virtual machine control block (VMCB) illustrates a memory map (page table) and a processor map. The processor map includes storage for level $n+1$'s IC and *R-B*. Also included but not illustrated is the level $n+2$ NEXT_SYLLABLE which is stored whenever level $n+1$ issues an LVMID instruction.

(2) a mechanism to invoke *f*—A multi-syllable VMID register and a LVMID instruction are added. When a virtual machine is activated, its IC and *R-B* are loaded from its control block (VMCB).

(3) a composer—A hardware-firmware composer supported with scratchpad memory and associator (for performance reasons) is added. We do not discuss the details of the implementation.

(4) the action on a *VM*-fault—The IC and *R-B* are stored in their VMCB, the appropriate syllable(s) are removed from the VMID and control passes to a fixed known location, say 1, in the *VMM*.

Note that this example illustrates a Type I *f*-map in which resources of level $n+1$ are mapped into resources of level n. Thus, the relocation-bounds register value of level n does not enter into the mapping. In this example when LVMID is executed, relocation is coincidentally zero, but need not be.

The example

Figure 6 shows the state of main memory in our hypothetical hardware virtualized machine. We show VMCB's together with a number of instructions and data. For purposes of illustration, we assume the existence of a simple instruction, LOAD, that accesses memory. Figure 6 also shows the three registers, VMID, *R-B*, and IC, but their values are not indicated. Instead, Table II shows six sets of values for VMID, *R-B*, and IC. For each set, we identify the instruction which is executed and the evaluation sequence used in developing an absolute physical memory address. The table entry includes indication of a process exception, *VM*-fault, and any change to the VMID. The *R-B* register values are represented as *r-b* where *r* is the relocation (in thousands of words) and *b* is the amount of contiguous allocation (in thousands of words).

The six lines of Table II divide into three sets, Lines 1-3, 4, and 5-6. Within these sets, Lines 1-3 execute consecutively and Lines 5-6 also execute consecutively.

TABLE II - HARDWARE VIRTUALIZER EXAMPLE: EVALUATION SEQUENCES

LINE	VMID	R-B	IC	EVALUATION SEQUENCE	VMID AFTER EVAL. SEQ.	POINT ILLUSTRATED
1	NULL	0-14	2000	IC is 2000 ϕ(2000) = 2000 Fetch inst: LVMID 2800 ϕ(2800) = 2800 Append 1 to VMID	1	LVMID instruction
2	1	1-3	2100	IC is 2100 ϕ(2100) = 3100 f_1(3100) = 4100 Fetch inst: LOAD 128 ϕ(128) = 1128 $f_1$1128 = 6128 Load 999	1	Virtual Machine instruction execution
3	1	1-3	2101	IC is 2101 ϕ(2101) = 3101 f_1(3101) = 4101 Fetch inst: LOAD 3500 ϕ(3500) = e	1	Process exception in virtual machine
4	2	2-4	1100	IC is 1100 ϕ(1100) = 3100 f_2(3100) = 9100 Fetch inst: LOAD 100 ϕ(100) = 2100 f_2(2100) = t	NULL	Virtual Machine fault and different VMID
5	1	0-5	3200	IC is 3200 ϕ(3200) = 3200 f_1(3200) = 4200 Fetch inst: LVMID 1300 ϕ(1300) = 1300 f_1(1300) = 6300 Append 1 to VMID	1.1	LVMID with recursion
6	1.1	2-2	1100	IC is 1100 ϕ(1100) = 3100 $f_{1.1}$(3100) = 2100 f_1(2100) = 5100 Fetch inst: LOAD 500 ϕ(500) = 2500 $f_{1.1}$(2500) = t	1	Recursive instruction execution and VM-fault

Referring to Figure 6 and Table II, let us step through the first several evaluation sequences. In Line 1, we are in the *VMM* running on the real machine. All control blocks have been set up and it is time to activate virtual machine 1. The instruction counter value is 2000. Since the *R-B* map is 0-14, we add zero to 2000 and obtain $\phi(2000) = 2000$. The VMID is NULL. Therefore, the resource name 2000 is a real resource and we fetch the instruction at physical location 2000, LVMID 2800. We apply the *R-B* map to 2800 and eventually fetch 1 which is loaded into the VMID register.

Virtual machine 1 is now activated and its IC and *R-B* registers are loaded from VMCB1. Thus, IC is now 2100 and *R-B* is 1-3. Even though the memory of virtual machine 1 is 5000 words (as can be seen from its page table) the *R-B* register limits this active process to addressing only 3000 words. This limit was presumably set by the operating system of virtual machine 1 because the active process is a standard (non-monitor) user.

Now we are in Line 2 and the IC is 2100. To apply the ϕ-map, we add 1000, checking that 2100 is less than 3000, and obtain $\phi(2100) = 3100$. Since the VMID is 1, we must apply f_1 to map the virtual resource 3100 to its real equivalent. The page table, pointed at by VMCB1, indicates that virtual page 3 is at location 4000. Therefore, $f_1(3100) = 4100$ and the LOAD 128 instruction is fetched.

The other sequences may be evaluated in the same manner. Line 3 illustrates a process exception to the local exception handler of VM1, Line 5 illustrates activation of recursion, and Lines 4 and 6 illustrate *VM*-faults to the fault handler of their respective *VMM*s.

It should be noted that we have added a paged f-map which is invisible to software at level n. The pre-existing *R-B* ϕ-map remains visible at level n. Thus, operating systems which are aware of the *R-B* map but unaware of the page map may be run on the virtual machine without any alterations.

Note that the addition of an *R-B* f-map instead of the paged f-map is possible. This new *R-B* f-map would be distinct from and an addition to the existing *R-B* ϕ-map; it would also have to satisfy the recursion properties of f-maps.[19] Similarly, a paged f-map added to a machine such as the IBM 360/67 would be distinct from the existing paged ϕ-map.

CONCLUSION

In this paper we have developed a model which represents the addressing of resources by processes executing on a virtual machine. The model distinguishes two maps: (1) the ϕ-map which maps process names into resource names, and (2) the f-map which maps virtual resource names into real resource names. The ϕ-map is an intra level map, visible to (at least) the privileged software of a given virtual machine and expressing a relationship within a single level. The f-map is an inter-level map, invisible to all software of the virtual machine and establishing a relationship between the resources of two adjacent levels of virtual machines. Thus, running a process on a virtual machine consists of running it under the composed map $f \circ \phi$.

Application of the model provides a description and interpretation of previous virtual machine designs. However, the most important result is the Hardware Virtualizer which emerges as the natural implementation of the virtual machine model. The Hardware Virtualizer design handles all process exceptions directly within the executing virtual machine without software intervention. All resource faults (*VM*-faults) generated by a virtual machine are directed to the appropriate virtual machine monitor without the knowledge of processes on the virtual machine (regardless of the level of recursion).

A number of virtual machine problems, both theoretical and practical must still be solved. However, the virtual machine model and the Hardware Virtualizer should provide a firm foundation for subsequent work in the field.

ACKNOWLEDGMENTS

The author would like to thank his colleagues at both MIT and Harvard for the numerous discussions about virtual machines over the years. Special thanks is due to Dr. U. O. Gagliardi who supervised the author's Ph.D. research. In particular, it was Dr. Gagliardi who first suggested the notion of a nested virtual machine fault structure and associated virtual machine identifier (VMID) register functionality.

REFERENCES

1. Buzen, J. P., Gagliardi, U. O., "The Evolution of Virtual Machine Architecture," *Proceedings AFIPS National Computer Conference*, 1973.
2. Berthaud, M., Jacolin, M., Potin, P., Savary, H., "Coupling Virtual Machines and System Construction," *Proceedings ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
3. Meyer, R. A., Seawright, L. H., "A Virtual Machine Time-Sharing System," *IBM Systems Journal*, Vol. 9, No. 3, 1970.
4. Parmelee, R. P., "Virtual Machines—Some Unexpected Applications," *Proceedings IEEE International Computer Society Conference*, Boston, Massachusetts, 1971.
5. Winett, J. M., "Virtual Machines for Developing Systems Software," *Proceedings IEEE International Computer Society Conference*, Boston, Massachusetts, 1971.
6. Casarosa, V., Paoli, C., "VHM—A Virtual Hardware Monitor," *Proceedings ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
7. Keefe, D. D., "Hierarchical Control Programs for Systems Evaluation," *IBM Systems Journal*, Vol. 7, No. 2, 1968.
8. Buzen, J. O., Chen, P. P., Goldberg, R. P., "Virtual Machine Techniques for Improving Software Reliability," *Proceedings IEEE Symposium on Computer Software Reliability*, New York, 1973.
9. Attanasio, C. R., "Virtual Machines and Data Security," *Proceedings ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
10. Madnick, S. E., Donovan, J. J., "Virtual Machine Approach to Information System Security and Isolation," *Proceedings ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
11. Adair, R., Bayles, R. U., Comeau, L. W., Creasy, R. J., "A Virtual Machine System for the 360/40," *IBM Cambridge Scientific Center Report* No. G320-2007, 1966.

12. Srodawa, R. J., Bates, L. A., "An Efficient Virtual Machine Implementation," *Proceedings AFIPS National Computer Conference*, 1973.
13. Fuchi, K., Tanaka, H., Namago, Y., Yuba, T., "A Program Simulator by Partial Interpretation," *Proceedings ACM SIGOPS Second Symposium on Operating Systems Principles*, Princeton, New Jersey, 1969.
14. *IBM Virtual Machine Facility/370—Planning Guide*, IBM Corporation, Publication Number GC20-1801-0, 1972.
15. Goldberg, R. P., "Hardware Requirements for Virtual Machine Systems," *Proceedings Hawaii International Conference on System Sciences*, Honolulu, Hawaii, 1971.
16. Lauer, H. C., Snow, C. R., "Is Supervisor-State Necessary?," *Proceedings ACM AICA International Computing Symposium*, Venice, Italy, 1972.
17. Lauer, H. C., Wyeth, D., "A Recursive Virtual Machine Architecture," *Proceedings ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
18. Gagliardi, U. O., Goldberg, R. P., "Virtualizable Architectures," *Proceedings ACM AICA International Computing Symposium*, Venice, Italy, 1972.
19. Goldberg, R. P., *Architectural Principles for Virtual Computer Systems*, Ph.D. Thesis, Division of Engineering and Applied Physics, Harvard University, Cambridge, Massachusetts, 1972.
20. Goldberg, R. P., *Virtual Machine Systems*, MIT Lincoln Laboratory Report No. MS-2687, (also 28L-0036), Lexington, Massachusetts, 1969.
21. Schroeder, M. D., Saltzer, J. H., "A Hardware Architecture for Implementing Protection Rings," *Communications of the ACM*, Vol. 15, No. 3, 1972.
22. *The Fourth Generation*, Infotech, Maidenhead, England, 1972.
23. Liskov, B. H., "The Design of the VENUS Operating System," *Communications of the ACM*, Vol. 15, No. 3, 1972.
24. Goldberg, R. P., "Virtual Machines—Semantics and Examples," *Proceedings IEEE International Computer Society Conference*, Boston, Massachusetts, 1971.
25. Madnick, S. E., *Storage Hierarchy Systems*, Ph.D. Thesis, Department of Electrical Engineering, MIT, Cambridge, Massachusetts, 1972.
26. Schroeder, M. D., "Performance of the GE-645 Associative Memory while Multics is in Operation," *Proceedings ACM SIGOPS Workshop on System Performance Evaluation*, Cambridge, Massachusetts, 1971.

The computer-aided design environment project (COMRADE)

by THOMAS R. RHODES*

Naval Ship Research and Development Center
Bethesda, Maryland

BACKGROUND

Since 1965, the Naval Ship Engineering Center (NAVSEC) and the Naval Ship Research and Development Center (NSRDC), sponsored by the Naval Ship Systems Command, have been actively involved in developing and using computer facilities for the design and construction of naval ships. The overall goals of this effort, known as the Computer-Aided Ship Design and Construction (CASDAC) project, have been twofold—first, to achieve significant near term improvements in the performance of ship design and construction tasks, and second, to develop a long term integrated CASDAC system for all phases of the ship design and construction process.[1]

While pursuit of the first goal has achieved notable cost savings,[1] it has also produced a situation tending to delay the attainment of the second goal, that of an integrated CASDAC system. There soon were many individual batch-oriented computer programs, designed and operated independently of each other, involving relative simplicity, low cost, and short-term benefits, all of which contrasted against the complications, high cost, and projected benefits of an interactive integrated-application system. But yet, it was considered that a quantum improvement in the time and cost of performing ship design could only be realized through a coordinated and integrated approach. The real question facing the Navy was whether such an approach was technically and economically feasible.

In an attempt to demonstrate the feasibility of an integrated approach, members of the Computer-Aided Design Division (CADD) and the Computer Sciences Division (CSD) of NSRDC joined with members of the CASDAC office of NAVSEC to investigate and develop a prototype system.

The phase of ship design known as concept design was chosen to be modeled as an integrated system and to provide the macrocosm for studying system requirements. Some of the characteristics favoring choice of this phase, were:

- that as an initial phase of ship design, wherein basic features such as ship type and size, weapons and electronics systems, propulsion machinery and major shipboard arrangements were determined, it represented a critical phase where optimization over many alternatives could result in improved ship performance and lower development costs;
- that as an activity with a high level of creativity and analysis, in which operational requirements were transformed into a feasible engineering reality, it could be enhanced through application of computer aids;
- that as an activity with extensive interaction and information exchange among a multiplicity of engineers representing different disciplines (e.g., naval architecture, marine, mechanical, electrical, etc., engineering), it produced a dynamic atmosphere that was considered a "natural" for an integrated solution; and,
- that with relatively few engineering tasks and data requirements compared to later ship development phases, it offered a tractable situation for analysis and application of existing computer technology.

SYSTEM REQUIREMENTS

The initial study effort led the engineers and application programmers of CADD and the systems programmers and analysts of CSD along different, but complementary paths in viewing the system requirements—one view reflecting the engineering requirements of concept design and the other, the imposed computer requirements.

The engineering analysis sought to identify the various tasks, their contribution to the overall design process, their relationships, dependencies, data input and output requirements, and the role of the engineer throughout the design process. Each task was further divided to reveal the major steps and to explore the computer implementa-

* The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Navy.

tion of them. While a "building-block" approach to problem solving, involving a strong interaction between the engineer and the system, was desired, questions were raised as to how much flexibility the system should provide. Should the designer work at a low level with "atomic" engineering functions to incrementally describe and solve his problem, or should the designer work within a pre-established set of alternatives, where major decision points have been defined, and he should have only to choose and sequence among a variety of algorithmic procedures in which much of the problem structure has been imbedded within the program logic? While the "atomic" approach appeared more flexible and was conceivably more adaptable to new design situations, the latter approach was favored for the initial effort since, under this approach it was deemed that satisfactory design results could still be obtained for a broad set of problems, many existing application programs were amenable for use, and less sophistication was required to develop and use the system.

From the analysis of the overall design process, a good indication of program and designer data requirements was obtained. The required data was organized to reflect various logical associations, producing a large and complex data structure.[2] To minimize data redundancy a distinction was made between data describing the characteristics of a particular ship and data that was common to many ships. This resulted in separate ship and catalog files and in having data associations both within and between files. This separation of data was favored also because the catalog files would be less subject to change during the design process than the relatively volatile ship file, and less queuing would be required during processing.

The data base was considered to be the crucial link through which information would be shared among designers and programs, and hence it represented the key element in an integrated system. The demand for information required that data management capabilities be made available to both the application program during execution, and to the designer working directly with the data base at the terminal. The large and complex data structure indicated that efficient and flexible techniques would be necessary to structure, store, access, and manipulate this data, and finally, some means of controlling access to the files would be required to preserve data integrity.

In addition to the analyses of the design process engineering requirements, consideration was also given to coordinating or controlling the overall design process. Although the designer would be responsible for performing design tasks, he would do so under the general direction of the project leader or design administrator. Task assignments and final acceptance of design results would normally be under the purview of this member of the design team, which implied that the system would need to be responsive to the administrator's role by providing controls over program and file access, and reports on the design status and system usage.

While the engineering analysis was directed toward identifying the elements and requirements of an integrated ship-concept design system, the computer science effort was directed toward providing general mechanisms that were adaptable to ship design and to similar situations where it was necessary to coordinate and integrate many users, programs, and data files.

The effort to produce a prototype ship-concept design system was termed the Integrated Ship Design System (ISDS) project,* while the effort to develop a framework of capabilities for constructing integrated systems was termed the Computer-Aided Design Environment (COMRADE) project.

SYSTEM DESCRIPTION

From the analysis done during 1970, design and development of a prototype system was scheduled to begin the following year using NSRDC's CDC-6700 computer with the SCOPE 3.3 Operating System.

The NSRDC computer configuration, shown in Figure 1, provides remote access from interactive graphic, conversational keyboard, and batch stations to high-speed dual processors with extensive secondary storage. Conversational teletype (TTY) and medium speed batch (200 UT) capabilities are provided through the CDC INTERCOM Time-Sharing software, while high-speed remote batch and interactive graphic communications are serviced by the CDC EXPORT-IMPORT software. This configuration appeared satisfactory for a prototype effort; however, the relatively small main memory resource and the separate job schedulers for interactive graphics and conversational keyboards were considered major problems for program development and integrated operations. To minimize difficulties in a first level effort, exclusive attention was given to use of the more available conversational keyboard (TTY) as the principal designer interface to the system rather than to the more desirable interactive graphic terminal. However, some graphic applications were planned and these would interface with the data base for test purposes.

From a consideration of the ISDS requirements and an examination of related efforts, such as the Integrated Civil Engineering System (ICES)[3], COMRADE proceeded to design and develop three major software components: an Executive system; a Data Management system; and a Design Administration system. References 4, 5, 6, and 7 describe these components in greater detail, however, the following summary gives an indication of their functions and capabilities:

- Executive System—An interactive supervisor program, functioning under the INTERCOM Time-Sharing system, that interprets and processes command procedures. Through supporting software,

* "The Integrated Ship Design System, Model 1—System Development Plan," internal document of Computation and Mathematics Department, NSRDC, February 1971.

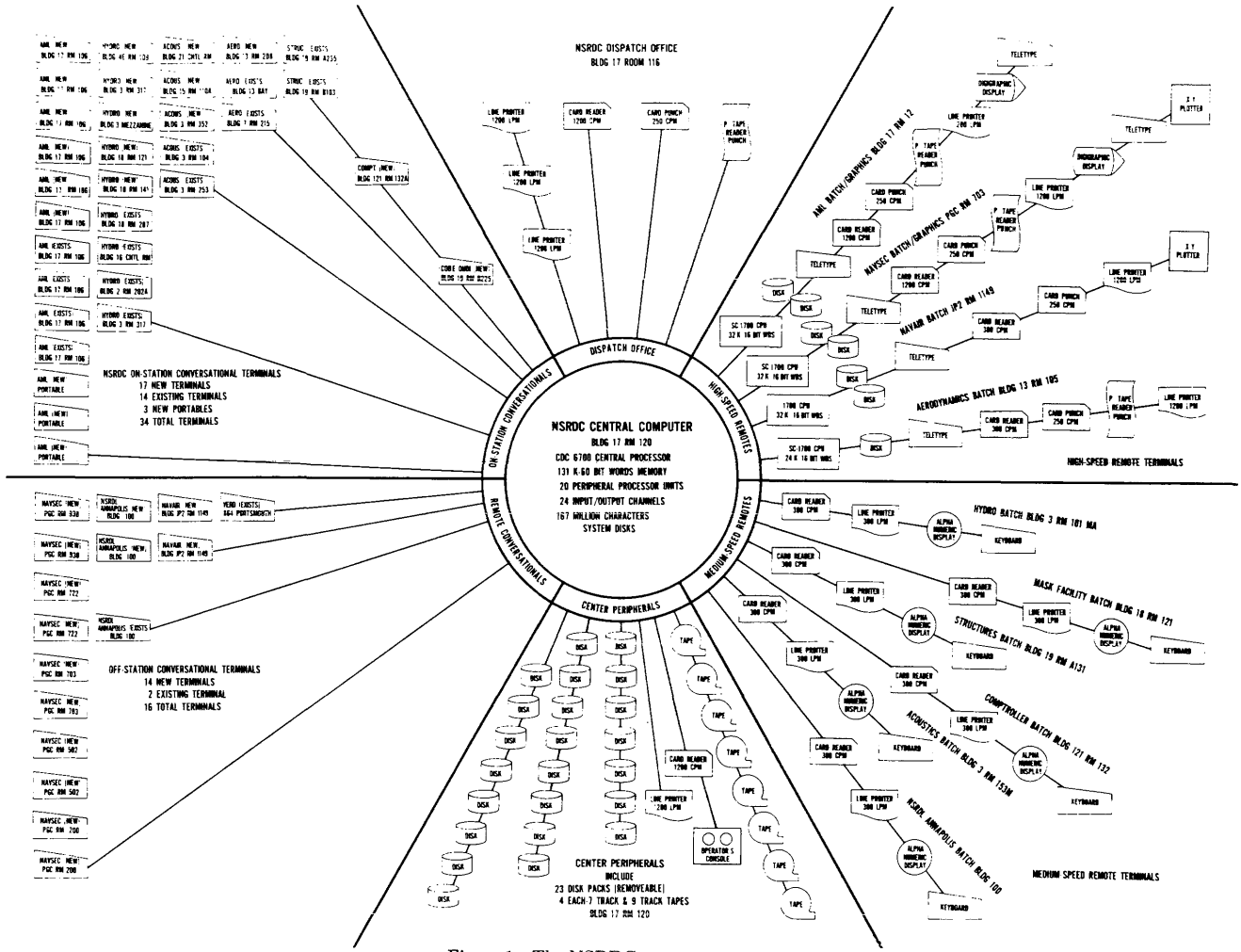


Figure 1—The NSRDC computer system

known as the Procedure Definition Language (PDL), command procedures are defined as the complex sequence of computer operations necessary to perform a corresponding design task. Operations that can be automatically performed include: printing tutorials or data at the terminal; reading data from the terminal and passing it to programs through a System Common Communication area, and vice versa; setting default data values in system common; attaching, unloading, and purging files; initiating programs for time-shared or batch execution; executing most SCOPE control statements; and, altering the sequence of operations through conditional or unconditional transfers. Capabilities are also provided to control command usage through command-locks and user-keys, and to define unique "subsystems" of related commands.

Through the Executive capabilities, command procedures representing design tasks can be defined in such a way as to present a problem-oriented interface to the user and to hide distracting computer operations. Since computer actions can be dynamically altered during command processing, considerable flexibility for user decision-making can be provided. Finally, during execution of an application program step, the Executive is removed

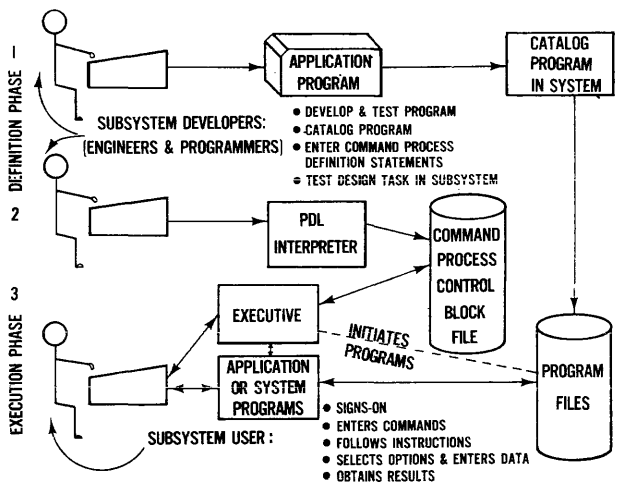


Figure 2—COMRADE command definition and execution phases

from main memory, permitting larger residency by the application module. Upon termination of the module execution, control is returned to the Executive. (In Figure 2, the command definition and execution phases are figuratively shown as steps 2 and 3.)

- Data Management System—A library of FORTRAN-callable subroutines and user-oriented command procedures that provide data management capabilities to both the programmer and terminal user. Users may store, update, and retrieve data by name, retrieve data via queries on data attributes, cross-link data in different files through pointers, and in general define and process large, complex file and data structures.

The COMRADE Data Management System (CDMS) is hierarchically structured into three levels:

(1) The foundation or interfaces with the SCOPE I/O operations consists of the direct access technique and directory processing programs. Variable length logical records can be accessed by name, where each name is "hashed" to form an index into a directory that can reference up to 516,000 data records. Previously used disk space is reallocated and a paged, circular-buffer is used to store and process data records.

This set of programs, called the COMRADE Data Storage Facility (CDSF), can be used by a programmer to store and retrieve data records or blocks by name; however, at this level, he would be required to do his own internal record structuring and processing.

(2) Built on this foundation are system procedures, called the Block-Type Manipulation Facility (BTMF), that enable the data record contents to be defined, stored, retrieved, and updated by name, thus enabling the programmer to logically define and process records without regard to the internal structure. At this level, the format of each unique block-type is defined before the data file is generated, and then, subsequently it is used to build and process corresponding data blocks. Each block-type can be logically defined as subblocks of named data elements. Each element can be of real, integer, character, or pointer data type, and can be single- or multi-valued (i.e., array). Single-valued elements can be "inverted" and used as keys for a query-language retrieval, and pointer-elements are used to form relationships among data records within one or several files. Sets of elements can be grouped together under a group name, with the group repeated as needed (i.e., repeating groups).

Using the BTMF programs, the user can then process data logically by name while the system identifies and resolves the internal record structure.

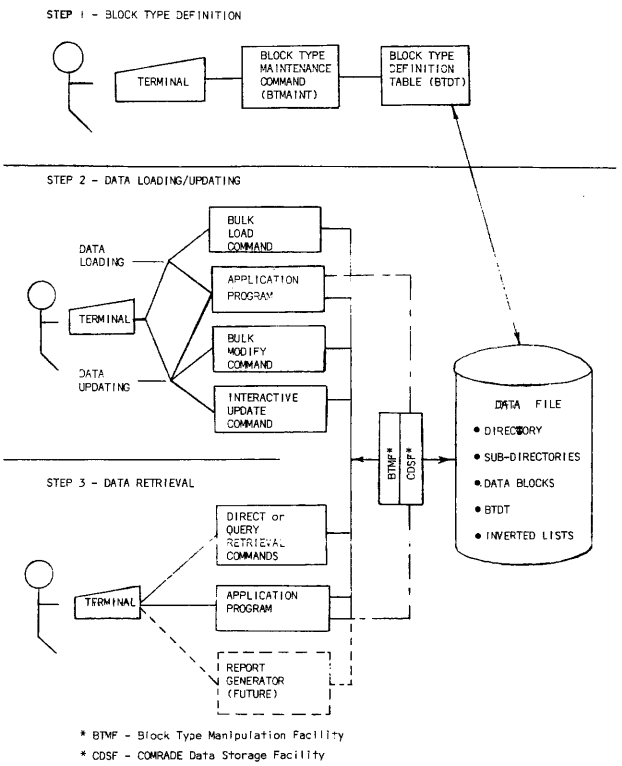


Figure 3—File definition and processing using COMRADE data management system

(3) While the second level capabilities were provided as subroutines for programmer use, the third level consists of user-oriented command procedures for terminal use. Utilizing the BTMF routines, these programs enable terminal users to define data records; to load and update files; to retrieve data by name or through a query language; and to obtain information on file characteristics, such as size, record names, block-types and inverted attribute names and ranges.

In Figure 3, the various CDMS components for file definition and processing are shown.

- Design Administration System—A set of command procedures and program capabilities to assist the design project leader or administrator to:
  - identify valid subsystem users and assign appropriate command and file access keys;
  - identify subsystem files and assign appropriate file locks and passwords;
  - selectively monitor and receive reports on subsystem activities, such as, names of subsystem users, dates and times, commands used, significant events, estimated cost of processing, etc.

Additional functions are provided to allow programs to dynamically attach and unload files during execution, and to prepare and cleanup necessary files during subsystem sign-on and sign-off.

STATUS

In the Spring of 1972, testing of the described COMRADE software began, using a selected set of ship design programs and data necessary to verify operational capabilities and to demonstrate the formation of an ISDS. Various interactive design commands were implemented, together with ship and catalog data files of limited size, for evaluation purposes. Figure 4 illustrates the functional components involved in the system development effort. During testing, engineers and ship designers who saw and used these capabilities and who were not involved with implementation, generally approved the system interface and the capabilities provided. Correspondingly, subsystem developers found the COMRADE capabilities to be convenient and necessary, but not always sufficient, thus providing feedback for corrections and further development.

While the current ISDS effort is directed toward constructing a set of design commands and data files sufficient to engage in actual ship concept design studies, COMRADE efforts have been concentrated on evaluating performance, "tuning" components for more efficient operation, documenting existing work, and planning major enhancements. For example, work planned for the coming year, includes:

- developing an interface between the Executive and the interactive graphics terminal for a balanced system environment;
- developing a report generator facility to conveniently retrieve and format data file information;
- developing a PERT-like facility to conveniently define, schedule, and monitor a subsystems activity; and,
- considering application of computer-networks for a computer-aided design environment.

While enhancements and improvements are planned, application of COMRADE capabilities to other areas of ship design, engineering, logistics, etc., will also be investigated.[8] For example, planning is under way to develop a Computer-Aided Piping Design and Construction (CAPDAC) system which will integrate shipyard planning, design, and fabrication activities related to piping systems.*

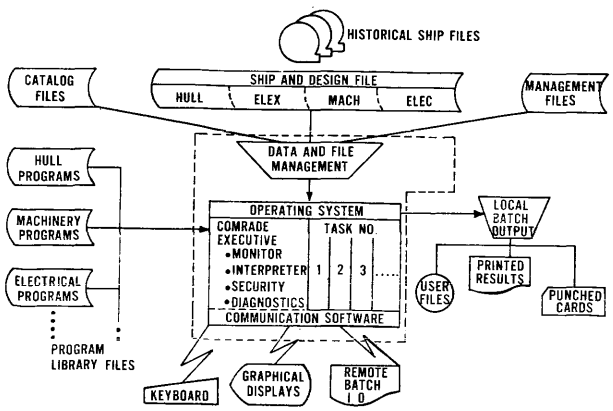


Figure 4—Integrated system components

SUMMARY

In response to the Navy requirements for an integrated ship design system the Computer-Aided Design Environment project has developed an initial set of general software capabilities, not merely limited to ship design, that provide a framework for assembling and coordinating programs, data, and their users into an integrated subsystem. The three major COMRADE components are: the Executive System, an interactive program operating under the INTERCOM time-sharing system of the CDC-6700 computer at NSRDC, which can process a complex and varying sequence of computer operations in response to user-defined commands; the Data Management System, a direct access capability to process large complex file and data structures via subroutines or terminal commands; and, the Design Administration System, a set of subroutines and terminal commands used to control subsystem and file access, and to optionally monitor and report on selected information, such as user-names, date and time, commands used, cost estimates, etc., during subsystem operations.

These capabilities have been applied to several prototype application systems, most notably the Integrated Ship Design System, and several other application systems are being planned.

While the COMRADE mechanisms have been shown to work, they are merely "tools" in constructing integrated systems and therefore depend on careful system planning and judicious use by subsystem developers to achieve an effective man-machine system. Many other factors, such as the performance and capabilities of the computer system, the application of software engineering techniques to modular program construction, the organization of data files and data communication regions for programs and users, and the efficiency of program elements are all particularly significant in determining the appearance and performance of an integrated system.

The COMRADE capabilities, in conjunction with the ISDS project, have demonstrated the technical feasibility of constructing a convenient and effective computer tool that will provide guidelines and direction for continuing and similar efforts toward achieving a completely integrated CASDAC system.

* Sheridan, H., "Formulation of a Computerized Piping System for Naval Ships," internal technical note, Computation and Mathematics Department, NSRDC, June 1971.

REFERENCES

1. *Naval Ship Systems Command Technical Development Plan—Computer-Aided Ship Design and Construction*, Naval Ship Systems Command, Washington, D.C., February 1970.
2. Thomson, B., "Plex Data Structure for Integrated Ship Design," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.
3. Roos, D., *ICES System Design*, second edition, M.I.T. Press, 1967
4. Tinker, R., Avrunin, I., "The COMRADE Executive System," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.
5. Willner, S., Bandurski, A., Gorham, W., and Wallace, M., "COMRADE Data Management System," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.
6. Bandurski, A., Wallace, M., "COMRADE Data Management System—Storage and Retrieval Techniques," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.
7. Chernick, M., "COMRADE Design Administration System," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.
8. Brainin, J., "Use of COMRADE in Engineering Design," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.

Use of COMRADE in engineering design

by JACK BRAININ*

Naval Ship Research and Development Center
Bethesda, Maryland

INTRODUCTION

The Naval Ship Research and Development Center began formal work in computer aided design in 1965. The initial tasks undertaken were the development of individual batch application programs which were little more than the computerization of manual design methods. The programs developed included those shown in Figure 1.

These programs were used by those people acquainted with them, and many programs existed in various agencies in a number of versions. New users had to determine the version most suitable for their purpose by individual contact and then had to prepare the necessary data inputs, often in a rather laborious manner. Occasionally, users linked a number of batch programs together to form a suite of programs. Such suites could deal with modest segments of a technical problem. Existing or independently developed suites would deal with other segments of the problem. The interfaces between suites was very difficult, unwieldy, and sometimes impossible. The resulting system was inflexible, running time tended to be excessive and no user-directed interaction was possible. Generally, computer-aided design lacked what might be called an overall strategy.

OVERVIEW OF INTEGRATED SYSTEMS

Integrated systems provide computer-aided design with an overall strategy and greatly reduce the time required to execute a design. Such systems permit data transfer between various functional design disciplines and provide the engineer with the means to use the computer as a productive tool without the requirement that he also become a programmer.

The integrated systems approach facilitates the work of designers and managers by:

a. permitting communication between users via a preformatted design file.
b. providing tools for the project leader to maintain control over the programs to be used, the personnel to be employed and by permitting him to set up target dates for tasks and generate performance reports via the computer.
c. permitting the exhange of data between programs via computer files which are automatically created for the user in any integrated suite of programs.
d. permitting tasks to be initiated by engineers using simple language statements at computer terminals.
e. providing program instructions and program descriptions to engineers directly from the teletype terminal.
f. permitting engineers to exercise their disciplines without having to acquire programmer expertise.
g. demanding programs to be standardized for inclusion in the system and hence inducing homogeneity into routine processes.
h. aiding in the building of compatible suites of programs.

- Feasibility studies for destroyers, submarines and auxiliary ships
- Structures - Ship midship section design
  - Ship hull lines fairing and plate development
- Ship propulsion system design - Propeller, shafting, reduction gear and vibration and dynamic shock analysis
- Steam power plant - Heat balance
  - Condenser design
- Design of piping systems for compressible and incompressible flow
- Electrical - Power distribution analysis
  - Cable sizing
- Electronics - Antenna matching networks
- Data retrieval - Machinery, electrical/electronic component catalogs and libraries
- Interactive graphics - Machinery arrangements
  - Detection of electromagnetic hazards
  - Electronic system block diagrams
  - Compartment arrangement & floodable lengths
- Construction - Scheduling operations
  - Numerical control applications
- Space modeling

Figure 1—CAD Computer programs have been developed for the above

HISTORY OF INTEGRATED SYSTEMS/COMRADE

A number of integrated systems have been envisioned over the last five years. In the Navy, these systems included an Integrated Ship Design System (ISDS) for the preliminary design of naval ships and a Ship Inte-

* The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Navy.

grated System (SIS) for detail ship design and ship construction. Associated with the SIS are a number of functional subsystems dealing with:

electrical/electronics
hull structure
ship arrangements
piping
heating, ventilation and air conditioning
stores and replenishment
document generation and control

The Integrated Ship Design System for preliminary design has been under development at the Naval Ship Research and Development Center since 1969 and it provided a focal point for COMRADE[1] in the real world. The development of the Ship Integrated System is part of the engineering development plan under the Computer-Aided Design and Construction Project within the Department of the Navy.

In addition to the use of COMRADE for the Integrated Systems noted in the foregoing, the COMRADE ideas have provoked discussion and investigation for use by:

a. The National Aeronautics and Space Administration's Integrated Preliminary Aerospace Design System.[2] The feasibility of this system is currently being investigated by two aerospace contractors.
b. The Navy's Hydrofoil Program Office. This office has determined to use COMRADE for its Hydrofoil Design and Analysis System[3] now being initiated.

COMRADE is capable of application to the design and construction of more or less any complex manufactured product.

COMPONENTS OF AN INTEGRATED SYSTEM

The development of an integrated system may be broken down into three major areas of development: System Software Development; File Development; and Application Program Development.

The System Software Development effort, in the case of COMRADE, has been subdivided into an executive system,[4] a data management system,[5] and a design administration system.[6] The executive, among other things, provides an interface between the engineering user at a remote terminal and the computer system. The data management system supports the transfer of data between users, programs and files, and provides a common data handling capability. The design administration system logs system usage, generates usage reports and specifies and controls access to the engineering system, its commands and files.

The file development effort may be thought of as being broken into a catalog file and a design file. The catalog would contain standard engineering data which may be used in the construction of a number of products. In contrast, a design file will contain data pertaining specifically to the particular product being designed.

While the System Software Development effort may be applied to a number of product lines the application program development is product dependent. For example, an application program to fair the lines on a ship would not be the same program that would fair the lines on an aerospace vehicle. Similarly, a ship design file would differ from an aerospace design file.

APPLICATION OF COMRADE

Figure 2 cites the application of the COMRADE System Development effort to the case of a building design. For greater generality a building was chosen, rather than a ship, to further illustrate the potential application of the system. In this instance, the design file is a BUILDING design file and contains building design data. An engineering user sitting at a teletype enters a simple English language command (e.g., BUILDING) to identify that he would like to design a building. A BUILDING subsystem of COMRADE is automatically initiated if the COMRADE design administration system recognizes the terminal user as a valid user of the subsystem BUILDING. The procedure definition language of COMRADE then

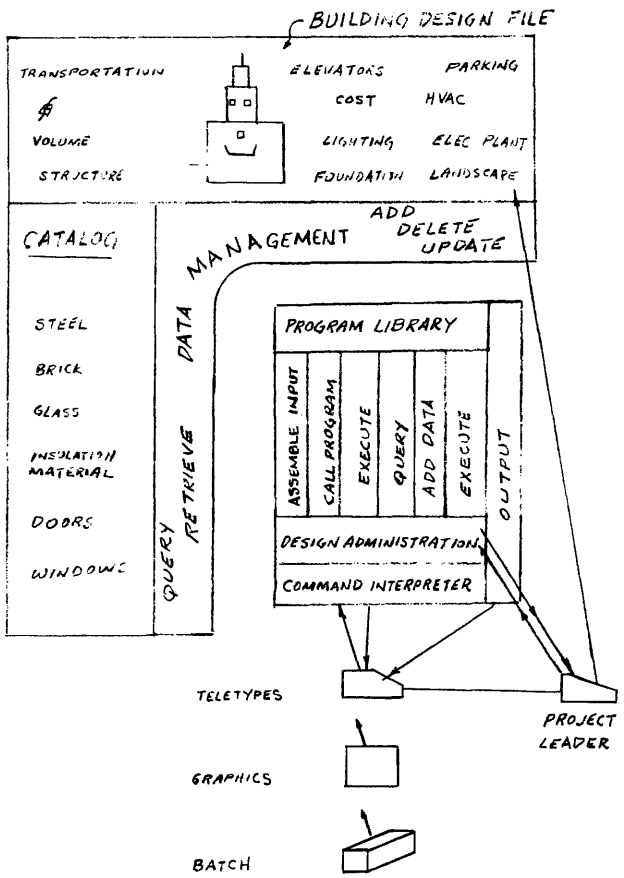


Figure 2—Integrated building design

issues a tutorial to the terminal user which queries: "WHAT BUILDING?" The engineer responds to the question, with an English language statement specifying the building type and name (e.g., skyscraper 123, private dwelling 102, or bank 149). The design administration system compares a list of valid users of each building design with the name of the terminal users. As an illustration, assume the user enters the command BANK1. If the user is permitted to work on BANK1, there will be a match and the user will be permitted to proceed into the selection of a design procedure. A user may have access privileges to BANK1 but not to BANK2 or BANK3. If a user enters BANK2 and has not been authorized to operate on BANK2, a diagnostic message will be returned to the user informing him, "YOU ARE NOT AUTHORIZED TO USE BANK2." This provides another level of access control which prevents people who may be working on the system, but are not working on this particular bank, from gaining access to this bank's files. Upon approval of the design administration system the user is accepted as a valid user of the BUILDING subsystem and the BANK1 design. However, this still does not permit the user to arbitrarily use any command within the system (all users of the BANK subsystem will not be able to access the details of the alarm system) nor does it permit him to gain access to all of the files within the system as each command and file has associated with it a command access key or a file access key which must be matched by the user input.

The tasks required to design a bank are stored in a library of application programs (structural details, security systems, power and lighting systems, ventilation, furniture and vault arrangements, etc.) The user selects and enters an English language command, from the library, which is the name of the design procedure he would like to execute. For illustrative purposes, he may enter ELEC PWR DIST to indicate that he is designing an electrical power distribution system. Upon approval of the design administration system the user will be accepted by the system as a valid user of the named command procedure (ELEC PWR DIST). The user is offered a choice of prompting messages, either complete or abbreviated. The user's response is dependent on his familiarity with the program. A new user will normally select the complete tutorial option and an experienced user will normally select the abbreviated option. A convention has been established that the user's response is selected from the choices enclosed in parentheses.

The execution of the design procedure is an interactive iterative procedure involving a dialogue between an engineer at the terminal communicating with the tutorial directions which are typed step-by-step by the teletype to the user, who no longer needs to be a computer expert but rather a reasonable engineer with good engineering judgment. These tutorials which are "human-readable" are produced by the Procedure Definition Language of the COMRADE executive. During the terminal session the user is given choices as to the source of the input data. Data may be input from a catalog file (which would contain data such as motors, generators, cables and their corresponding impedances), a design file (data pertaining to the BANK being designed, such as the physical space available to locate a vault), a card deck, an old file, or from the user at the terminal (who would enter data such as the path and length of cables). The user of a given design need not manually input all required data if this data is already within the system as a result of a previously executed design procedure. Figure 3 illustrates a hypothetical user-system interaction for the design procedure ELEC PWR DIS. The terminal user merely responds to the tutorials by providing the underlined values. The user begins by entering the design procedure name (ELEC PWR DIS) and from then on the user and system interact as shown.

The ELEC PWR DIS design procedure consists of four program modules (represented as A, B, C and D in Figure 4) which may be combined in a number of ways to perform an electrical power distribution analysis. Module A calculates the impedance matrix of the input circuit; B performs a short circuit analysis; C performs a load flow analysis; and D performs a transient stability analysis.

```
???? BUILDING
WHAT BUILDING? BANK1
???? ELEC PWR DIS
WOULD YOU LIKE COMPLETE (COMP)
    OR ABBREVIATED (ABBR) TUTORIALS? COMP
(OLD) OR (NEW) CIRCUIT? NEW
DESCRIBE NEW CIRCUIT:
IS DATA TO COME FROM (CARDS), (OLD FILE) OR (TERM)? TERM
IDENTIFY ELEMENT, LEADING NODE, TRAILING NODE AND RETURN CARRIAGE
TYPE DONE TO INDICATE CIRCUIT IS COMPLETE
EL1? GEN1, 0, 1
EL2? GEN2, 0, 2
EL3? - - - - -
EL4? - - - - -
EL5? DONE
IMPEDANCE MATRIX CALCULATION IS COMPLETE
WHICH DESIGN MODULE:
(SC) SHORT CIRCUIT
(LF) LOAD FLOW
(TS) TRANSIENT STABILITY
(END) EXIT FROM ELEC PWR DIS
? SC
ENTER TYPE OF SHORT (3) 3 PHASE OR (1) 1 PHASE, NODE I.D.
? 3, 2
```

Figure 3—Sample terminal session for design procedure ELEC PWR DIST

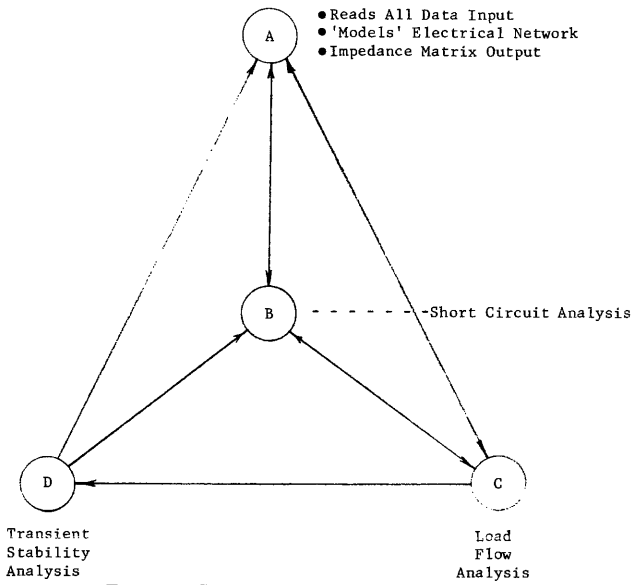


Figure 4—Electrical power distribution analysis

Possible control flow through the modules is indicated by the connecting arrows. For example, after calculating the impedance matrix (A), it is possible to perform a short circuit (B) or a load flow (C) analysis, but not a transient stability analysis (D).

Data is retrieved for use by design procedures from the design file and the catalog files by the subroutine calls of the COMRADE data management system. This is possible only if the files are structured in the data block format readable by the COMRADE data management system.

The data blocks are defined by the data block definition facility of COMRADE and the data is loaded into the data blocks using the COMRADE bulk load capability from a batch terminal. COMRADE also provides an interactive update capability for the terminal user to modify the contents of a data base providing the user has the appropriate passwords.

Design procedures are called from a library of application programs. The output of the applications programs which may serve as input to other design procedures is stored in the design file. However, each user engineer cannot be granted access to this critical file in order to ensure the integrity of the design. Therefore, the engineering user would obtain the results of his design procedure either at a terminal or from a high speed printer and he would analyze the acceptability of the results. If the results are acceptable to the engineer he would inform his project manager, who would then execute an update command to update the design file. In the case of ELEC PWR DIS, the project manager would execute the command UP ELEC PWR DIS which would place the BANK1 data resulting from the ELEC PWR DIS design procedure on the design file, thus this data is made available to other engineers to use as input data to other design procedures such as for generator foundation design and for arrangements. As more and more programs are executed the design file will grow, and when it is complete it will contain a complete digital description of the building being designed.

COMMENTS AND CONCLUSIONS

The development of integrated design systems using COMRADE is gaining substantial acceptance in the Navy. With its introduction, attention is directed to further considerations which will require careful scrutiny.

The system attracts user acceptance because of the tutorial features readily available to the user. On occasion, users have executed successfully new programs at the first attempt. However, the development work is arduous and requires workers competent in engineering design, workers competent in computer systems and workers competent in COMRADE software. In addition, a limited number of people are required with considerable expertise in all three areas!

Consideration must be given to system portability. The bulk of the COMRADE software is written in FORTRAN but the structure rests on the INTERCOM facilities associated with the Scope 3.3 system of the CDC 6700 computer. Other computers have similar facilities but the effort involved in conversion to another machine has not yet been addressed.

Consideration must be given to the size and storage cost of the files involved for application to particular engineering problems. For regular application to ship design and construction it is envisaged that an on-line mass storage device will be required.

Consideration must be given to the place of graphics in the particular design application. Interactive and passive graphics facilities are expected in COMRADE but are not available as yet.

In conclusion, COMRADE has demonstrated its acceptability as a base for integrated systems. COMRADE permits the development of an efficient man-machine team, the man always having control over the design process.

Programs gain substantial user acceptance because of the step-by-step tutorial features, because commands are user-oriented and simple to use, and because designs may be executed in a directed logical sequence. Communication between engineers of various disciplines is vastly improved since access to the design file at any time provides authentic information about the current state of development of the design project. Design progress is accelerated and the design manager has tools he may use to direct and monitor design progress.

REFERENCES

1. Rhodes, T., "The Computer-Aided Design Environment Project (COMRADE)," presented at 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.

2. *Feasibility Study of an Integrated Program for Aerospace-Vehicle Design*(IPAD), Statement of work (1-15-2183 Exhibit A) October 1971.
3. *Hydrofoil Analysis and Design Tool* (HANDE), currently under development by the Navy's Hydrofoil Program Office (115), Naval Ship Research and Development Center.
4. Tinker, R., Avrunin, L., "The COMRADE Executive System," presented at 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.
5. Willner, S., Gorham, W., Wallace, M., Bandurski, A., "The COMRADE Data Management System," presented at 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.
6. Chernick, M., "The COMRADE Design Administration System," presented at 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.

The COMRADE executive system

by ROBERT W. TINKER and IRA L. AVRUNIN*

Naval Ship Research and Development Center
Bethesda, Maryland

INTRODUCTION

The role of the executive system

The role of the Executive System within the Computer-Aided Design Environment (COMRADE) has been to provide the focus for the coordination of the overall system capabilities. Such a task is especially difficult given the dynamic atmosphere of the design process.[1,2] Typically one must deal with a multiplicity of concurrent users operating in a highly interactive fashion with large volumes of data, a variety of file types, and an ever increasing array of user programs. In the case of COMRADE, the issue was further complicated by required implementation on a large scale, third generation computer** whereon the users of COMRADE formed only a small portion of the total user community. This implied that the COMRADE Executive could take no special liberties with the computer operating system but was constrained to function within installation imposed conventions.

The duties of the Executive within an integrated design system are typically to supervise the execution of the various operational modules, to provide system security features, and to directly or indirectly account for a battery of software support functions such as program management, file management, menu selection, etc. The structure of the Executive often accounts for the external appearance of the computer-aided design system as a whole. Inevitably it reflects the prejudices and pre-suppositions of its designers. Too often, however, an Executive constrains the functioning of the rest of the system by imposing a philosophy or methodology which limits capabilities for system modification or expansion. Such limitations must be avoided at all cost. This is especially important in an area as vaguely defined as computer-aided design where the detailed specification of requirements is exceedingly difficult to formulate.

An approach toward reconciling these difficulties is to insist that the Executive be constructed in a fashion which promotes the modularity and extensibility of the entire system. If this can be accomplished, one then has a very general purpose Executive—one which can provide a logical framework for building systems and which does not pre-suppose the exact nature of those systems or how they might wish to grow. In fact, such an Executive is now not confined to serving the needs of computer-aided design only, but may be of use wherever the systemization of related programs is desired. The overriding goal of the COMRADE Executive was to embody these qualities of modularity and extensibility and thus inherit as general a nature as possible. In so doing the COMRADE Executive has achieved the ability to support any number of independent subsystems regardless of their individual areas of concern.

To make such an Executive work, it is imperative that capabilities for "operation definition" be available as a fundamental aspect of the system. Although the Executive does not pre-suppose the nature of subsystems implemented under it, clearly decisions regarding their form and appearance must be made at some point. Such decisions may lead to rigorously defined and highly integrated systems or to more loosely defined systems of relatively independent operational modules. The COMRADE Executive provides capabilities, as an adjunct to the Executive itself, for defining sequences of computer processing events which it will perform in response to user requests. The nature of these event sequences, the degree to which they are inter-related, and the external characteristics they exhibit to the user ultimately determine the form, the appearance, and the utility of the subsystem which they embody.

System users

With these considerations it becomes instructive to note that there are actually several classes of COMRADE Executive "users." First there is the subsystem designer. Given the initial need for an integrated system he performs the requisite analysis, determines what the system capabilities should be, and designs or secures the operational modules to perform the required tasks. Moreover, he must answer questions respecting system characteristics. Module size, running time, degree and form of user interaction, and myriad other topics must be considered. When these questions have reached an acceptable level of resolution, the subsystem designer may authorize a first cut at implementation using the

* The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Navy.

** CDC 6700, SCOPE/INTERCOM Operating System

definition capabilities provided in conjunction with the COMRADE Executive. In any event he may be assured that his system will inherit capabilities for growth or change, that program modules or event sequences may be added, deleted, or altered in a relatively painless manner.

The second class of COMRADE Executive System users is comprised of programmer-implementers. Working to the specifications of the subsystem designer, their primary responsibility is coding the operational modules which make up the heart of the subsystem. They are also responsible for the mechanics of encoding the various event sequences to be performed in response to each user request. To accomplish this they must be apprised of the operation definition facilities, communication conventions, and utility modules offered by the Executive system.

The ultimate raison d'etre of COMRADE and indeed of all computer-aided integrated systems is to service the third and most important class of users. This class is comprised of engineers, managers, personnel administrators—individuals whose duties may be greatly facilitated through the use of the computer and more specifically through the use of a carefully constructed and consistent system of computer tools. Such individuals are generally not computer specialists. They seldom are interested in the details of computer control language or in the mechanics of program loading and execution. Generally they are interested in solving their problems—and doing so in a way that seems natural to them. Hopefully, this is a major goal of a computer-aided system: to bring to bear upon a specific task area all the data storage and computational power of the computer and to do so in a manner whereby a specialist in that task area can make use of that power without needing to know the operational details involved.

COMMAND INTERPRETATION

Command procedures

The COMRADE Executive functions in response to requests issued by users situated at keyboard type terminals. Each request initiates a sequence of computer activities—the "event sequence" mentioned previously. For example, such a sequence might be:

(1) Obtain input file: FILE1
(2) Obtain input file: FILE2
(3) Execute program to merge files
(4) Save resulting output file

Or the sequence might be more complex involving user decisions and logical branching, thus:

(1) Ask user for name of input file
(2) Accept name of file from user
(3) Obtain required file from library
(4) Ask user for file disposition
(Analyze or Display)
(5) Accept disposition from user
Analyze? Yes, go to 12
Display? Yes, go to 6
(6) Ask user where to display file
(Teletype or High Speed Printer)
(7) Accept display mode from user
Teletype? Yes, go to 8
High Speed Printer? Yes, go to 10
(8) Execute teletype display routine
(9) Stop
(10) Route copy of file to high speed printer
(11) Stop
(12) Execute analysis module
(13) Stop

Each such sequence of computer activities is termed a "command procedure," and each command procedure is identified by a unique name.

The COMRADE user requests the initiation of a command procedure simply by typing in the identifying command procedure name. This form of the user request—a single word entry—in some cases sacrifices convenience for generality. In some systems, for example, the user request takes the form of a command name generally followed by a number of parameters. These parameters may be optional, multiple choice, or data value entries. If the kinds of requests in such a system are very closely interrelated, and if the words which form the requests are oriented toward a particular discipline, we have what is called a "problem-oriented language."[3] If the requests are of a more general nature and are more specifically computer-oriented, this is usually called a "command language." An advantage of such language-like constructions is fairly obvious: the user can enter both the request (the command name) and the required parameters on the same input line. Hence, there is no delay while the computer prompts for the parameter inputs. On the other hand, there are several disadvantages to such language oriented approaches:

(1) The user is forced to remember, along with the command names, the kinds of input parameters, their order, and their formats.
(2) If the action to be performed by a given command is reasonably complex, a considerable number of input parameters may be required. These might well be difficult to accommodate within the format of a single command.
(3) The structure of the commands is often limited by the kind of general parsing that takes place. Delimiters, data types, etc. are usually pre-established for all commands.

Clearly, command or problem-oriented languages are useful where the actions associated with each command are short, where the number of parameters is small, and where the format of these parameters is not complex. However, the command procedure approach of the COMRADE Executive is considerably more general, as:

(1) The user is prompted for required inputs during the course of command procedure execution.
(2) Substantial amounts of data can be input in a wide variety of formats.
(3) Tutorials, warnings, etc. can be issued before the user enters important data items.
(4) Complex sequences of events requiring many levels of user interaction can be initiated through a single command procedure.

Figure 1 shows the user-system dialogue for a typical command procedure, named, in this case, "RETRIEVAL." User input entries are underlined.

```
????RETRIEVAL

ENTER YES OR NO FOR LIST OF OPERATIONS:YES
OPERATIONS ARE:
    (GET)     BASIC RETRIEVAL
    (QUERY)   QUERY
    (STATS)   FILE STATISTICS
    (DBFILE)  RE-DEFINE DATA FILE
    (HALT)    HALT PROCESSING

IDENTIFY FILE FOR RETRIEVAL
ENTER C IF COMRADE FILE; S IF SCOPE FILE:S
ENTER:
PERMANENT FILE NAME:CAWEPRES
PASSWORD REQD? ENTER YES OR NO:NO
FILE ATTACHED AS:CDMSDB

SELECT AND ENTER OPERATION:QUERY
ENTER YES OR NO FOR TUTORIAL -NO
QUERY?

PRINT WIFE .WHERE.SURNAME .EQ. "KENNEDY";

WIFE              JACQUELINE
PROCESSING COMPLETE
QUERY?

EXIT;

END OF PROGRAM
 STOP

SELECT AND ENTER OPERATION:HALT
FILE USED FOR RETRIEVAL NOW UNLOADED
PROCESSING COMPLETE FOR COMMAND RETRIEVE

????
```

Figure 1—Command procedure RETRIEVAL

Communication regions

In order to direct the flow of control throughout the execution of a command procedure, it is often necessary for the COMRADE Executive to communicate with the various operational modules involved and also with the terminal user. For example, the next step to be executed in a command procedure event sequence might well depend upon prior results arrived at by an operational module or upon a decision made by the user at his terminal. Such results and decisions are communicated to the Executive through two types of communications areas: (1) System Common, and (2) Subsystem Common. Both of these areas are linear arrays, each element within them being referenced by its ordinal displacement from the start of the respective array. Moreover, the two regions are basically scratch areas in that they are local to each terminal user and values within them are not saved from one terminal session to another.

System Common is internal to the COMRADE Executive and exists for a given user from sign-on to sign-off. It is used by the Executive itself to record important items relating to the terminal session, such as the user name, his account number, the name of the subsystem under which he is currently operating, the name of the command procedure which is being executed, etc. It may also be used by any of the various utility modules resident with the Executive for temporary storage of input or output parameters.

Subsystem Common, on the other hand, is external to the COMRADE Executive and exists only for the duration that a user operates under a specific COMRADE subsystem. If a user should switch subsystems in the middle of a terminal session, a new Subsystem Common region (associated with the new subsystem) would be initialized for him at that time. Subsystem Common resides on a disk file and is referenced by random access methods. Routines are provided, in conjunction with the COMRADE Executive, to store and retrieve values in Subsystem Common by element number. These routines may be combined as necessary with the various operational modules which comprise a subsystem in order for the modules to communicate among themselves or with the Executive.

It should be pointed out that the way in which Subsystem Common is used can be an important factor in the design of a subsystem. In a subsystem dealing with ship hull design, for example, different portions of the Common file could be allocated, for the duration of the terminal session, to store different design parameters, such as hull offsets, water-lines, or curves of form. Of course, the operational modules involved would have to be carefully constructed to honor the pre-defined areas and eliminate conflicts. On the other hand, subsystems which may be comprised of largely independent operational modules might wish to use Subsystem Common only sparingly, perhaps simply to communicate control information to the COMRADE Executive.

It should also be noted that Subsystem Common, since it is sequentially organized and relatively transient, is not meant to take the place of a large-scale structured data base such as is needed for an integrated system of some size. Such capabilities are amply provided for in another area of the overall COMRADE Project: The COMRADE Data Management System.[4]

Procedure control blocks

The COMRADE Executive executes a command procedure by interpretively processing control information contained in

what are called "procedure control blocks." Each command procedure is represented by an ordered set of such procedure control blocks. Their interpretation by the Executive effects the event sequence associated with the corresponding command procedure.

There are six specific types of procedure control blocks, each representing a specific and very primitive function. The six functions are:

(1) Preset values in System or Subsystem Common. Real, integer, alphanumeric, or octal values may be used. This function is employed to establish input parameters within the Common regions for use by Executive System utility routines or by subsystem operational modules.
(2) Move values from one location to another within the Common regions. Values may be moved within System Common, within Subsystem Common, or between the two regions. The move function is generally used to gather parameters output from a given utility routine or operational module, and to place them where they can be stored or used for input by another utility routine or operational module.
(3) Perform an unconditional branch. The COMRADE Executive normally processes the procedure control blocks for a given command procedure in a sequential fashion. The branch function causes sequential interpretation to be interrupted and resumed at some other point.
(4) Perform a conditional branch. Values within the Common regions are tested against each other or against pre-established constants. Branching as above will take place only if specified relational conditions are met. The usual relational operators are allowed: equal, not equal, greater than, less than, etc.
(5) Execute a program. This function causes a specified utility routine or operational module to be executed immediately. The program may be core resident with the Executive (typically a short utility routine useful to all subsystems) or it may be a substantially larger operational module pertinent to a specific subsystem and residing externally on a disk file. In the former case, the Executive simply transfers to the required routine. In the latter case, the appropriate file is made available, the COMRADE Executive is swapped out, and the required operational module is loaded and executed. Upon completion of the operational module, the COMRADE Executive is recalled and procedure control block interpretation continues.
(6) Halt procedure control block interpretation. This function signifies the end of a given command procedure. The Executive prepares for the user to issue a new command procedure name.

Note that it is the very atomic nature of these functions which largely contributes to the generality of the COMRADE Executive. For if, instead, the procedure control blocks represented large macro-level functions, the command procedures embodied by them would be more or less compelled to adopt the characteristics imposed by those high level functions. For example, if a given type of procedure control block of itself invoked, say, terminal I/O, then it could be concluded that such I/O would be accomplished in a specific and unalterable manner; namely, according to the nature of the routine executed when this procedure control block was encountered. But if, on the other hand, terminal I/O is accomplished (as indeed it is) by an execute block (type 5, above) specifying the name of a module to perform the I/O, then we are free to add other such modules to perform, as we like, different styles of terminal I/O. We might, for instance, have two terminal I/O modules, one of which performs FORTRAN-like formatted I/O, and another which is used to provide a free-format capability. The important point is that, at the level of command procedure interpretation, no prejudgments are made respecting the nature of the various modules which might be executed.

COMRADE SUBSYSTEMS

Subsystem characteristics

A COMRADE subsystem is comprised of a set of command procedures possessing some measure of commonality. For example, a ship hull design subsystem would contain command procedures which initiated events related to that discipline, while the command procedures of an information retrieval subsystem would likely deal with the manipulation of a particular kind of data base and, perhaps, with report generation. A COMRADE subsystem, generally, is also associated with a particular set of users. It is unlikely that a civil engineer working on a highway design project would have the need or desire to issue command procedures concerned with pipe sizing aboard a destroyer. Hence the civil engineer would be an acknowledged user of subsystem "ROADS" but not of subsystem "PIPES." Still a third characteristic of a subsystem is the set of operational modules which are used by the command procedures to perform the required tasks within that subsystem. These three groupings —a set of command procedures, users, and operational modules—may be thought of as logically defining a COMRADE subsystem.

While it is generally true that a particular set of COMRADE users may be associated with a particular subsystem, it might also be true that not all of these users are allowed to use all command procedures implemented under the subsystem. For example, it might be desirable that only a single individual have the power to make crucial updates to the master design file within a preliminary ship design subsystem. It is only after careful and critical inspection of the updating data (provided by design engineers with the aid of the available command procedures) that this individual would initiate a special command procedure to accomplish the required update. This special command procedure, then, is usable only by the authorized data administrator and not by the other users of the subsystem. To provide this often needed user-command procedure control, the COMRADE

Executive uses a simple "lock and key" mechanism. Each command procedure is associated with a specific access lock which is established when the command procedure is defined; and each user, when he enters a subsystem, is assigned a specific access key. Without going into the details involved, suffice it to say that if a user's key "fits" a command procedure's lock, then the user may employ that command procedure.

Subsystem general

A signed-on COMRADE user may operate under only one subsystem at a time (i.e., employ only that subsystem's command procedures). However, if he is authorized to use more than one subsystem it is a simple matter to switch from one to another within the same terminal session. In general, of course, he would rarely have the need or desire to do so. An exception to this, however, concerns a somewhat special subsystem, called the "GENERAL" subsystem.

When a user first signs-on to COMRADE, the Executive places him in the GENERAL subsystem (special procedures, however, may allow a user to go directly into a pre-specified subsystem). The GENERAL subsystem contains a number of command procedures not specifically related to a particular discipline but of more general use to all COMRADE users. In fact, even after he has signed-on to another subsystem, the COMRADE user may still employ the command procedures defined under the GENERAL subsystem! The GENERAL subsystem is unique in this respect.

Subsystem sign-on procedures

Among the command procedures available under the GENERAL subsystem is a special group called "subsystem sign-on procedures." These command procedures are employed by COMRADE users whenever they wish to enter a subsystem (from GENERAL) or to switch from one subsystem to another. The names of these command procedures, as might be expected, are in fact the names of the subsystems recognized by the COMRADE Executive. Thus, if a user wishes to enter the Integrated Ship Design System (ISDS), he merely issues command procedure "ISDS" immediately after signing-on to COMRADE.

The subsystem sign-on procedures implemented under subsystem GENERAL are expected to follow a standard protocol as they go about the business of signing a user on to a subsystem. Among their required tasks are: (1) sign-off the previous subsystem, if necessary; (2) verify that the user may use this subsystem; (3) assign the user a command procedure access key; (4) initialize Subsystem Common; and (5) locate the procedure control blocks for the subsystem. Actually, a wide latitude is given the various subsystem sign-on procedures respecting what they might accomplish. They may be fairly simple or quite complex. However the steps listed above are indicative of some of the tasks that they *must* carry out. The COMRADE Executive provides a number of resident utility modules which are useful in abetting these tasks.

COMMAND PROCEDURE DEFINITION

A command procedure is encoded as an ordered set of procedure control blocks. When a COMRADE user requests the execution of a command procedure, the correct set of blocks is located, and the Executive begins to interpret and process them. This processing effects the event sequence associated with the command procedure.

Before the command procedure can be executed, however, it must have been previously defined; that is, the procedure control blocks must have been established. This is accomplished with the aid of a special language: the Procedure Definition Language (PDL). PDL programs are coded by subsystem designer/implementers to define the command procedures within a subsystem. The output of a PDL program is a set of procedure control blocks corresponding to a command procedure.

The Procedure Definition Language consists of two kinds of statements: Phase I statements, and Phase II statements. A one to one correspondence exists between the Phase I statements and the functions represented by the procedure control blocks as described previously. The Phase II statements exist simply for convenience. The functions which they incorporate can in each case be represented by an appropriate sequence of Phase I statements.

Following is a description of the format and use of the PDL statements. Items enclosed in brackets [ ] are optional; braces { } denote repeatable quantities; and elements listed vertically

$$\left(\begin{array}{r}\underline{\mathrm{x}}\\ \text{e.g. } \underline{\mathrm{y}}\\ \mathrm{z}\end{array}\right)$$

indicate that one of the elements must be selected. Capitalized items are written as shown; lower case elements are variable.

The general form of a PDL statement is:

[lable;] OPERATION △ operand(s)

where: "label" is a variable name which may be used to identify any PDL statement. Control may be transferred to any labeled statement through the GOTO statement described below.

"OPERATION" is a keyword which indicates the statement function.

"operands" are statement parameters which vary in form according to the statement type.

The Phase I PDL statements are used and coded as follows:

(1) statement name: PRESET
function: establish values in System or Subsystem Common

form: PRESET $\left\{\begin{matrix}\underline{\mathrm{S.}}\\ \underline{\mathrm{C.}}\end{matrix}\right.$ ordinal = value [(quantity)]$\Big\}$,

{ }, etc.

where: "S." indicates System Common
"C." indicates Subsystem Common
"ordinal" is the common element number to be set
"value" is the real, integer, octal, or alphanumeric value to which the common element is set.
"quantity" is the number of consecutive elements within Common (beginning with "ordinal") to be assigned the indicated "value." If omitted, 1 is assumed.

example: To set Subsystem Common location 25 to the character string "HORSEPOWER," and to load System Common locations 56 through 60 with the value "25.8," one would code:
PRESET C.25 = 'HORSEPOWER,'
S.56 = 25.8(5)

(2) statement name: MOVE

function: Move values within or between System and Subsystem Common

form: MOVE $\left\{\begin{matrix}\text{S.}\\\text{C.}\end{matrix}\right.$ ordinal $-$ $\left.\begin{matrix}\text{S.}\\\text{C.}\end{matrix}\right\}$ ordinal [(quantity)], { }, etc.

where: "quantity" is the number of consecutive elements (beginning with "ordinal") to be moved. If omitted, 1 is assumed. Other parameters are described above.

example: To move three values from System Common locations 65-67 to Subsystem Common locations 533-535, and to move one value from System Common location 12 to System Common location 24, one would code:
MOVE S.65 – C.533(3), S.12 – S.24

(3) statement name: unconditional GOTO

function: Perform an unconditional branch

form: GOTO label

where: "label" is the statement label to which control is transferred

example: To interrupt sequential command procedure interpretation and to resume with another PDL statement labeled "THERE," one would code:
GOTO THERE

(4) statement name: conditional GOTO

function: perform branching as above only if the parenthesized condition is true

form: GOTO label $\left(\dfrac{\text{S. ordinal}}{\text{C. ordinal}}\text{ . rel. }\dfrac{\text{value}}{\text{S. ordinal}}\right)$

where: "rel" is one of the following logical operators —
EQ—equal
NE—not equal
LT—less than
GT—greater than
LE—less than or equal
GR—greater than or equal
other parameters are as described previously

example: To branch to a PDL statement labeled "ERROR" if System Common location 261 is not equal to zero, one would code:
GOTO ERROR (S.261 .NE. 0)

(5) statement name: EXECUTE

function: locate, load, and execute the utility routine or operational module whose name is found in System Common locations 6-9.

form: EXECUTE [program name]

where: "program name," if present, is first PRESET into System Common locations 6-9 by the Executive System before the EXECUTE function is performed.

example: To execute an operational module, named "INTERACTIVEUPDATE," one would code:
EXECUTE INTERACTIVEUPDATE

(6) statement name: STOP

function: terminate processing of this command procedure

form: STOP

The Phase II PDL statements are used and coded as follows:

(1) statement name: CONTROL

function: execute one or more operating system control statements

form: CONTROL {control statement}, { }, etc.

where: "control statement" is an operating system control statement

example: To rewind and unload a file, named "CDMSDB," one would code:
CONTROL REWIND(CDMSDB), UNLOAD(CDMSDB)

(2) statement name: PRINT

function: perform terminal output using FORTRAN formatting

form: PRINT '(format)' $\left[,\left\{\begin{matrix}\text{S.}\\\text{C.}\end{matrix}\right.\right.$ ordinal $\left[\text{(quantity)}\right]\left.\right\}$, { }, etc.

where: "format" represents a FORTRAN output specification other parameters are as described previously

example: To print the character string "WEIGHTS ARE:" followed by five values in Subsystem Common locations 600-604, one could code:
PRINT '(*WEIGHTS ARE:* 5F10.4),' C.600(5)

```
          Statement                                    Comment
          PRINT '(* FILE NAME?-*)'                     Ask for file name
          READ '(4A10)', S.263(4)                      Read response
          PRESET S.262 = 'INFILE'                      If file known to COMRADE,
          EXECUTE PF                                   get passwords
          GOTO ER1(S.261 .NE. 0)                       Error if file not known
          EXECUTE ATTX                                 Get file from library
          GOTO ER2 (S.261 .NE. 0)                      File not in library error
          PRINT '(*DISPOSITION?-*)'                    Ask user for disposition
          READ '(A10)', S.200                          Read response
          GOTO ANALYZE (S.200 .EQ. 'ANALYZE')          Branch if "analyze"
          PRINT '(*WHERE(TTY,HSP)?-*)'                 Ask user where to display
          READ '(A3)', S.200                           Read response
          GOTO TTY (S.200 .EQ. 'TTY')                  Branch if "teletype"
          CONTROL BATCH,INFILE,PRINT.                  Print file at central site
          STOP                                         End procedure
TTY;      CONTROL COPY,INFILE,OUTPUT.                  Print file at teletype
          STOP                                         End procedure
ANALYZE;  EXECUTE STATISTICALANALYSIS                  Analyze data file
          STOP                                         End procedure
ER1;      PRINT '(*FILE NOT KNOWN*)'                   Error message
          STOP                                         End procedure
ER2;      PRINT '(*FILE NOT IN LIBRARY*)               Error message
          STOP                                         End procedure
```

Figure 2—PDL example

(3) statement name: RE AD
function: perform terminal input using FORTRAN formatting

form: READ '(format),' $\left\{\frac{\text{S.}}{\text{C.}}\right\}$ ordinal [(quantity)]$\Big\}$,

$\{$ $\}$, etc.

where: parameters are as in the "PRINT" statement

example: To input three alphanumeric values from the terminal and store them into System Common locations 300-302, one could code:
READ '(A10),' S.300(3)

Figure 2 shows a sample PDL program which could be used to effect the second event sequence noted at the start of the second section of this paper.

SUMMARY

The COMRADE Executive System is a general purpose interactive supervisor. It facilitates the definition and construction of subsystems of integrated programs by providing a framework under which they may operate. Since no a priori assumptions are made regarding the nature of these programs, no restrictions are imposed respecting the characteristics of the subsystems which they embody. The COMRADE Executive can support highly integrated subsystems where the programs involved are typically molecular and where communication among them is crucial, or more loosely organized subsystems comprised of independent modules.

The Executive services its ultimate users by initiating possibly complex sequences of programs, perhaps as modified by user decisions, in a manner whereby the user is not burdened with the computer-related details involved.

REFERENCES

1. Brainin, J., "Use of COMRADE in Engineering Design," presented at the 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.
2. Rhodes, T., "The Computer-Aided Design Environment (COMRADE) Project," presented at the 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.
3. Roos, D., *ICES System Design*, second edition, 1967, M.I.T. Press.
4. Willner, S., Banduaski, A., Gorham, W., and Wallace, M., "The COMRADE Data Management System," presented at the 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.

COMRADE data management system

by STANLEY E. WILLNER, ANN E. BANDURSKI, WILLIAM C. GORHAM, JR. and MICHAEL A. WALLACE*

Naval Ship Research and Development Center
Bethesda, Maryland

INTRODUCTION

The Executive system of COMRADE[1] has provided the basic "living quarters" for all members of the computer-aided design team. Three distinct types of individuals have finally been united in this environment: *the user*, the person with little or no computer experience, possibly an aeronautical engineer or naval architect who received his training before electronic data processing was in vogue; *the application programmer*, not necessarily the recipient of any schooling regarding the design he is helping to implement; and *the subsystem designer*, the link between the other two, the person who must be part engineer, part computer scientist, and—most important—part nursemaid, soothing the inevitable wounds of the other team members, each of whom is a little too close to his own area of specialization to see, at all times, the design system in a completely objective manner.

But now an obvious problem arises with respect to the role of the subsystem designer. This subsystem designer is neither ever-present nor all-knowing. He can perhaps plan a smooth design as tasks proceed from the initial feasibility model to the finished product, but he can most assuredly not monitor the immense flow of data that is emerging from the design: data produced by one design module for use in several other design modules; data which is output from a design module and must be immediately available upon request by the engineer-user to aid in his decision-making process; data which serves as the only means of communication between designers in different disciplines within an integrated system; and finally, data which represents the finished product, the design of a nuclear submarine, a supersonic jet, or a multi-million dollar medical school and hospital.

There must be a data base management system to handle the vast number of data items considered to be the output of the preliminary design stage of a U.S. Navy ship.[2] This data base management system must provide capabilities for the creation and management of complex file and data structures; the storing of enormous quantities of data; the selective, "item by item" update of this information; and the rapid retrieval of data base information either directly by name or by conditional expressions.

Moreover, these capabilities should not only be extremely easy to use by non-computer-oriented personnel, but should also be efficient (both in processing time and main memory utilization) when dealing with great quantities of data; such efficiency is expected by the application programmer and subsystem designer, both of whom are responsible for the overall use of computer resources within the design system.

The different backgrounds and needs of the people involved in computer-aided design cannot be emphasized too much, for it was the existence of such diversity that led to the development of the COMRADE Data Management System (CDMS).

The CDMS capabilities are built upon two major software components, these being the

- Data Storage Facility—Programs to build and randomly process user-named blocks of data on disk storage; and programs to build and process "inverted" element lists for retrieval by value; and the
- Block Type Definition Facility—A mechanism for naming, defining, and specifying individual data fields within a block, thereby enabling conversational use of the data base.

Building upon these components, the CDMS has attempted to satisfy the requirements of the developers and users of a computer-aided design system by providing various functional capabilities suited to different processing modes:

- Interactive commands
- Stand-alone batch programs
- Subroutine library

This introduction has tried to bring home the fact that it is no simple job to satisfy all members of the computer-aided design team at the same time. In particular, papers on computer-aided design are rarely equally valuable to

* The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Navy.

all involved in the design effort. For this reason, the remainder of this paper has been written primarily with only one type of user in mind, the ultimate user of the system, the design engineer. The focus is on capabilities—what are they, and how are they used? Explanations of programming techniques, for the most part, are omitted. Therefore, the Data Storage Facility, the mechanism for creating and maintaining data blocks (logical records) and "inverted" lists is not discussed in this paper. That facility is discussed, however, in Reference 3.

BLOCK TYPE DEFINITION FACILITY

While it is possible for one person, responsible for his own programs and data files, to employ a storage management capability such as the COMRADE Data Storage Facility, he then must be aware of the address and data structure of each block for use in data processing. The complexity of computer-aided design data bases, the multiplicity of users and design tasks, and the diverse roles of the members of the design team in an integrated system, however, render such an approach to data management impractical.

And so, enter the COMRADE Block Type Definition Facility (BTDF)! If the Data Storage Facility is the "thought," then the BTDF is the "voice," providing the means of communication between the engineer-user and the subsystem designer, the subsystem designer and the application programmer, the programmer and the programs, and, in general, between all of these data users and their data bases.

The BTDF is the mechanism by which the so-called data administrator defines the formats of all data blocks within the data base, thereby giving names by which each data item may be referenced. A CDMS data base may consist of a number of block type definitions. Some subset of the total number of data blocks within the data base will be thought of as belonging to each block type. For example, Figure 1 illustrates a "Presidents" Data Base,* in which five block types are defined and relationships structured. A printout of one of these types—the block type PRES—is shown in Figure 2. For block type PRES, which describes the personal information regarding each United States president, George Washington through Lyndon Johnson, there are 35 data blocks. There are 45 data blocks of type ELECTION, one for each presidential election between 1789 and 1964; 53 data blocks of type ADMIN; 90 of type CONGRESS; and 50 of type STATES. Thus five "block types" define the format of all 273 data blocks within the data base.

A block type, and therefore all data blocks of this type, consists of named data elements, repeating groups, and sub-blocks. A data element—the smallest quantity that may be referenced within a CDMS data base—may be defined as one of four data types: alphanumeric, real, integer, or pointer. An element of any of these types may

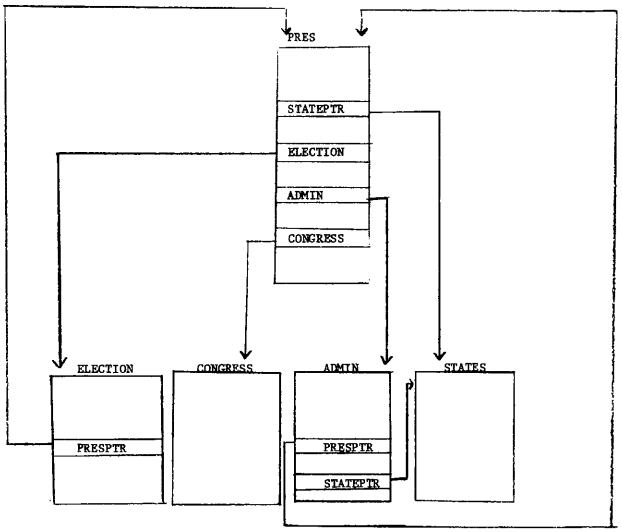


Figure 1—"Presidents" data base structure

be defined to be a single value or an array. In addition, every single-valued element may or may not be inverted. Therefore, every element appearing in a block type definition will have one status out of a possible twelve.

There may be a need in some data blocks to store more than one occurrence of a data element or elements. To this end, a set of consecutively defined elements may be united by membership within a named "repeating group." For example, a set of points $(X_i, Y_i Z_i)$ on a curve may be defined as three single-valued real elements, all members of repeating group COORD. The user is now free not only to request the retrieval of all X's, or of a specific Y or Z, but also to retrieve, say, the third point on a curve by asking for the third "occurrence" of repeating group COORD. Although the name "repeating group" implies repetition, this is by no means necessary. It is sometimes very handy for a higher level name to be assigned to a group of data elements even if there can never be more than one occurrence for the group within the data block (e.g., elements STREET, CITY, STATE, ZIPCODE might be grouped under the repeating group ADDRESS).

The elements of a block type may be divided into sub-blocks. The use of sub-blocks is twofold. First, it provides still another level of data element grouping so that one name may be used to refer to perhaps 20 or 30 data elements, thereby greatly simplifying a retrieval request. In addition, if a retrieval or update request on an element indicates its sub-block membership, the transaction will be processed in a more efficient manner than if the block type were not partitioned in this way or if no sub-block information were provided.

The mechanics of block type definition are indeed trivial. The "define block type" program prompts the user for a block type name, number of sub-blocks and first sub-block name. The user then defines the elements of the first sub-block (i.e., element name, status). When all elements are defined, the user types END. The program then asks for repeating group definitions (i.e., name and

* A data base containing information regarding the U.S. Presidents has been used as the basis for all examples in this paper.

```
BLOCK TYPE - PRES
    NUMBER OF SUB-BLOCKS-    3
    SUB-BLOCK   1  DEFINITION -
         NAME-PERSONAL
         ELEMENT     1 -  SURNAME    ALPHA              INVERTED
         ELEMENT     2 -  FIRSTNAM   ALPHA
         ELEMENT     3 -  INITIAL    ALPHA
         ELEMENT     4 -  MONTHB     ALPHA
         ELEMENT     5 -  DAYB       INTEGER
         ELEMENT     6 -  YEARB      INTEGER            INVERTED
         ELEMENT     7 -  STATEB     ALPHA              INVERTED
         ELEMENT     8 -  STATEPTR   POINTER
         ELEMENT     9 -  HEIGHT     ALPHA
         ELEMENT    10 -  PARTY      ALPHA              INVERTED
         ELEMENT    11 -  COLLEGE    ALPHA
         ELEMENT    12 -  ANCESTRY   ALPHA              INVERTED
         ELEMENT    13 -  RELIGION   ALPHA              INVERTED
         ELEMENT    14 -  OCCUP      ALPHA      ARRAY
         ELEMENT    15 -  MONTHD     ALPHA
         ELEMENT    16 -  DAYD       INTEGER
         ELEMENT    17 -  YEARD      INTEGER
         ELEMENT    18 -  CAUSE      ALPHA
         REPEATING GROUP     1  DEFINITION -
         NAME-NAME
              ELEMENT      1
              ELEMENT      2
              ELEMENT      3
         REPEATING GROUP     2  DEFINITION -
         NAME-BIRTH
              ELEMENT      4
              ELEMENT      5
              ELEMENT      6
              ELEMENT      7
         REPEATING GROUP     3  DEFINITION -
         NAME-DEATH
              ELEMENT     15
              ELEMENT     16
              ELEMENT     17
              ELEMENT     18
    SUB-BLOCK   2  DEFINITION -
         NAME-FAMILY
         ELEMENT     1 -  FATHER     ALPHA
         ELEMENT     2 -  MOTHER     ALPHA
         ELEMENT     3 -  WIFE       ALPHA
         ELEMENT     4 -  MONTHM     ALPHA
         ELEMENT     5 -  DAYM       INTEGER
         ELEMENT     6 -  YEARM      INTEGER
         ELEMENT     7 -  CHILDREN   INTEGER
         REPEATING GROUP     1  DEFINITION -
         NAME-MARRIAGE
              ELEMENT      3
              ELEMENT      4
              ELEMENT      5
              ELEMENT      6
              ELEMENT      7
    SUB-BLOCK   3  DEFINITION -
         NAME-HISTORY
         ELEMENT     1 -  ELECTION   POINTER    ARRAY
         ELEMENT     2 -  ADMIN      POINTER    ARRAY
         ELEMENT     3 -  CONGRESS   POINTER    ARRAY
```

Figure 2—Block type "PRES"

```
????BTMAINT
WOULD YOU LIKE A LIST OF THE OPERATIONS-YES
OPERATIONS ARE:
 DEFINE BLOCK TYPE  (D)
 MODIFY OR DELETE BLOCK TYPE  (M)
 PRINT BLOCK TYPE  (P)
 RE-IDENTIFY BLOCK TYPE FILE  (R)
 HALT  (H)
ARE YOU CREATING A NEW BLOCK TYPE FILE?NO
ENTER PFN OF BLOCK TYPE FILE-CAMEPRESIDENTS
ENTER OPERATION-D
BLOCK TYPE--ELECTION
NUMBER OF SUB-BLOCKS--1
SUB-BLOCK  1 DEFINITION
NAME--SUB1
ELEMENT  1--YEAR,I,V
ELEMENT  2--WINNER,A,V
ELEMENT  3--PRESPTR,P
ELEMENT  4--WPARTY,A,V
ELEMENT  5--VOTES,I
ELEMENT  6--OPPNAME,A
ELEMENT  7--OPPPARTY,A
ELEMENT  8--OPPVOTES,I
ELEMENT  9--END
REPEATING GROUP  1 DEFINITION
NAME--OPPONENT
ELEMENTS--6,7,8
REPEATING GROUP  2 DEFINITION
NAME--END

FORMAT TABLE UPDATE SUCCESSFUL
BLOCK TYPE--END
 STOP
ENTER OPERATION-P
ENTER BLOCK NAME-ELECTION
                  BLOCK DESCRIPTION


    BLOCK TYPE-ELECTION
    NUMBER OF SUB-BLOCKS-    1
    SUB-BLOCK   1DEFINITION-
         NAME-SUB1
         ELEMENT     1-YEAR     INTEGER       INVERTED
         ELEMENT     2-WINNER   ALPHA         INVERTED
         ELEMENT     3-PRESPTR POINTER
         ELEMENT     4-WPARTY   ALPHA         INVERTED
         ELEMENT     5-VOTES    INTEGER
         ELEMENT     6-OPPNAME ALPHA
         ELEMENT     7-OPPPARTYALPHA
         ELEMENT     8-OPPVOTESINTEGER
         REPEATING GROUP     1DEFINITION-
         NAME-OPPONENT
              ELEMENT      6
              ELEMENT      7
              ELEMENT      8
ENTER BLOCK NAME-END
END OF BLOCK DESCRIPTION
 STOP
ENTER OPERATION-H
PROCESSING COMPLETE
```

Figure 3—Processing block types using command BTMAINT

the ordinal positions of the member elements). The definition of repeating groups is also terminated by END at which point the second sub-block (if any) will ensue. When all sub-blocks have been defined, the program asks for another block type. When the last block type has been defined, the user once more types END, this time to terminate program execution. This program is part of an interactive command procedure, BTMAINT, which also includes programs to modify and display block type definitions. BTMAINT is indicative of all data management command procedures in its handling of file retrieval and cataloging functions; that is, it relieves the user of a great deal of the "bookkeeping" burden by submerging this aspect of data base processing behind a front of everyday conversation. Figure 3 illustrates a typical session with this command procedure.

MODES OF USING CDMS

Batch programs

COMRADE is basically a software system for the interactive designer and CDMS is basically a data management system for the interactive user. However, certain data management tasks are more suited to stand-alone batch processing. The first such task that comes to mind is a massive data base loading procedure. This procedure might occupy the central processor for minutes and some other resources of the computer system for hours. Therefore, such a run would best be made at a low-priority rate during non-prime time for financial reasons. The COMRADE Bulk Data Loader is such a program. This program will accept card image input specifying data blocks to be defined and values to be assigned to elements within these data blocks. A new data base may be created by the Bulk Data Loader or data blocks may be added to an existing data base.

The input to this program consists of Block Type records (only when the next block to be defined is of a different type than the previous block), Block Name records, and, optionally, Sub-block records. The majority of the input, however, is found in the Data records. Each Data record may define a variable number of data fields of the format

element name/value (for single-valued elements)

```
BLOCK TYPE=PRES
BLOCK NAME=LINCOLN
SUBBLOCK=PERSONAL
SURNAME/"LINCOLN" FIRSTNAM/"ABRAHAM" INITIAL/" " $ENDRGO
MONTHB/"FEBRUARY" DAYB/12 YEARB/1809 STATEB/"KENTUCKY"
$ENDRGO STATEPTR/KENTUCKY HEIGHT/"6FT. 4IN."
PARTY/"REPUBLICAN" ANCESTRY/"ENGLISH" OCCUP/2/"LAWYER" "FARM WORK"
MONTHD/"APRIL" DAYD/15 YEARD/1865 CAUSE/"ASSASSIN." $ENDRGO
SUBBLOCK=FAMILY
FATHER/"THOMAS" MOTHER/"NANCY" WIFE/"MARY" MONTHM/"NOVEMBER"
DAYM/4 YEARM/1842 CHILDREN/4 $ENDRGO
SUBBLOCK=HISTORY
ELECTION/2/E1860 E1864  ADMIN/2/A21 A22  CONGRESS/3/C37 C38 C39
$ENDER
BLOCK NAME=WILSON
SUBBLOCK=PERSONAL
SURNAME/"WILSON" FIRSTNAM/"WOODROW" INITIAL/" " $ENDRGO
MONTHB/"DECEMBER" DAYB/28 YEARB/1856 STATEB/"VIRGINIA" $ENDRGO
STATEPTR/VIRGINIA HEIGHT/"6FT. 0IN." PARTY/"DEMOCRATIC"
COLLEGE/"PRINCETON" ANCESTRY/"ENGLISH" RELIGION/"PRESBYT."
OCCUP/2/"LAWYER" "TEACHER" MONTHD/"FEBRUARY" DAYD/3 YEARD/1924
CAUSE/"HEART DIS." $ENDRGO
SUBBLOCK=FAMILY
FATHER/"JOSEPH" MOTHER/"JESSIE" WIFE/"ELLEN" MONTHM/"JUNE"
DAYM/24 YEARM/1885 CHILDREN/3 $ENDRGO WIFE/"EDITH" MONTHM/"DECEMBER"
DAYM/18 YEARM/1915 CHILDREN/0 $ENDRGO
SUBBLOCK=HISTORY
ELECTION/2/E1912 E1916 ADMIN/2/A37 A38 CONGRESS/4/C63 C64 C65 C66
$ENDER
$EXIT
```

Figure 4—Input to bulk data loader for block type "PRES"

or

element name/dimension/value value . . . (for array elements)

Alphanumeric values must be surrounded by quotes. The flags $ENDRGO, $ENDER, and $EXIT signify the end of a repeating group occurrence, the end of a data block, and the end of the input stream, respectively.

Figure 4 is the Bulk Data Loader input for data blocks LINCOLN and WILSON. The definition of block type PRES in the previous section might help the reader to follow the input stream.

The user is by no means required to load all defined data elements in all data blocks for a given block type. On the contrary, it is not uncommon to load data "currently available," leaving large gaps in every data block in the entire data base for future use. To fill these gaps, the user has at his disposal the COMRADE Bulk Data Modifier, another program to be used in the batch mode. This program permits the user to add, modify, and delete data in existing data blocks while using an input format almost identical to that of the Bulk Data Loader.

Subroutine library

Much of the COMRADE development effort paralleled the development of the Integrated Ship Design System (ISDS) at NSRDC.* ISDS, a computer-aided design system which was to use the COMRADE software, provided many specific requirements that influenced the direction taken by the COMRADE team. In this respect data management was no exception. The pilot model of ISDS, demonstrated in the spring of 1970, revealed several drawbacks in the data management techniques then being employed. The most glaring of these deficiencies was the inability of the ISDS application programs to dynamically access the data base during execution. What this meant was that all data that might be needed from the data base for a given design task had to be retrieved and stored on a temporary working file for subsequent application processing. In some cases, large quantities of data were retrieved, and then, due to a design decision that could not be made prior to retrieval, more than 50 percent of this data went unused in the processing of the design task. Likewise, the update of the ship design data base was interrupted by "the middle man," a temporary file for storing update transactions during the gap in time between the termination of the applications program and the execution of an independent update procedure.

Clearly, a library of subroutines was required that permitted dynamic data base retrieval and update from a FORTRAN applications program. The examination of several existing software packages* that did provide this FORTRAN interface, but at a primitive level, led to the conclusion that the development of a data management system which satisfied the requirements of both the FORTRAN applications programmer and the terminal user, the design engineer, was indeed necessary.

Interactive command procedures

While CDMS provides a FORTRAN-callable subroutine library for application programmers to build, process, and retrieve data, the ultimate user of a computer-aided design system is the design engineer. Once a design subsystem of COMRADE reaches the production mode, the application programmer and the subsystem designer fade into the background as much as is possible, while the design engineer steps forward. He is the man who works at the remote terminal day after day performing his design tasks. Just as he must be able to communicate with his applications in a conversational manner, so must he be able to "converse" with his data base through general purpose data management programs.

To this end, COMRADE has provided three interactive command procedures, BTMAINT, UPDATE, and RETRIEVAL. BTMAINT, the procedure for defining, modifying, deleting, and displaying block types is not really intended for use by the design engineer; rather, its use would probably be restricted to the subsystem designer or data administrator, the person or people responsible for the data base structure.

The primary purpose of the UPDATE command procedure is the selective inspection and modification of data items within the data base. The two programs comprising this procedure are the Basic Retrieval Program and the Interactive Update Program. In both programs the user specifies a data block name and a particular "item" within that block, and the programs will display (Basic Retrieval) or modify (add, delete) the item (Interactive Update). An "item" may be

* Informally documented in two reports, the first by R. Stevens and T. Rhodes (Jan 1969); the second by T. Corin and T. Rhodes (Feb 1971).

* The examination of these software packages is discussed in the Corin and Rhodes informal report referenced earlier.

- a non-repeating group element
- one or all of the occurrences of an element within a repeating group
- one or all of the existing occurrences of an entire repeating group
- a new repeating group occurrence to be added
- a sub-block
- an entire data block

The Basic Retrieval Program may also be found in the command procedure RETRIEVAL. Another program accessible through this procedure is the File Statistics Program. This program retrieves and displays information regarding the data base, such as

- name and size of all data blocks
- name and description of any or all block types
- name and value range of all inverted elements

While the CDMS Basic Retrieval Program provides a mechanism for data retrieval of a certain nature, namely by physical location, the COMRADE Query Processor satisfies another type of retrieval request, i.e., retrieval by condition. The goal of the Query Processor was to provide the user with a language, not totally unlike English, with which he could conditionally query the data base on *both* element values and file structure. This program is truly the union of all features within CDMS: the Data Storage Facility which contains the software necessary to process "inverted" element lists thereby, providing the rapid response necessary to an on-line query system;[3] the Block Type Definition Facility, essential to the expression of a query; and file structure traversal via elements of the "pointer" data type.

The syntax of a query is

VERB OBJECT LIST OF CLAUSE WHERE CLAUSE;

VERB—The destination of the output of a particular query is governed by the user's choice of verb. PRINT will direct all output to the user's terminal. FILE will return only diagnostics directly to the user and write all query results on a scratch file. Prior to the completion of command procedure RETRIEVAL, the user will have the opportunity to dispose of this file as he sees fit, either to have it printed at one of several remote sites, or to keep this file for further use.

OBJECT LIST—The object list is simply a list of those items to be printed or filed for all "hit" blocks, i.e., for all data blocks satisfying the conditions of the query. Valid object list items are

- element names
- repeating group names
- BN (block name)
- BT (block type)

The appearance of a repeating group name in the object list will result in the output of all member elements for the "hit" data blocks. A maximum of twenty items may be requested, with a repeating group name counting as *one*, regardless of the number of member elements.

OF CLAUSE—.OF. search path

where 'search path' consists of a starting data block name and a sequence of element names separated by slashes, each element being of "pointer" data type. This arbitrary path will be traversed, with all terminal data blocks considered to be "hits." The search path may consist of zero to eight pointer names, with zero signifying a basic retrieval operation.

Examples

(a) PRINT WIFE .OF. LINCOLN;
This is simply a basic retrieval. The element WIFE in data block LINCOLN will be retrieved and printed.

(b) PRINT WIFE .OF. E1960/PRESPTR;
The data block E1960 is of block type ELECTION. One of the elements, PRESPTR, of the block type, is a pointer to the data block containing the personal information regarding the man who won that election, i.e., Kennedy. Therefore the "answer" is Jacqueline.

(c) PRINT CAPITAL .OF. E1960/
PRESPTR/STATEPTR;
In data block KENNEDY (i.e., E1960/PRESPTR), there is an element STATEPTR, a pointer to JFK's state of birth, Massachusetts. Hence the output of the query is Boston.

These simple examples do not illustrate the possibility of a rapidly expanding tree structure such as

.OF.BLOCK1/PTRA/PTRB

where BLOCK 1 has a 100 word pointer array PTRA, and each of the data blocks referenced by this array contains a 100 word pointer array, PTRB. Quickly this search path has produced a "hit" list of 10000 data blocks.

Now, a final word on the OF CLAUSE. If the user would like to specify the same search path in consecutive queries, he need only type .OF. SAME in the second query.

WHERE CLAUSE—.WHERE. conditional expression

In this part of the query, the user establishes the conditions a data block must satisfy to be considered a "hit" via inverted list searching. The simplest conditional expression is a relational expression. A relational expression is defined as

element relational operator value

where the operator is one of the following: .EQ., .NE., .GE., .GT., .LE., .LT.

or,

element .BET. value1, value2

Since both alphanumeric and pointer values are character strings, alphanumeric values must be surrounded by quotes to differentiate between these two data types.

Numeric values may be represented with or without the decimal point; however, exponential notation is not permitted.
Examples of legitimate relational expressions are

WEIGHT .NE. 100
AREA .BET. 18., 29.2
NAME .EQ. "JOHN SMITH"

More complex conditional expressions may be formed by joining up to five relational expressions with the Boolean operators .AND. and .OR. As in FORTRAN, the natural precedence of .AND. first, .OR. second, may be altered by parenthesizing parts of the expression. Finally the Boolean operator .NOT. may be used to form the complement of all or part of the conditional expression.
For example,

(a) WEIGHT .LT. 20 .AND. .NOT.(NAME .EQ. "SMITH" .OR. NAME .EQ. "JONES")
(b) .NOT.(A.EQ.10 .AND. XPTR .NE. BLOCK12)

```
QUERY?
PRINT NAME .WHERE. YEARB .LT. 1755;

SURNAME         ADAMS
FIRSTNAM        JOHN
INITIAL

SURNAME         JEFFERSON
FIRSTNAM        THOMAS
INITIAL

SURNAME         MADISON
FIRSTNAM        JAMES
INITIAL

SURNAME         WASHINGTON
FIRSTNAM        GEORGE
INITIAL
PROCESSING COMPLETE
QUERY?

PRINT SURNAME,YEARB,PARTY .WHERE. YEARB .LT. 1755 .AND.
  STATEB .EQ. "VIRGINIA";

SURNAME         JEFFERSON
YEARB                1743
PARTY           DEM-REP

SURNAME         MADISON
YEARB                1751
PARTY           DEM-REP

SURNAME         WASHINGTON
YEARB                1732
PARTY           FEDERALIST
PROCESSING COMPLETE
QUERY?

PRINT WIFE , YEARM .WHERE. LAST;

WIFE            MARTHA
YEARM                1773

WIFE            DOLLEY
YEARM                1794

WIFE            MARTHA
YEARM                1759
PROCESSING COMPLETE
```

```
QUERY?

PRINT CAPITAL,POPUL .OF. TEXAS;

CAPITAL         AUSTIN
POPUL             10972000
PROCESSING COMPLETE
QUERY?

PRINT NEWSTATE .OF. POLK/ADMIN;

NEWSTATE        TEXAS
NEWSTATE        IOWA
NEWSTATE        WISCONSIN
PROCESSING COMPLETE
QUERY?

PRINT CAPITAL .OF. E1932/PRESPTR/STATEPTR;

CAPITAL         ALBANY
PROCESSING COMPLETE
QUERY?

PRINT CITIES .OF. SAME .WHERE. CITYPOP .GT. 300000;

CITY            NEW YORK
CITYPOP            7781984
CITY            BUFFALO
CITYPOP             532759
CITY            ROCHESTER
CITYPOP             318611
PROCESSING COMPLETE
QUERY?

EXIT;

END OF PROGRAM
```

Figure 5—Sample queries

Just as .OF. SAME specifies a repeated search path, .WHERE. SAME specifies a repeat of the previous conditional expression. Not only does this feature save time on query input, but processing time is reduced in that the "hit" list from the last inverted list search has been saved.

Figure 5 presents a variety of queries on the Presidents Data Base thereby illustrating the range of capabilities of the COMRADE Query Processor.

SUMMARY

A successful computer-aided design system is a mixture of diverse design tasks, administrative tasks, and people. Every task and person possesses certain data management requirements. Application programmers require efficient dynamic data base access from their design programs. The data administrator requires a facility for describing an arbitrary data and file structure suited to his design. Project leaders and design engineers require a natural means of communication between themselves and their data bases.

The COMRADE Data Management System was developed with the idea that these requirements can in no way be compromised. Unless all personnel using the data base can do so comfortably and efficiently, the design effort will suffer. Therefore, COMRADE has attempted to provide a data management system which offers a wide variety of capabilities to be used in a wide variety of operational modes by a wide variety of users.

REFERENCES

1. Tinker, R., 1973 Avrunin, L., "The COMRADE Executive System," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.
2. Thompson, B., "Plex Data Structure for Integrated Ship Design," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.
3. Bandurski, A., "COMRADE Date Management System—Storage and Retrieval Techniques," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.

Plex data structure for integrated ship design

by BERNARD M. THOMSON*

Naval Ship Research and Development Center
Bethesda, Maryland

INTRODUCTION

The true computer-aided integrated design system must comprise more than a collection of related design programs; one essential element in such a system is a data system capable of passing mutual data among programs with minimum effort required of the user. A large and diverse integrated design system places numerous requirements on its central data file. This paper discusses these requirements as encountered in developing the Integrated Ship Design System, and describes the structure of the Ship Design File which has evolved to satisfy these requirements.

The Integrated Ship Design System (ISDS)[1] is a collection of batch, interactive, and graphics applications programs in various engineering disciplines, coupled together through a common executive system[2] and a shared, central repository of data called the Ship Design File (SDF). The SDF is a digital description of the ship and its components, complete to the level of detail required to support and document the preliminary phase of ship design for the U. S. Navy. The SDF solves the data communications problem among previously independent design programs by providing a single, common, current compendium of design information. The SDF provides input to, and receives output from, applications programs dealing with concept feasibility, surface definition of the hull form, powering requirements, electronic and mechanical system design, space and weight summaries, and numerous other ship design problems. Data entities include hierarchies of shipboard hardware systems, subsystems and components; surface definitions of decks, bulkheads and the ship's hull itself; hierarchies of space, weight, and cost classifications; and a hierarchical spatial breakdown of the ship into its compartments.

The various design disciplines view the ship from different perspectives, and require common data to be organized in a variety of different ways. The SDF employs a plex structure** which relies upon sundry types of pointers to connect data blocks in the various relationships required by the user engineers and the applications programs.

The SDF is required to be responsive to the associative data requirements of interactive graphics programs and to inquisitive perusal by engineers, as well as to the support of numerous number-crunching analysis programs.

REQUIREMENTS ON SHIP DESIGN FILE

The task of designing the Ship Design File (SDF) began with an analysis of summary documentation presently used to record the results of Navy preliminary designs. This documentation typically contains a number of arrangements drawings, schematic drawings of weapons and electronics systems, equipment lists, and the results of various engineering analyses. The SDF must contain data to duplicate this documentation, and must also include any other intermediate data derived and used in the course of the preliminary design. This aggregate of data can effectively be classified in three parts:

- Catalog Data: Any data which are not dependent upon a particular ship design are catalog data. Such data include physical properties of off-the-shelf equipment, structural properties of steel shapes, and other standard design data.
- Results of Analyses: Various engineering analyses, such as computing the hydrostatic properties of the hull and estimating propulsion requirements, result in specific numeric "answers" which are parametric measures of the ship's performance. Values for the same parameters, more or less, are derived for all ships, and these parameters may be assigned fixed "homes" in the SDF and referenced on all ships by the same standard data element names.
- Physical Ship Data: The physical ship must be defined in terms of its hull shape, its decks and bulkheads, the compartments into which it is divided, and the equipment with which it is outfitted. This third type of data presents the most problems in structuring the SDF, and it alone is the subject of the remainder of this paper.

Analysis of the physical ship data determined that it was possible to represent all of it using a relatively few—

* The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Navy.

** A plex structure is the most general form of data structure in which any given node may be related to any other.

perhaps a dozen—types of data entities. Examples of these entities are:

- Systems, subsystems or components of shipboard equipment, e.g., weapons systems, propulsion systems, piping systems
- Decks
- Bulkheads
- Weight classification groups
- Compartments (rooms) on the ship

The most significant characteristic of the SDF is that most data elements have several distinct relationships to other data elements. For instance, a piece of electronic equipment may belong to a sonar system, may be located in the Sonar Control Room, may be classified in weight group 412, and may be physically connected to various other components in the sonar system and in the electrical distribution system, the water cooling system, and the fire control system. The data structure must allow the electronic component to be referenced via any of the above relationships. This requirement reflects the inclination of various disciplines to view the ship from different perspectives, and it dictates a high degree of interconnectivity and flexibility within the SDF.

This requirement for multi-relationship access of data, along with the high volatility and large size of SDF, demand a random access file. Reference 3 describes the COMRADE Data Management System (CDMS) which has been developed as the software to access and maintain the SDF.

DESCRIPTION OF FILE STRUCTURE

Early experience with a list structured Ship Design File revealed that such a structure was too restrictive for the requirements of ISDS. Subsequent study indicated that other specialized data structures such as rings and associative structures were also too inflexible, and it was accepted that a very general plex structure was required. Furthermore, it was decided that the optimum file structure should utilize many small data blocks, each containing a few elements of attribute data (e.g., the name, area, and volume of a compartment) and a number of pointers to other data blocks which constitute the logical interblock relationships. Thus, the basic unit of data is the CDMS data block, and each block consists of a number of named data elements which are the particular attributes and pointers of the block. Calls to CDMS routines enable the Fortran programs to retrieve or update a single element or a group of elements at one time.

A CDMS "block type" is defined for each entity required to represent physical ship data. Examples of block types are Arrangement Blocks (representing compartments) and Equipment Blocks. Each block type has a unique set of attribute elements and pointer elements defined for it.

The remainder of this paper describes a few of the representative block types and inter-block relationships which were defined and assembled to produce the ISDS Ship Design File.

The SDF is really a plex structured file in which any block may be defined to point to any other block. For ease of comprehension, however, it is best represented as in Figure 1 as a cross-connected tree structure. Each major branch of the tree represents one form of organizational hierarchy, and most branches consist of data blocks of a single block type. Pointers in each block contain the relationship information to represent the hierarchical branches and to relate each block to appropriate blocks in other branches.

Ship systems branch

Blocks in the Ship Systems Branch are referred to as equipment blocks. Each block at the bottom of the branch represents one "component" of equipment—a pump, a radar console, a table, a missile launcher, etc. The components are organized hierarchically into subsystems, systems and groups of systems. There is no predetermined number of levels in the equipment hierarchy; the cognizant engineers are free to structure this branch in as many levels as are convenient to represent the complexity of the various systems. We would expect more levels in the systems branch of an aircraft carrier, for example, than would be used for a shipyard tug.

Figure 2 is the Equipment Block format. The elements are grouped into two sub blocks—one for attribute data, another for pointers. Attribute data include the alphanumeric name of the equipment, the X, Y, and Z location of the equipment in the ship coordinate system, and three rotation angles to orient the equipment relative to the ship. Note that no physical attributes of the equipment itself are represented. These data reside in a block on an equipment catalog which may be accessed by following

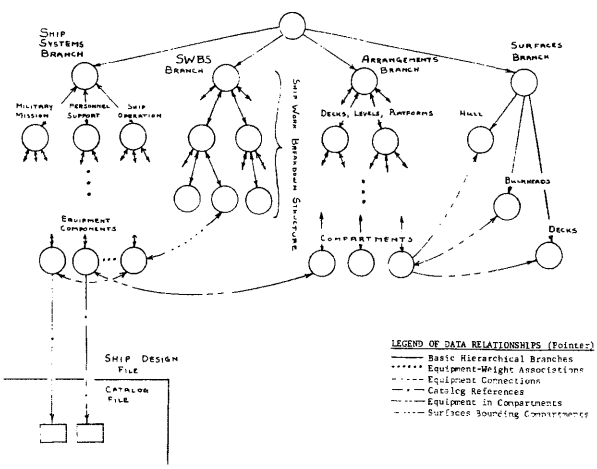


Figure 1—Partial overview of ship design file structure

the CL pointer* which is part of the equipment block. This scheme consolidates catalog data for ease of maintenance and eliminates the duplication which would occur when a particular item of equipment appears many times on a ship.

Other pointers represent relationships with other blocks on the SDF: The CM pointer* indicates the block of the compartment in which the equipment is located. The SW pointer* relates to a block in the Ship Work Breakdown Structure (SWBS) Branch, used for cost and weight accounting. The PR pointer is the parent pointer to the equipment block immediately above in the Equipment Branch, and the element EQ is a repeating group of pointers to equipment blocks immediately below. The NOTEPTR and CONNPTR pointers are used for association with auxiliary blocks containing, respectively, alphanumeric notes respecting the equipment, and data defining the physical connections (piping, wiring, etc.) with other equipment components.

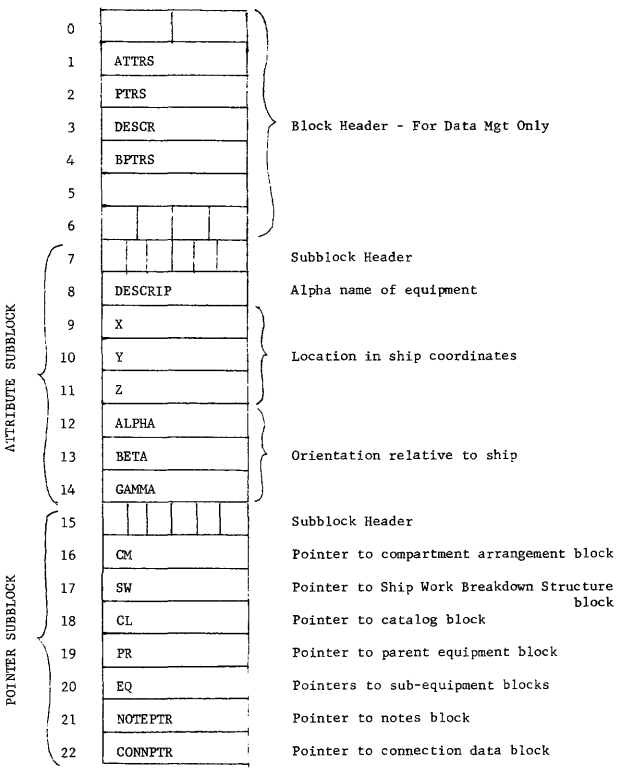


Figure 2—Equipment block format

Figure 3 shows a portion of the Ship Systems Branch in more detail, illustrating the relationship between equipmen blocks, catalog blocks, and notes blocks.

Ship work breakdown structure branch

The total weight, center of gravity, and moments of inertia of the finished ship determine her static displacement, trim and heel, and the dynamic characteristics affecting her maneuverability and motions in a seaway. A traditionally basic aspect of naval architecture is the meticulous accounting of the weight and location of each piece of material which constitutes the ship. All weight items on a ship are classified according to the Ship Work Breakdown Structure (SWBS) for the purposes of weight budgeting and accounting. The same SWBS organization is used for cost accounting and for cost and man-hour estimating.

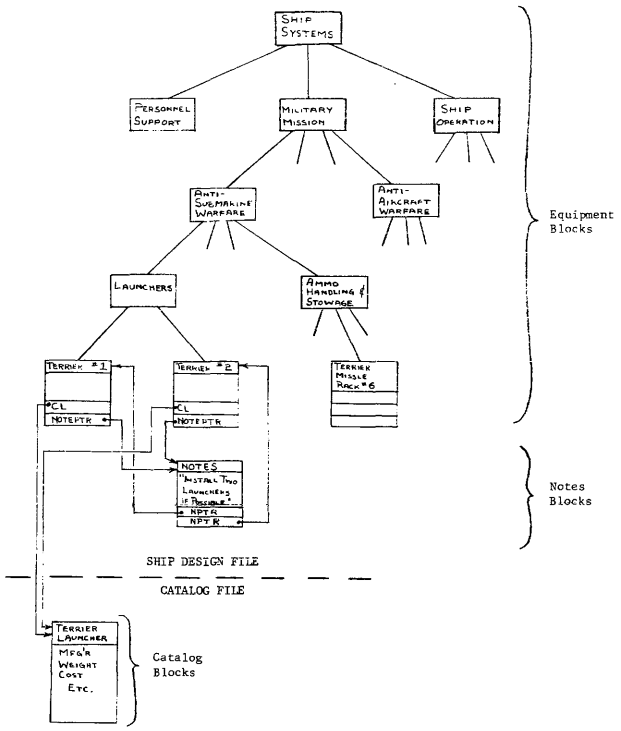


Figure 3—Data structure in ship design branch

Figure 4 is a portion of the SWBS classification, which extends for three and sometimes four levels of hierarchy. This standard hierarchy will automatically form the upper levels of the SWBS Branch of the SDF, and engineers will be able to expand the SWBS Branch through

```
Group 300     Electric Plant
      •
      •
      •
      310       Electric Power Generation
      311         Ships Service Power Generation
      311.1         Ship Service Generators
      311.2         Ship Service/Emergency (Diesel)
      311.3         Ship Service/Emergency (Gas Turbine)
      312         Emergency Generators
      314         Power Conversion Systems
```

Figure 4—Portion of ship work breakdown structure

* See Figure 2

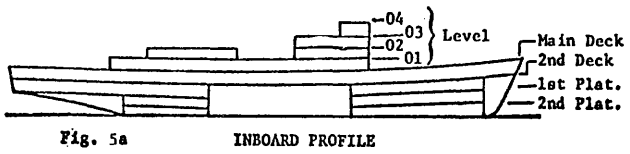


Fig. 5a INBOARD PROFILE

Note parabolic sheer on 01 Level, Main Deck, and 2nd Deck; straight line sheer elsewhere.

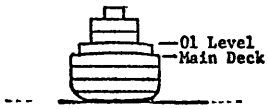


Fig. 5b SECTION VIEW

Note camber on Main Deck and 01 Level

Figure 5—Illustration of sheer and camber

any of the terminal third or fourth level SWBS blocks by defining additional SWBS blocks if such would be useful.

Each SWBS block may contain estimates of weight and cost which have been made at its level of detail, and summaries of estimated or computed weight/cost data from SWBS blocks below it. The lowest level SWBS blocks contain pointers to equipment blocks which fall into their respective accountability. These pointers indicate the equipment block highest on the Equipment Branch such that all equipment blocks under it belong to the same SWBS group; it is thus not necessary to have pointers to every component of a system which is entirely accounted for by one SWBS block.

Other SWBS data elements include the longitudinal and vertical centers of gravity relative to ship coordinates, up-and-down pointers to form the hierarchy of the SWBS branch, and a pointer to a NOTES block containing relevant alphanumeric comments.

Surfaces branch

The various blocks of the Surfaces Branch contain the descriptions of the physical surfaces which bound and subdivide the ship—the hull envelope, the decks and platforms, the bulkheads and numerous small partitions which segment the ship into its many compartments.

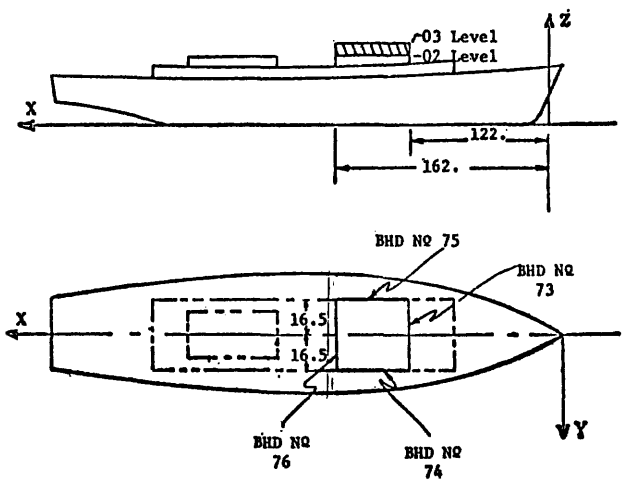
The ship hull, i.e., the bottom and sides of the external watertight skin, is a complex three-dimensional shape. It is defined by specifying the coordinates of points on the hull lying in regularly spaced transverse planes, or stations. Arbitrary points on the hull may be obtained by double interpolation.

Decks are surfaces which are horizontal or nearly horizontal. Decks are further classified as "levels" (decks in the superstructure), "platforms" (decks in the main hull which are not longitudinally continuous), and continuous decks (See Figure 5a). In general, decks have curvature in two directions. Sheer is the longitudinal curvature (Figure 5a) and camber is the transverse curvature (Figure 5b). Sheer and camber are mutually independent and each is defined analytically; therefore, the deck surfaces are analytically defined.

The bulkheads of a ship are analogous to a building's interior walls. Nonstructural bulkheads may be termed partitions. ISDS requires bulkheads to be flat and vertical. Most bulkheads are oriented either longitudinally or transversely, but special definition also allows "general" bulkheads which are oblique to the centerline axis of the ship.

Bulkheads are defined as finite planes; a key designates the bulkhead as longitudinal or transverse, a single dimension locates the plane, and four pointers indicate two decks and two bulkheads (or hull surfaces) which bound the plane (Figure 6). It is significant that the edges and corners of bulkheads are not directly specified but are defined indirectly through pointers to bounding surfaces. This scheme greatly simplifies the updating of changes in bulkhead definition.

Numerous data are associated with each bulkhead. In addition to the elements mentioned above, other data elements designate whether the bulkhead is watertight, oiltight or non-tight, structural or non-structural, and pointers are included to all compartments bounded by the bulkhead. Data to be added in the future could include



The Data Structure Below Defines The Four Partition Bulkheads Shown Above.

BHD NO	73	74	75	76
Orientation	TRANSV.	LONG'L	LONG'L	TRANSV.
Location	122.	16.5	-16.5	162.
Bounding BHD's	74	73	73	74
	75	76	76	75
Deck Above	03 Level	03 Level	03 Level	03 Level
Deck Below	02 Level	02 Level	02 Level	02 Level

Figure 6—Data structure for partition bulkheads

access openings, penetrations, structural details, and weight data. The bulkhead is thus an important physical entity, but to the problem of shipboard arrangements it is important primarily as a space-bounding surface.

Arrangements branch

The structure of the arrangements branch enables the designer to think of the ship as a deck-oriented tree structure of spaces.[4] One such tree structure is shown in Figure 7. The first subdivision below "SHIP" always represents the platforms, levels or complete decks of the ship. Each deck, level, and platform is then further subdivided, as directed by the designer, into progressively smaller units. The smallest subdivision is the actual compartment on the ship. Each subdivision, large or small, is represented by an arrangement block and corresponds to one node of Figure 7. An arrangement may comprise a deck, a level, a platform, a superstructure house, a segment of a platform, a space, a group of spaces, or any contiguous designer-designated portion of one deck (or level or platform) of a ship.

Data elements for each arrangement block include:

- deck area, volume, and the name of the arrangement

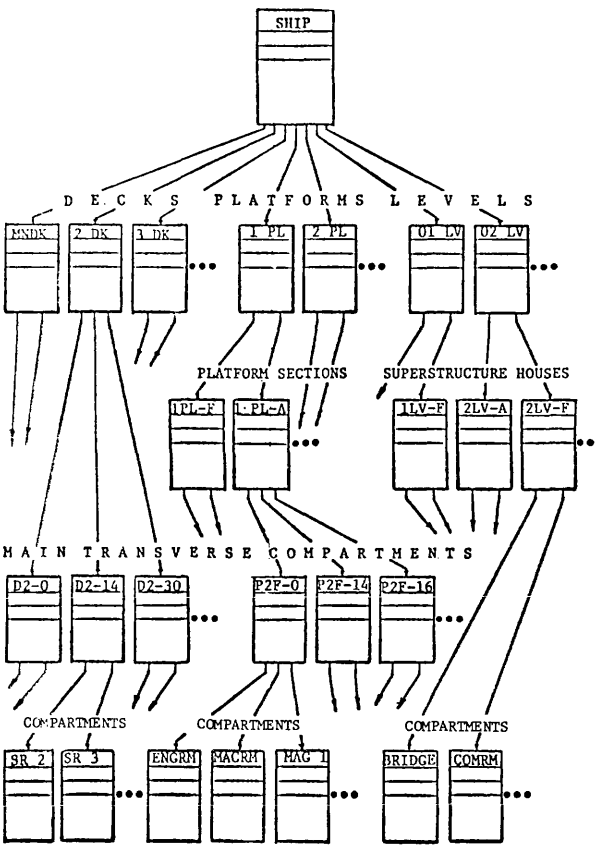


Figure 7—Typical portion of arrangement branch

- pointers to bulkheads or hull surfaces forming the perimeter of the arrangement
- pointers to the decks above and below
- pointers to its component arrangement blocks and a backpointer to its parent arrangement

The "undefined attribute" capability of CDMS[3] provides means for storing additional attributes for specific blocks as needed. Those low level arrangement blocks representing compartments on the ship could contain summaries of component weight and requirements for electrical power, ventilation, or cooling water.

The reader will realize that there is a rigid logical interdependence between the subdividing surfaces and the subdivided spaces in a ship, building, or similar entity. The data structure chosen for the Ship Design File has been designed to minimize the chance of allowing contradictory surface/arrangement data to occur.

Typical accesses of ship design file

An attempt was made early in the development of ISDS to use COMRADE's inverted file capability to manage the physical ship data of the Ship Design File (SDF). The inverted file structure works well on the equipment catalog files, for which it was developed, but it was soon discovered that there is a basic difference between a typical query of a catalog file and the kind of accesses characteristic of the physical ship data. In a catalog query the user asks the program to identify components which possess certain prescribed attributes. For example, he may ask for the names and manufacturers of all pumps capable of a flow of 1500 gallons per minute. The query processor must locate unknown data blocks based upon known attributes. This requires that extensive cross-reference files be maintained on each attribute to be queried.

The logic involved in a typical access to physical ship data is quite different from that of an inverted file query. In this case we start with a known block in the SDF and follow particular pointers to retrieve blocks akin to the known block by a relationship which is also known. Some examples of this type of query are:

- What are the bounding bulkheads and components of a particular compartment? This question is common in graphics arrangement programming.[4,5]
- In which compartment is this component located?
- In which compartments are components of this system located?
- What are the compartments on this deck?
- What weight groups are represented by this system?
- List the cost of components in all compartments on this deck.

The reader will note that some of the above retrievals require the processor to follow several sets of relational pointers. The last example requires the following logic:

1. Start with the prescribed deck. Follow the CM pointer to the arrangement block for that deck.
2. From arrangement block, follow CM pointers to next level arrangement blocks. Repeat down to bottom of Arrangement Branch; lowest blocks represent compartments.
3. From compartment arrangement blocks, follow EQ pointers to all equipment blocks within each compartment.
4. From each equipment block, follow CL pointer to catalog block for that component. Retrieve cost of component and print with name of component.

The COMRADE Data Management System[3] has developed a "pointer chasing" algorithm which processes this type of query. The user must specify the start block, the sequence of the pointers to be followed, and the data to be retrieved at the end of the search.

The pointer-chasing retrieval routines can be called from Fortran programs as well as by a teletype user. It will be a straightforward step to build simple summary or computational algorithms around the pointer-chasing routines. Typical of this type of program will be one for performing a weight-moment summary for the whole SWBS Branch.

CONCLUSION

The development of the ISDS Ship Design File is principally notable for the following points, most of which will apply to other integrated design systems:

1. The basic problem in data structure for integrated design was defined as the requirement for multi-relational access of common data.
2. A clear differentiation was made between equipment catalog data and ship dependent data.
3. A plex data structure was designed in response to the above requirements, which is modelled for convenience as a cross-connected tree structure. It features small blocks of attribute data connected by many relational pointers.
4. The data structure is a logical model of the physical ship, whose principal entities are surfaces, spaces bounded by those surfaces, and items of equipment in the spaces. This logical structure directly serves graphic arrangement routines, and preserves the arrangement data in digital form for use by number-crunching analysis programs. Most of this model is directly applicable to architectural design of buildings, and part of it to the design of aircraft and other vehicles.
5. A "pointer-chasing" query capability was developed to facilitate use of a data base dominated by inter-block relationships.

REFERENCES

1. Brainin, J., "Use of COMRADE in Engineering Design," presented at 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.
2. Tinker, R., Avrunin, L., "The COMRADE Executive System" presented at 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.
3. Willner, S., Gorham, W., Wallace, M., Bandurski, A., "The COMRADE Data Management System," presented at 1973 *National Computer Conference*, New York, June 1973, American Federation of Information Processing Societies.
4. Chen, R., Skall, M., and Thomson, B., "Integrated Ship Design by Interactive Graphics (ISDIG)," *Proceedings of SHARE* XXXVI, 1971.
5. *Operators'/Users' Manual* for the Computer Oriented Graphics Arrangement Program (COGAP), prepared by Lockheed-Georgia Company for the Naval Ship Engineering Center, Washington, D.C., 1972.

COMRADE data management system storage and retrieval techniques

by ANN ELLIS BANDURSKI and MICHAEL A. WALLACE *

Naval Ship Research and Development Center
Bethesda, Maryland

INTRODUCTION

The design of a data management software system involves the consolidation of a balanced set of individual techniques. Today's third generation computers provide the resources for the storage of large, complex sets of data, and for the rapid retrieval of that data. Random-access mass storage devices will store millions of bits of information, and central processors have instruction execution times on the scale of tenths of microseconds. But central memory space is limited and mass storage access time is not negligible.

The COMRADE Data Management System was designed as a tool under the Computer-Aided Design Environment (COMRADE) software system[1] to manage the large quantities of data involved in the design of a complex system. The computer-aided design environment has characteristics which place demands upon a data management system and the ways in which it utilizes computer resources.[2]

The computer-aided design environment is characterized, first, by an immense and volatile set of data: data brought into the design process, data sorted, calculated, and communicated during the design process, and data which constitute the end product of the design process. Another characteristic of the computer-aided design environment is the wide range of disciplines represented by the individuals who are involved in the design process and make use of the design data.

In response to these characteristics, the COMRADE Data Management System (CDMS) focuses not only upon furnishing efficient means for the storage of large quantities of data, but also upon making facilities available through which data are readily accessible to the non-programming designer. Rather than present an extensive package which attempts to be everything to everyone, CDMS provides a three-part data management capability. The flexibility allowed by this approach permits the individual using the system to make his own trade-off decisions between ease of use and efficiency.

The three parts of the CDMS package are built one upon the other. The lowest-level component deals with the interface between the data and the computer environment. The package at this level provides a compact and extremely efficient capability for the dynamic storage and retrieval of variable-length data blocks. The subroutines in this portion of the package may be used directly by the programmer who seeks economy above all else, but they also constitute the foundation for the higher-level components.

The package at the second level[3] provides a mechanism whereby sets of data can be organized within storage blocks; block types can be defined and names can be given to data elements within the block types. This facility allows "pointer" as well as attribute data to be defined so that data values within blocks may contribute a logical structure to the data base. A sub-routine library is provided at this level through which these features are made available to the programmer.

The third component of the system[3] is provided to make the system most usable to the designer who may not be a programmer. This level provides a set of interactive command procedures. Using the system at this level, the designer can sit at a remote terminal and interact with the data base directly as he develops a design. Also included at this level are programs to initiate as well as execute data management functions.

Although a user may access the data base through any of these three levels, it is the low-level component which actually stores and retrieves data. This low-level component is known as the COMRADE Data Storage Facility (CDSF). The CDSF handles all of the underlying data base management functions, including file usage and inverted list processing as well as data block storage and retrieval.

The CDSF consists of a set of FORTRAN-callable subroutines, most of which are written in FORTRAN. This component, as part of the COMRADE system, is operational on a CDC-6700 computer under the SCOPE 3.3 Operating System.

The configuration through which the CDSF utilizes computer resources ensures two things to the users of COMRADE data management at all levels. First, it ensures that there will be a minimum of restrictions on

* The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Navy.

the size and uses of a data base. It provides for the dynamic allocation of variable-length data blocks up to 2^{16} -1 words each, and will handle up to 516,000 data blocks in an unstructured format on each data base file.

Secondly, it ensures the efficient operation of CDMS. It lays the foundation for the rapid response which is vital to the designer working at the teletype and accessing design data. It also minimizes both core and disk space requirements by strict management of these spaces.

There are three basic elements of CDSF:

- Data block handling routines furnish the operations necessary for the dynamic storage, retrieval, update and deletion of data blocks on disk files. These routines maintain a two-level directory on each file whereby any data block may be accessed directly through its block name. To conserve disk space, a reallocation scheme is included; and a circular buffer is managed which will allow more than one data block to be in main memory at a time. The mechanisms involved in data block handling will be described in more detail shortly.
- Inverted list processing routines maintain ordered lists of the values of certain data elements and the names of the data blocks in which those values occur. These lists allow data blocks to be referenced on the basis of data values they contain without requiring a linear search of all blocks on a file. The methods used for inverted list creation, update and retrieval are presented later.
- File use routines allow multiple random-access disk files to be established for data base use under CDMS. With these routines the programmer may transfer between files within one program. They lay the foundation for processing the cross file data pointers which are defined and used through higher-level CDMS components.

When a file is established as part of a CDMS data base, the file use routines issue a request to the operating system for a file on a permanent disk file device. Space is allocated on this file for CDSF file management tables which are maintained on the file throughout its existence. The file use routines open a file to be used and prepare its I/O interface with the operating system. They also rewrite updated file management tables onto a file when it is closed.

Now, details of the techniques utilized in the handling of data blocks and the processing of inverted lists will be presented.

DATA BLOCK HANDLING

The storage and access of the multitude of data blocks which are needed in a design data base are managed by CDSF. When a data block is stored, it is given a block name. CDSF keeps a directory of all the names of data blocks on a file and the disk addresses where those blocks may be found on the file. This makes it possible for a symbolic name rather than a numerical index to be used to access a data block during its residence on the file.

CDSF provides all of the basic data management functions to handle variable-length data blocks, while allowing them to be referenced by name. A data block may be stored on any file which has been established for data base use. All or portions of a data block's contents may be retrieved. Modification of the contents of a data block is permitted, including that which requires increasing or decreasing the size of a block. Finally, removal of a data block from a file may be accomplished.

The access of data blocks by name is provided through a two-level directory which is maintained on each file. The first level or main directory is brought into main memory when a file is opened and remains there throughout the time the file is in use. The second level of this directory consists of fixed-length subdirectories which are created on the file as they are needed. Only one subdirectory is brought into core at a time to be used. The main directory contains the addresses of the subdirectories on that file. It is in the subdirectories that the disk addresses of all data blocks on the file are stored. Through the use of this expandable two-level directory, up to 516,000 data blocks can be stored on a file. Since the main directory is brought into main memory when the first data block on a file is accessed, all blocks which are subsequently referenced on that file require only two disk retrievals (one to get the subdirectory and one to get the data block).

When access to a data block is requested, its block name (48 bits; 8 characters) and its version number (a 5-bit number from 0 to 31) are specified. This block name/version number pair is put through a scatter function which transforms such pairs into uniformly distributed values. Binary digits (bits) are extracted from the resultant value to form a "key". This key is used as an index into the main directory where the address of the appropriate subdirectory is found. The key is then used to index the subdirectory to locate the address of the data block (see Figure 1).

It is generally found that block names which are generated by one person have a high degree of correlation. To scatter the indexes into the directory, a multiplicative

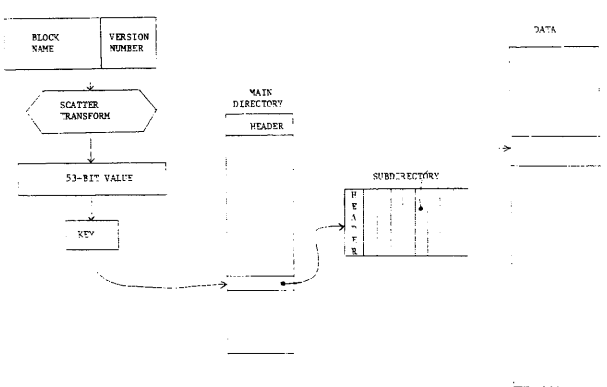


Figure 1—Access of data blocks by name

congruential transform was chosen to apply to the block name/version number pairs.

The block name and version number are concatenated and the resultant 53-bit value is multiplied by a 53-bit transform. The result of the multiplication (modulo 2^{53}) has bits extracted from it to produce the key into the main directory.

When the first data blocks are stored on a file, one subdirectory contains the entries for all blocks. These entries are divided into two groups: each entry is placed in either the upper or the lower half of the subdirectory, according to the value of the first bit in its key. When there is only one subdirectory, there is only one address in the main directory and it is not necessary to use any bits from the index to find the address of the appropriate subdirectory.

When one of the halves of the subdirectory becomes filled, a new subdirectory is created into which the entries from the filled half are moved. Each of these entries is placed in the new subdirectory's upper or lower half according to the value of the second bit in its key. Now two subdirectory addresses will appear in the main directory. The first bit in a key must be used to determine which of these addresses refers to the appropriate subdirectory.

The length of the main directory is always a power of two so that whenever it must expand to accommodate new subdirectory addresses, it simply doubles in size. When the directory doubles, the addresses already in the directory are spread throughout its new length by placing each address in two contiguous locations. The address of the new subdirectory then replaces half of one of these address pairs.

Each time the directory doubles, an additional bit must be used from the key to find the appropriate subdirectory. Correspondingly, each time half of a subdirectory splits out to form a new subdirectory, the half where an entry is placed in the new subdirectory is determined by the bit in the key following the one used to determine entry locations in the previous subdirectory. Figure 2 illustrates the process by which the directory expands.

The entries are placed in the subdirectories sequentially from each end towards the middle. Additionally, the entries in each half of a subdirectory are chained together to form four lists. The two bits of the key following the bit which determines the half of the subdirectory are used to determine which of these four lists an entry is chained onto.

In order to quickly locate a data block entry within a subdirectory, each subdirectory has a header which gives the address of the first entry on each of the four lists in each of its halves. This header also indicates which bit in a key should be used to determine the subdirectory half for a particular entry. It also points to the next empty location in the upper and lower halves of the subdirectory in which an entry may be placed. Figure 3 shows the arrangement of entries in a subdirectory and the contents of the header.

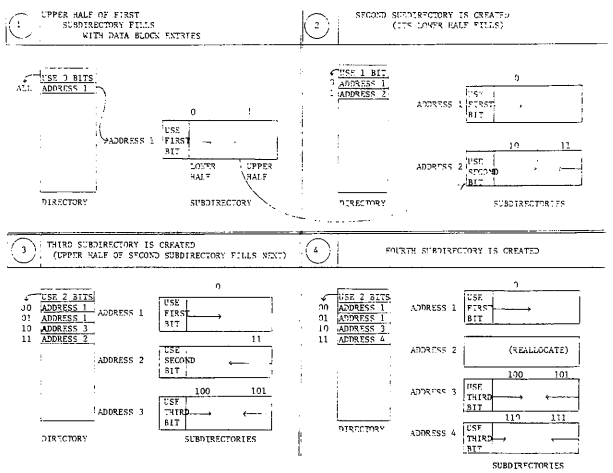


Figure 2—Directory expansion

An entry in a subdirectory for a data block contains the transformed block name/version number, the disk address where the block is stored, the length of the data block and the amount of disk space in which the block is stored.

Disk space reallocation

When a data block is stored on a file, it is usually written directly at the end of information on the file using the least amount of disk space which will contain the block. When a data block is removed from a file, the allocated disk space in which it was stored may be used for another data block. A disk space reallocation table is maintained on each file to keep track of blocks of disk space which are available for reuse. This table consists of an index and up to 72 lists, each of which holds the addresses of up to 503 reallocatable disk space blocks in a particular range. The index for the reallocation lists is brought into main memory when the file is opened.

An entry is made in one of the reallocation lists each time a data block is removed from the file. The exact size of the reallocatable space is kept in the list along with its address. An entry is made in the index if the block to be reallocated is the first one in a particular size range and a new list is created for it.

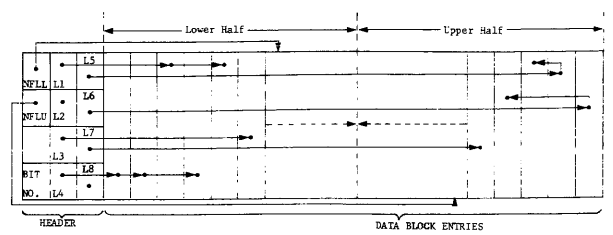


Figure 3—Subdirectory format example

The reallocation table is referenced each time a new block is to be stored on the file and each time a larger block of disk space is needed to store a data block whose size is being increased. The entry in the name-access subdirectory for each data block retains the exact size of the disk block in which the data is placed, so that no disk space is lost if the block is again reallocated.

Circular buffer

Although data blocks may be created that are up to 2^{12} -1 words in size, it is usually not desirable to use an enormous amount of main memory space to transmit a data block to the disk file. In order to be able to handle data blocks of almost any size, the CDSF uses a circular buffer for I/O whose size is defined by the programmer. When the retrieval of a large data block is requested, the circular buffer allows one portion of the block to be brought into main memory at a time.

The circular buffer will also retain portions of data blocks or entire data blocks until the space which they occupy in the buffer is needed for other data. This multi-block capability operates on a first-in, first-out basis.

Because of this feature, it may not be necessary to access a data block through the directory each time it is requested. The contents of the circular buffer are checked for the desired block. and if part of that block is in main memory (in the buffer), the need to read and search a subdirectory and possibly the need to read the data block is obviated.

INVERTED LIST PROCESSING

When a data base file is created under COMRADE, retrieval by block name is the most efficient type of retrieval because the file is set up with a block name directory. If the user wishes to retrieve information from the file on the basis of the values of data elements within data blocks, the file may be set up to include inverted lists to make this type of retrieval efficient also.

An inverted list is a list of all of the values for one data element which occur in data blocks throughout the file, with information for each value indicating where in a data block it may be found. Such a list is ordered on the basis of the data values so that the entries for one value or a range of values may be quickly located.

When the higher-level CDMS components are used to store and request data, the inverted lists will be automatically created and referenced as follows: When a data block is stored which contains a data value for an inverted element, the value is also placed in the inverted list for that element; when a query requests information on those data blocks in which particular values of inverted elements occur, the inverted lists are searched and the desired information retrieved.

The CDSF is responsible for building and maintaining the inverted lists, and for searching them to retrieve information to satisfy query requests.

Inverted list storage

At the time a file is defined under CDMS, one of the tables for which disk space is allocated is an inverted element directory. This directory is brought into main memory each time the file is opened. Once values have been stored on the file for an inverted element, this directory will contain the name of the element and the disk address of the first block of its inverted list.

As an inverted list grows, it may require many disk storage blocks. Each of these blocks may contain up to 340 value entries which are ordered within that block. The inverted list storage block for an element whose values occur in only a few data blocks may be accessed directly from the inverted list index. When the first storage block for an element is filled, it becomes an index for up to 510 additional storage blocks. Each time another storage block becomes filled, the entries which it contains are divided equally between itself and a new block. A new entry is made in the index to reflect the existence of the new block and where it occurs in the set of ordered blocks. Even though there may be storage blocks for an element, only the directory and one storage block need be accessed to locate a particular value. Figure 4 shows the relationships of the different types of inverted list blocks. In Figure 5, the format for an inverted list index is shown, and in Figure 6, the inverted value storage block format is given.

The addresses of inverted list storage blocks which have been emptied by the deletion of values are kept for reallocation within the inverted list index.

Inverted list update procedure

An entry must be placed in an inverted list so that its value falls between immediately greater and smaller values. Inverted list processing time may be minimized by a procedure for accumulating inverted list insertions and deletions and making many updates at one time. The entries are presented so that it is necessary to reference each inverted list block only once. The CDSF provides a two-part procedure for making bulk updates.

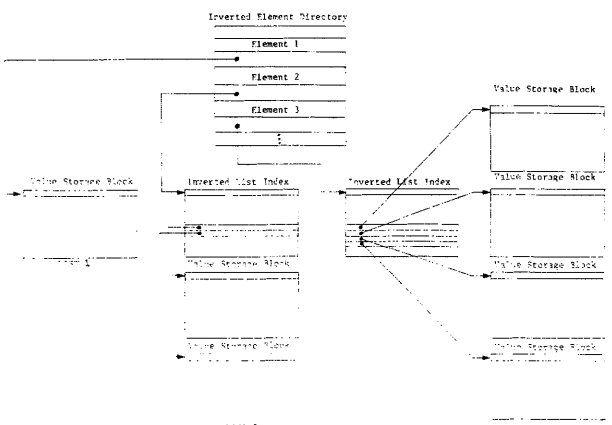


Figure 4—Inverted list structure

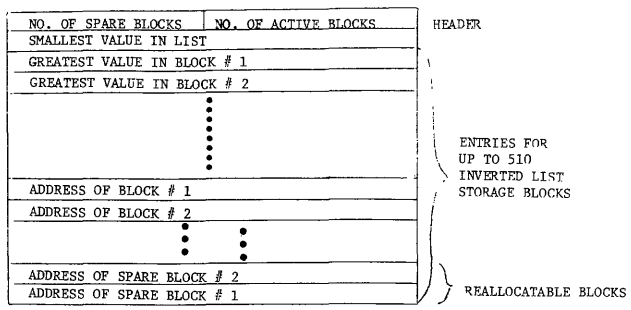


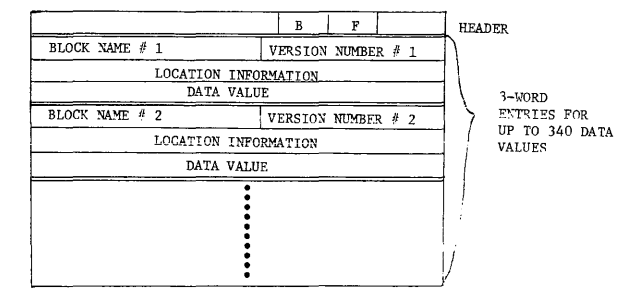
Figure 5—Inverted list index

First, a sequential list of additions and deletions to be made to all inverted lists is kept in the order that the specifications are given. The entries in this list for each element are linked together in inverse order, from the last one entered to the first. As this list grows, portions of it may be placed on a special disk file created for this purpose. The list will continue to grow until one of two things takes place: until the end of the computer run during which it is built; or until a query is made which uses the inverted lists.

Then, when one of these things occurs, the linked list of additions and deletions for one element is extracted from the total set of specifications. This list is sorted according to element value; then the value entries are integrated into the existing portions of the list. During this process it must be remembered that the sequence in which specifications are given is significant. When the specifications for one list are sorted into order on the basis of value, the original order must be maintained so that if one specification nullifies another, the correct result will remain. Figure 7 shows the two parts of the inverted list update procedure.

Inverted list information retrieval

An inverted list is consulted in order to locate within the data base the occurrence (or occurrences) of a particular value or range of values for an inverted element. The information kept in the inverted list for each value includes not only the block name/version number of the data block in which a value occurs, but specifies where within the block the value is located.



Where: B is a flag indicating that low value in this block equals high value in previous block

F is a flag indicating that high value in this block equals low value in previous block

Figure 6—Inverted list storage block

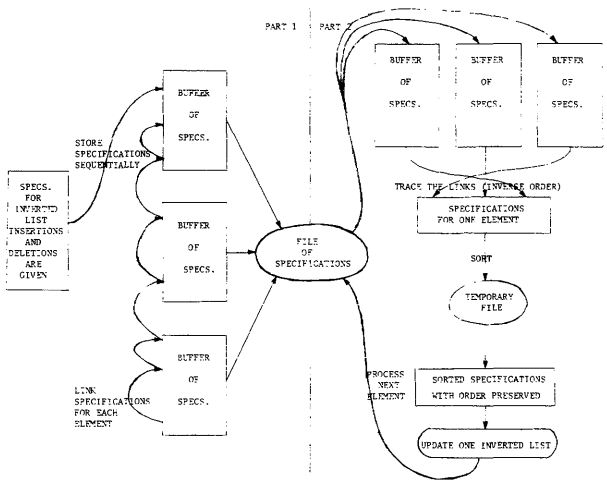


Figure 7—Inverted list update procedure

Since there may be many or few occurrences of the specified value(s) within an inverted list, a disk file is created to receive entries for up to 64,897 values. The names of the data blocks which contain these values may be picked up from this file. If further information from the data blocks is required, the individual blocks may be accessed through the directory.

CONCLUSION

We have seen some of the storage and retrieval techniques which have been developed to handle large quantities of blocked data using direct retrieval and limited core storage. Capabilities for the access of data blocks by name, for inverted list processing, and for multi-file usage, provide an efficient and flexible data management system. The higher level components of CDMS, together with this foundation, provide the user a variety of capabilities from which to manage his data efficiently and conveniently.

REFERENCES

1. Rhodes, T., "The Computer-Aided Design Environment Project (COMRADE)," presented at the *National Computer Conference,* American Federation of Information Processing Societies, New York, June 1973.
2. Yuille, I., *A System for On-Line Computer Aided Design of Ships-Prototype System and Future Possibilities,* presented to the Royal Institution of Naval Architects, London, April 1970.
3. Willner, S., Bandurski, A. E., Gorham, W., and Wallace, M., "COMRADE Data Management System," presented at the National Computer Conference, American Federation of Information Processing Societies, New York, June 1973.

COMRADE design administration system

by C. MICHAEL CHERNICK*

Naval Ship Research and Development Center
Bethesda, Maryland

INTRODUCTION

The Design Administration System of COMRADE[1] completes the set of major functions required for integrated design system support.[2,3] The purpose of the Design Administration (DA) System of COMRADE is

- To provide capabilities which allow the manager(s) of a large, computer-aided design effort to control and monitor design activity, and hence, to control and monitor use of their COMRADE subsystem.
- To provide subsystem developers and users rapid on-line file access and file access control.

To accomplish these objectives, the Design Administration effort was divided into three interrelated areas of activity:

(1) The COMRADE Permanent File Management System (CPFMS)
(2) The COMRADE Logging and Reporting System, and
(3) Support Functions for COMRADE Subsystems.

While the three areas overlap, their basic functions differ. CPFMS is concerned with the management and security of files in the design environment. The Logging and Reporting System deals with the generation of reports for managers, for software system designers, and for applications designers (the "users" of the system). Finally, a variety of support functions are provided to aid the subsystem designer.

COMRADE PERMANENT FILE MANAGEMENT SYSTEM

That part of the operating system concerned with the cataloging, retrieving, and decataloging of files (but *not* with the processing of files) is called the file management system. The file management system is a key element in any computer based system, especially in a multipurpose computer aided design system such as COMRADE, where files must be quickly and easily accessed and securely maintained if the system is to be effective.

The goals of the COMRADE Permanent File Management System (CPFMS) are to provide:

- rapid on-line access to COMRADE files,
- user-file access control, and
- file-use profiles for effective file management.

The SCOPE Version 3.3 operating system used on the Naval Ship Research and Development Center (NSRDC) CDC 6700 is a batch oriented operating system. This orientation is reflected in the SCOPE file management system which is not well suited for interactive design.

Before explaining the COMRADE permanent file system, it is helpful to discuss file processing (i.e., reading and writing files, mass storage allocation, etc.) under SCOPE. File processing under SCOPE is dynamic in that files can be created merely by issuing a system write request to a file that does not yet exist (a useful feature for interactive processing). The system automatically assigns the file to disk and allocates an initial block of disk space. As writing continues, file space is dynamically allocated as needed. The block size written on the disk is fixed by the system at 64 words (640 characters), but FORTRAN provides automatic blocking so application programmers need not be overly concerned with blocking factors. Files created in this manner are called local files.

For batch job processing, files created dynamically are ordinarily lost at the end of job. For time sharing users, files created at the terminal ordinarily remain available until logout from the system and then are lost. However, files may be cataloged such that they can be retrieved at a later time. A local file may be cataloged with a relatively simple control card. Once cataloged, a local file is then known as a permanent file. The CATALOG control card is used to specify a permanent file name (PFN). The PFN is used as unique identifier of this file in the system so that the file may be accessed later in another job by attaching it with an ATTACH control card specifying the file's PFN. The file may be decataloged or purged with a PURGE control card.

This permanent file system works reasonably well for batch jobs. However, this scheme is not altogether satisfactory for time sharing jobs. For one thing, an engineer

* The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Navy.

concerned with design does not want to worry about submitting control cards specifying unique names to save his files for later processing. Another problem is the time involved in attaching files. To attach a file to a job (or time sharing terminal) the system must search the permanent file directory. The directory entries are not hashed using the permanent file name so a search that may take 3 to 4 seconds must be made through the entire directory until the PFN is found. No computation may be done by the job doing the attaching during this search, and in fact the time sharing service to other users is often severely degraded. Thus it is essential for good service that the directory search take a minimum of time. Fortunately the directory may be divided into subdirectories and the permanent file system can be directed by a parameter on the ATTACH control card to search a specified subdirectory first. On the NSRDC system, the directory consists of 50 subdirectories and when the correct subdirectory is specified, the attach time is reduced such that the attach usually completes within a half second. This time saving is significant, but to take advantage of it the terminal user must somehow tell the permanent file system the correct subdirectory. Since a design process may require many permanent files (often created and purged within a few days) the terminal user cannot be expected to memorize subdirectory numbers.

The COMRADE Permanent File Management System (CPFMS) relieves the terminal user from the burden of submitting control cards and memorizing subdirectories to access his files. The two primary components of the system are: (1) a catalog system that stores pertinent information (including subdirectory numbers) about CPFMS files; and (2) a set of CPFMS interface routines. See Figure 1.

Additional routines used by CPFMS, and shown in Figure 1, are the permanent file utility routines which serve as a replacement for the SCOPE permanent file control cards. These routines were developed separately, and may be used independently of CPFMS or of COMRADE for that matter. They have the ability to access permanent files with up to 40-character names (including imbedded blanks and dashes), not possible through control cards. They give a calling program dynamic file management ability.

The **CPFMS Interface Routines** (CIR) are used to manage CPFMS files. The CIR use information from two sources: (1) information about CPFMS files stored in the CPFMS catalog and (2) information about a COMRADE system user stored in subsystem common.[2] File accessibility can then be controlled by the CIR based upon the information found in these two sources.

For file integrity, the CPFMS catalog system and CPFMS files must be accessible only through the CIR and CPFMS maintenance routines. File integrity is based upon the standard SCOPE permanent file system passwords, which may be assigned to permanent files as they are cataloged. The CPFMS security structure assumes that the basic SCOPE system is secure; to secure the

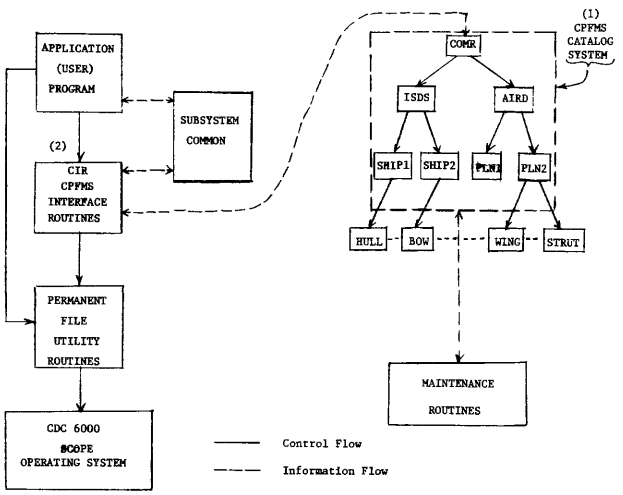


Figure 1—COMRADE permanent file management system

CPFMS system, then, we must secure the passwords used.

The passwords are kept within the CPFMS catalog system shown in Figure 2. The files that comprise the CPFMS catalog system are themselves protected with passwords. As may be seen in Figure 2, the catalog system is organized as a tree structure. The tree structure was chosen for: (1) its natural similarity to subsystem and project partitions, and (2) its ability to lend itself to quick and easy retrieval. The CPFMS files are given four level names, the first three of which are the three levels of catalog structure.

For example, in Figure 2, a file cataloged under the Integrated Ship Design Subsystem (ISDS)* of COMRADE might be named COMR-ISDS-SHIP10-HULL, where the first and second level names (L1 and L2) are COMR—subsystem name (i.e., ISDS in this example) by convention, the L3 name is a design project name (SHIP10), and the L4 name (HULL) is a particular data file associated with SHIP10.

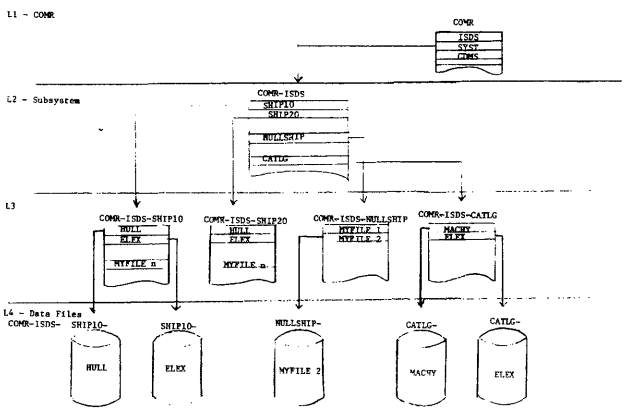


Figure 2—Sample COMRADE file directory entries

* "The Integrated Ship Design System, Model 1—System Development Plan," internal document of the Computation and Mathematics Department, NSRDC, February 1971.

As shown in Figure 2 the CPFMS catalog contains pointers from level to level. These pointers include next level names, passwords (to permit access to the next level) and subdirectory numbers (to speed access to the next level). In general, two types of information are kept in the catalog system: (1) security information needed to assure file integrity and to access the file; and (2) historical information about cataloged files that may be used to provide profiles on file usage.

The present implementation of CPFMS provides two security schemes for accessing files. In the first scheme subsystem projects are conceptually subdivided into subprojects. A file can then be associated (when cataloged) with a subproject or subprojects. CPFMS can prevent a subsystem user who has not been assigned responsibility for this subproject (or these subprojects) from accessing the file. This scheme involves maintaining a **F**ile **A**ccess **L**ock (FAL) for each CPFMS file in the catalog. The bit structure of the FAL reflects the area(s) of responsibility with which the file is associated. Similarly each user has a **F**ile **A**ccess **K**ey (FAK) associated with him (or her, for you feminists) that reflects his (or her) assigned area(s) of responsibility. The CPFMS routines verify that a user has been assigned the proper area(s) of responsibility before the file may be accessed. An example of the utility of this concept is as follows. Within ISDS a certain file is associated with the hull design effort. (Hull design is an area of responsibility.) This file can be accessed only by users who have been assigned hull design responsibility.

The second scheme for accessing files involves the use of "pseudo-passwords."* A pseudo-password of up to five characters may be assigned to a CPFMS file. The processing of the file is then limited to those users and programs which "know" the pseudo-passwords. The pseudo-password is also kept in the CPFMS catalog.

In addition to file access control information about CPFMS files, the CPFMS catalog includes historical information about file usage. This includes the name of the file creator and file creation date, as well as the name of the last user to modify the file and the date of the last file modification. This and other information stored in the catalog will be useful in producing reports concerning file usage.** It is envisioned that on-line ability to verify the last usage and modification of a file will prove invaluable in a large multi-user design system. (The file usage reports are considered to be an extension of the user-to-user communication process.)

At present, the COMRADE Permanent File Management System meets the needs of the present community of COMRADE users. However, expected future growth of the COMRADE community dictates that new capabilities be added and present techniques be modified. With the growth in the number of users the number of CPFMS files can be expected to grow correspondingly. As the number of files increases, tighter file controls will be called for, i.e., daily auditing of files may be necessary; a generalized file usage reporting system will be needed; and automatic file purging with an off-line hierarchical storage system may be needed. Each of these areas are to be addressed in the future.

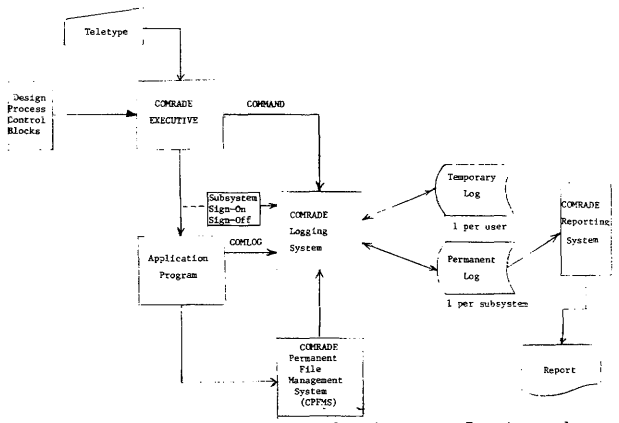


Figure 3—COMRADE design administration—Logging and reporting system

COMRADE LOGGING AND REPORTING SYSTEM

Early in the planning of COMRADE it was deemed essential that a comprehensive system for recording and reporting subsystem activity be developed. Reports were to be generated for subsystem managers, for developers, and for users for the purposes of subsystem management, monitoring, and inter-user communication. The recording of subsystem activity was to be as transparent to the user as possible. The result of this planning is the implementation of the COMRADE Logging and Reporting System (LRS). (See Figure 3). The system is designed to record and report activity within given subsystems rather than within COMRADE as a whole. Reports can be tailored to the needs of individual subsystems. LRS provides basic modules and techniques that make such reports possible.

To illustrate the necessity of tailoring the reporting of subsystem activity to the needs of the subsystem consider the example of the ISDS subsystem. Since ISDS is a ship design subsystem, the LRS for ISDS is closely concerned with providing reports based upon activities pertinent to ship designs. Thus an ISDS design manager can easily generate a report that monitors the activity within a given ship design rather than all ISDS activity. On the other hand, within a subsystem such as PERSONNEL,† reports are generated about the activity of the entire subsystem. In PERSONNEL, unlike ISDS, there is no natural boundary such as individual design projects to limit reports.

* The term "pseudo-password" is used rather than password to distinguish these passwords from the actual SCOPE passwords.

** Commands and programs needed for the report generation are not yet implemented.

† The PERSONNEL subsystem of COMRADE processes personnel data at NSRDC.

In a subsystem that has implemented logging, basic questions that can be answered by the COMRADE LRS include: Who used the subsystem? When did they use the subsystem? What commands were used? Much of this information can be and is obtained from the COMRADE Executive[2] which calls a design administration subroutine. The name of each command to be executed is placed into a file called the log. (A temporary log is created for each user at subsystem sign-on which is merged into the subsystem's permanent log when the user signs-off.)

It is essential that entries placed in the log be uniquely identified for proper processing by the REPORT command. For this reason each entry contains as its first word a header word. The header word includes the clock time that the entry was placed in the log, the number of words in the entry and two numbers that uniquely identify this entry. These numbers are called primary type and secondary type.

The secondary type is used strictly to distinguish between entries of the same primary type. However, the primary type is used for an additional purpose called "selective logging and reporting." Some events can be considered more important than others. The more important an event, the lower the primary type assigned to it. Selective logging causes only events with a primary type lower or equal to a preselected primary type (called High Type to Log or HTL) to be logged. Requests to place entries into the log with primary types greater than HTL will be ignored. The HTL value may be set by a system manager by executing the proper command.

The selective logging ability can be used to advantage for subsystem debugging. While a subsystem is being debugged many events can be logged. Then the subsystem implementors can use the resultant reports to closely monitor subsystem operations. Later the value of HTL can be lowered, eliminating excess logging.

Selective reporting is similar to selective logging. It is controlled by a parameter (called maximum level to report or MLR) that may be specified in requesting reports. The report generator (a COMRADE command appropriately titled REPORT) will include in reports only those events whose primary types are equal to or lower than MLR. Thus, reports may comprehensively reflect system usage or may be limited to only those events considered most important.

Other parameters that may be specified to the REPORT command include the site where the output is to be printed; a user name to which the report is limited; maximum number of lines to be printed; the first and last dates to be covered by the report; and in the case of subsystems where design names are significant, the design name to which the report is limited.

A sample report for the ISDS subsystem is shown in Figure 4. This report was generated on January 10, 1973. Parameters specified in generating this report include first and last dates of January 2, 1973 and January 10, 1973 respectively; maximum level to report of 30; and maximum lines to be printed of 25.

```
1   REPORT  1/10/73 -  FOR PERIOD FROM 01/02/73 THRU 01/10/73
2           MAXIMUM LEVEL REPORTED = 30
3
4   1/ 2/73  CAJBBRAINI   DDG3
5   14:08    ISDS ENTERED
6   14:33     FUEL       - CP       1.03      PP    23.66
7   15:07     LOGOUT     - CP       6.78      PP   244.89
8              ESTIMATED COST =  $ 29.50
9
10  1/ 3/73  CAGXDAVIS   DDG3
11  10:37    ISDS ENTERED
12  10:43     FUEL       - CP        .94      PP    29.41
13  11:14     LOGOUT     - CP       6.24      PP   254.65
14             ESTIMATED COST =  $ 18.50
15
16             CAJBBRAINI   DDG3
17  13:02    ISDS ENTERED
18  13:04     RETRIEVAL  - CP        .97      PP    25.22
19  13:16     UPFUEL     - CP       6.12      PP    82.05
20  13:18     RETRIEVAL  - CP      11.40      PP   155.19
21  13:32     LOGOUT     - CP      15.76      PP   214.22
22             ESTIMATED COST =  $ 15.00
23
24  1/ 4/73  CAWEWILLNE   SYSSHIP
25
26  LINE COUNT EXCEEDED
```

Figure 4—Sample report

Three complete ISDS sessions are reflected in this sample. The second session beginning on line 10 will serve as an illustration. A user known to COMRADE as CAGXDAVIS entered the ISDS subsystem on January 3, 1973 at 10:37 with a design project name of DDG3 (a ship design name). At 10:43 she executed the FUEL command. (The running CPU and PPU (peripherial processing unit) times were .94 and 29.41 seconds respectively at the *start* of FUEL execution.) At 11:14 CAGXDAVIS logged out of COMRADE. The ISDS session was estimated to cost $18.50.

SUPPORT FUNCTIONS FOR COMRADE SUBSYSTEMS

The support functions for COMRADE subsystems consist of programs and modules needed for a broad class of COMRADE subsystems, both design and non-design systems. The components, while performing widely different tasks, may together be used to achieve workable subsystems. Presently the support functions include the following components: (1) a command procedure and program for maintaining a list (file) of valid subsystem users; (2) several programs for initializing subsystem operations (subsystem sign on); and (3) a program to terminate subsystem operations (subsystem sign off).

The command MODUSERS allows the file of valid users for a subsystem to be displayed and modified at a terminal. The file contains the names, file access keys (FAK's) and command access keys (CAK's)[2] of the subsystem users, each of which may be requested by terminal user's option. The use of MODUSERS is limited to those COMRADE users (usually project managers) who have the proper CAK.

The file of valid users is processed by one of the subsystem sign on routines. This routine must verify that the user name is in the file of valid users. These keys are then available for processing by the Executive and by the CPFMS routines.

Before an exit is made from a subsystem, the subsystem sign off routine must be executed. The purposes of this routine include: temporary log to permanent log copy necessary for logging and miscellaneous subsystem cleanup.

The design support system is still under development. A significant project to develop a command procedure to initialize designs within design subsystems has recently been completed. However, a corresponding command procedure to remove designs has not yet been implemented. Future areas of research and further development include the area of file and command access conventions, explicit inter-user communications and active project monitoring and control.

REFERENCES

1. Rhodes, T. R., "The Computer-Aided Design Environment (COMRADE) Project," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.
2. Tinker, R. W. and Avrunin, I. L., "The COMRADE Executive System," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.
3. Willner, S. E., Bandurski, A. E., Gorham, W. C. Jr. and Wallace, M. A.. "The COMRADE Data Management System," Presented at the 1973 *National Computer Conference*, New York, June 1973. American Federation of Information Processing Societies.

A business data processing curriculum for community colleges

by DONALD A. DAVIDSON

LaGuardia Community College
Long Island City, New York

A business data processing curriculum must, of necessity, be both dynamic and flexible. We must constantly be seeking to fulfill the needs of industry in our environs.

LaGuardia Community College is located in New York City, which is probably the largest marketplace for business applications programmers in the world. Because LaGuardia is situated in the center of commerce, it was decided, when setting up the college, to make cooperative education the key thrust of the institution. The Cooperative Education Plan offers the student the opportunity to combine classroom learning with practical work experience. It is designed to help students determine and explore their own individual goals and, in general, to help them develop increased knowledge and skills in their major field of study, explore different career possibilities, and obtain experiences which will promote educational as well as personal growth.

Built into the structure of the college, cooperative education helps keep the college in touch with developments outside of it. Identifying work internships and placing students on various jobs attunes the college to changing needs in terms of career opportunities and related curricula. LaGuardia operates on a year-round quarter system. Full-time students spend their first two quarters studying on campus and then begin to alternate off-campus internship terms with on-campus study terms. In the course of the basic two-year program, a student will take five study quarters and three work internship quarters.

The paid work internships in many ways are also academic experiences because they allow the student to practice in the "real world" what they have learned in the classroom. Since the students are alternating work with study, there is a constant feedback between the students out on the work internship and the instructors in the classroom. The feedback is largely in the area of modification of course content in the data processing area, so as to encompass all of the latest innovations in the industry. We find that the students are very perceptive and wish to share the knowledge which they gain on the job with their fellow students and, of course, with their instructors. This knowledge is generally in the areas that are unique to the applications that they are working with. Some students may be working in banking, some in insurance, some in retailing, and some in manufacturing, etc.

These work-study jobs are developed by a dedicated cadre of cooperative education placement personnel in conjunction with the members of the data processing faculty, serving as technical liaison. Since we know the types of jobs that our students will undertake, both in their cooperative internships and also upon their graduation, we are able to tailor our curriculum and course content to the needs of the business data processing community. Because of the diversity of the marketplace, we feel that our curriculum will prepare our students to find jobs in the EDP field in all areas throughout the country.

Our curriculum, as it now stands, begins with an "Introduction to Data Processing" course taken during the student's first quarter in residence at the college. This course, which is business-oriented, includes such topics as: the history of EDP; a brief introduction to the punched-card and unit-record equipment; an introduction to electronic computer theory and numbering systems; analysis and flowcharting; and programming languages. In order to "turn the students on" to computers, we utilize the interactive BASIC language.

The hardware which we utilize in the introductory course is the Data General Nova 1200 with six (6) ASR33 Teletype terminals. These six terminals support five sections of about thirty students each, or roughly 150 students in our "Intro" course.

The second course that we introduce the students to is called "Basic COBOL Programming." We chose COBOL because most studies in the past two years (including our own) had shown that this language is used by at least 60 percent of the EDP installations in the greater metropolitan area of New York. We use behavioral objectives in teaching our EDP courses at LaGuardia. We set up goals for each student, so that they may ascertain their own mastery of the course. Students' grades are based on the number of programs that they complete. Evaluation of the levels of attainment aids both the faculty and the cooperative education coordinators in work internship placement.

During the third study quarter, we offer a course in "Advanced COBOL Programming" which covers

advanced applications of COBOL, such as nested loops, multi-dimensional table handling, and processing of disk and tape files. We examine various types of file management techniques and the student analyzes, flowcharts, codes, debugs, and documents many interesting programs. The advanced COBOL course is taken in conjunction with a course in "Systems Design and Analysis" that further advances the student toward the goal of becoming a constructive and useful member of the data processing community.

When the student returns for the fourth study quarter, he or she may take a course in "Operating Systems" and either "RPG Programming" or "Basic Assembler Language" (BAL). During the final study quarter, the student may opt for either PL/1 or FORTRAN, depending on their prospective employer's recommendations.

The sequence of courses during the last three quarters is generally influenced by the cooperative employer's needs. There is a constant series of contacts being made between students, instructors, and coop employers throughout the student's stay at LaGuardia. This team effort is the fulcrum around which everything revolves. We believe that the evolutionary business data processing curriculum at LaGuardia, which is constantly being re-evaluated by the very nature of the cooperative education program, could become a model for other community colleges throughout the nation.

Computing at the junior/community college—Programs and problems

by HAROLD JOSEPH HIGHLAND

State University Agricultural and Technical College
Farmingdale, New York

INTRODUCTION

This and the following papers contain different views of the same subject—"Computing Education at the Junior/Community College." It is a topic that has been long neglected, swept under the rug so to speak, in computing circles. It is about time that this topic was fully aired and its vital importance recognized by all engaged in computer education, business, industry and government. There are probably more students and more teachers involved in computer education at the junior/community colleges than at any other level of education.

Before proceeding, I should like to thank the participants of this panel for their enthusiastic cooperation and valuable contribution. Although they represent different parts of the country and different programs, they are all involved with junior/community college computer education.

- Dr. Alton Ashworth of Central Texas College (Killeen, Texas) has been in charge of developing a model program for the junior/community college under an Office of Education grant.
- Mr. Robert G. Bise of Orange Coast College (Costa Mesa, California) developed the prime program in computing education at the junior/community college-level, which has served as the prototype for such programs in California.
- Professor Donald Davidson of LaGuardia Community College of the City University of New York (Long Island City, New York) has been instrumental in developing a work-study program for underprivileged students in a metropolis.
- Dr. John Maniotes of the Calumet Campus of Purdue University (Hammond, Indiana) has had extensive experience in developing and running an integrated two-year and four-year program in computing on his campus.
- Professor Charles B. Thompson of New York State University Agricultural and Technical College at Farmingdale (Farmingdale, New York) has been very instrumental in the development of a dual program designed not only to meet the needs of career-oriented students but also one to serve students who plan to continue their education in this field at a four-year college.

Furthermore, I should like to define, or perhaps explain, the use of the term, *"academic computing in the junior/community college."* It was selected, not because we need to add to the myriad of terms we already have in computer education, but because there was no term broad enough to cover all aspects of computing education at this level of higher education.

- In some institutions, the term, *computer science*, is used but many times the courses and the level at which they are taught bear no relationship to computer science taught at a four-year college, following the guidelines of Curriculum '68 which was developed under Dr. William F. Atchison.
- In other institutions, the term, *data processing*, is used; but here again there are extremely wide variations. Not all such programs are solely and purely business-oriented.
- The term, *computer technology*, is likewise encountered at the junior/community college. Some of these programs are designed to educate electronic technicians; others involve the training of computer operators. Still others more closely resemble computer science at the four-year college or data processing in a college of business administration.
- Finally, we are beginning to encounter the term, *information processing*, since curriculum titles are being used at times to show that one is keeping up with the state of the art. Oftentimes, the courses and their content are far different from the program proposed by the ACM Curriculum Committee on Computer Education for Management (C^3CM) for undergraduate education under the leadership of Dr. J. Daniel Couger.

JUNIOR/COMMUNITY COLLEGE PROGRAMS

Having served as a director of a graduate business school as well as a dean of instruction at a four-year liberal arts college, I was startled when I joined a two-year

college to develop a program in computing. The lack of uniformity in course selection, course sequence, proportion of theory and application taught, were immediately evident. Programs were being developed all over the country and often were designed to meet immediate near-term needs or merely to be "modern." Little or no concern was given to the impact of technology and the possibility of obsolescence of the graduates from these programs. One of my associates engaged at the graduate level in computing assured me that diversity meant freedom. Yet, I see this diversity as chaos with too many charletons and humbugs performing rituals in hexidecimal in the classroom without concern for the quality of education or the future of their students, or sincere educators who cannot upgrade the quality of their educational programs because they cannot provide their administration with "authoritative, professional guidelines." Now, many years later, I find that some of the extremes have died but there is still a lack of cohesion in computing programs at the junior/community college level. Let me just touch several of these areas.

Department structure and affiliation

Where is computing taught at the junior/community college level? In some schools there is a separate department of computer science, data processing, computer technology; a department which sets its own curriculum and guidelines. In other institutions, computing is part of the business department; it is one more 'major' in the class of marketing, accounting or management. Still at other colleges, computing is part of the mathematics department; here we most often find that curriculum which is closest to the four-year college in computer science. Yet, the emphasis is primarily a mathematical approach without concern for computing applications in other areas.

Faculty education and experience

Because of the rapid growth in the number of junior/community colleges over the past several years and the increased demand for computer faculty at four-year and graduate schools, the junior/community colleges have been low man on the totem pole. Except for a core of dedicated teachers, most of those with adequate education and experience have not, until recently, been attracted to the junior/community colleges. At a level where application is emphasized at the expense of theory, we find many teachers who have never had practical experience in computing in industry, business and/or government. Too many teach from the textbook or repeat what they have learned from their professors at four-year and graduate schools, and many of them as well have spent all their years in the ivory tower.

Programs and options

Depending upon the individual school, the student interested in computing may be forced into some narrow area, such as business data processing. Many of the junior/community colleges are too small to offer a broad spectrum of computing courses. The areas in which they train students include:

- keypunch operators
- computer operators
- computer programmers.

In some schools, the computing programs are career-oriented, and except in few cases, these students find that their two years of education is discounted if they wish to continue their education at a four-year college. In other schools, computing is computer science oriented and the student wishing to work upon graduation does not possess the necessary skills to find and hold a job.

The problem of training computer operators is a critical one at the junior/community college level. Too many of the schools have inadequate computing facilities to provide a proper program in this area. Some have turned to simulators to do the job. Any of you who have had experience with most of these simulators recognize their numerous shortcomings. (I should apologize to my colleagues in the simulation area for that comment since I am editor of the *ACM SIGSIM Simuletter*, but in this case, I feel that it is the truth.) Other schools have turned to work-study programs or cooperative training programs wherein the students study the theory of operations in school but obtain their experience at cooperating companies in industry and business.

Computer courses and sequence

In this stage of computing, can anyone imagine spending time in a one-year course in "electric accounting machines?" Yet, there are a number of two-year schools, both public and private, that train their students in unit record equipment and spend considerable time in wiring. At the other end of the spectrum are the schools which require career-oriented students to take courses in logic circuits, Boolean algebra, and hardware specifications. In fact, until recently, there was one school which spent one half of a one semester course teaching keypunching to students who supposedly were being trained to become junior programmers.

Where does a course in systems analysis fit into the curriculum? Is this one which is taught in the first semester concurrent with an introduction to data processing, or is this the capstone in which students can utilize the information they learned in all their other courses? Similarly, should the students be required to take statistics with the mathematics department and do their work with pencil and paper or even a calculator, or should they use

the computer and spend less time on the mechanics and more on the concepts?

Credits in computing

How many credits in a two-year program should be devoted to computing? Currently, there are schools that offer a data processing "major" with as little as 12 credits in computing (and six of these are in electric accounting machines) to schools which require almost 40 credits out of a total of 62 to 64. What is the minimum number of courses and/or credits which should be taught? And which courses?

Computing facilities

Many of the junior/community colleges have some computing facilities available for student use. Yet there are some offering computing programs that do *not* have any computing facility available for student use. One cannot but question the value of such a program.

Furthermore, what type of facilities are available for which programs? Do you need the same for computer science (in the four-year sense) as you do for a career-oriented program in the business area?

It is possible to continue in examing other areas of diversity, but it should be apparent that there is a wide spectrum of programs under the heading of computing in the junior/community college.

SOME PROBLEMS TO BE ANSWERED

The two-year college, no matter what it is called (junior or community or as has become fashionable to leave either term out), is a unique institution in education. In some cases its students vary little from those who enter a four-year institution, but in other cases, these two-year schools receive those students who cannot be admitted to the four-year colleges.

Computing languages

The number and intensity of languages studied varies greatly among the junior/community colleges. There is also great variation in which language is taught first and in what sequence are languages studied by the student. Among the languages offered by the two-year colleges are: BASIC, COBOL, FORTRAN, RPG, PL/I, APL, AL, JCL. At some schools students are required to take only one language during their entire two years, while in a few three and even four languages are taught to all students as part of the basic program.

At this level of instruction, which is the simplest language to introduce to the students? Some look upon BASIC as being too much like FORTRAN and therefore too scientific, unsuitable to students going into the business field. Many schools start with FORTRAN, but in one, a year of COBOL is the prerequisite for the study of FORTRAN. A few, like my own college, start with PL/I.

Since these schools are more often under local community control as compared with four-year colleges and universities, the programs should be designed to meet community needs. But a broader view is also necessary. It is about time that we recognized that the four-year colleges in computing are almost monolithic in their programs as compared with the two-year schools. The computing community has an obligation to see that some level of competency or adequacy is set for these institutions. I am not proposing a system of accreditation but the establishment of some guidelines, fundamental curriculums to meet several purposes.

Attention should also be devoted to the high level of attrition in these programs. Is it really the student's fault that they have failed? Or is it the lack of a properly sequenced curriculum, adequate teaching aids, quality of the teachers, or what?

Many teachers and administrators at the junior/community college level require some professional ammunition in attempting to get college presidents, local appropriation committee, etc., to upgrade existing equipment and programs. It is here that ACM can play a strong role, but it must be done now.

In addition, a greater exchange of information among the junior colleges is necessary. An exchange of curriculums, course outlines, booklists—an airing of problems: how much mathematics should a student be required to take, what techniques have been used to cut attrition, what arrangements have been made for transfer of students—is essential in this area.

It appears apparent that in light of accelerated computer technology, the limited computing facilities at the junior/community college level and concomitant problems that many of the two-year programs offer today will not be viable within the near future. Computing is already making inroads at the secondary school level. In some parts of the country, and this we have locally, there are special vocational educational centers for intensive training of high school students. If they develop adequate programs for training input clerks, control clerks and computer operators (and eventually they will), what will be taught at the two-year level?

Finally, to do its job effectively, the junior/community college must have better information about industry's and government's needs in the computing field. Today, the Bureau of Labor Statistics either lumps all those engaged in computing into a single category or at most separates programmers from the rest. What can be done to obtain greater insight into these needs so that more effective programs can be developed and taught at the junior/community college level?

The problems are many and those who are truly interested in doing are few. Those of us within ACM should seek some dynamic structure within which to operate; now is the time to start.

The two year and four year computer technology programs at Purdue University

by JOHN MANIOTES

Purdue University
Hammond, Indiana

INTRODUCTION

There are eight kinds of educational programs in the United States which presently supply industry with the majority of its computer-EDP oriented employees. For lack of better terminology, I would classify these programs as follows:

(1) The two year Data Processing (DP) programs* which are concentrated at vocational institutions, community colleges, junior colleges and at some senior colleges and universities.
(2) The four year academic programs offered by many senior colleges and universities in the areas of Business Data Processing and Computer Science.
(3) The graduate level programs in Information Systems and Computer Science offered at some major colleges and universities.
(4) The specialized programs offered by the various computer manufacturers' education centers.
(5) The company in-house training programs.
(6) The so-called private commercial schools, many of which have been established through franchising, and which usually offer training ranging from 3 to 12 months.
(7) The private home study schools.
(8) The various high schools which offer vocational oriented training programs in EDP.

Purdue University offers extensive instruction in the areas of computing and information processing ranging from the freshman level to the Ph.D. degree. Purdue University currently offers B.S., M.S., and Ph.D. degrees in Computer Science as well as A.A.S. and B.S. degrees in Computer Technology. The main difference between these two areas is that the Computer Technology program is practical and application-oriented, while the Computer Science program is theoretical and research-oriented.

The Computer Technology programs are offered at Purdue's three regional campuses at Hammond, Fort Wayne, and Westville and at Indiana University's Indianapolis regional campus. Table I describes the regional campuses with respect to their location, distance from the Lafayette campus, enrollment, and principal computing equipment. The Computer Science programs are offered at Purdue's Lafayette Campus.

* Sometimes these programs bear the name of Computer Programming Technology, or simply Computer Technology.

THE TWO YEAR COMPUTER TECHNOLOGY PROGRAM

Currently, the two year programs at the regional campuses are designed to produce graduates in the occupational group that begins with the computer programmer in either the commercial or technical areas of programming. The regional campuses offer two options in their two year programs. These options are referred to as the Commercial option and the Technical option, respectively. However, even with the dual options the enrollment is overwhelmingly business-oriented. For this reason this section will reflect primarily with the business-oriented two year Computer Technology program. The curriculum for the two year program is divided into five areas:

(1) Data processing and computer basics and equipment
(2) Assembler and compiler languages
(3) Organization of business
(4) Business applications
(5) Supporting sciences and electives

During the first year of the program as indicated in Appendix A, the students acquire an introduction to data processing, programming, and computers. In addition, they study such academic courses as English composition, speech fundamentals, basic and intermediate accounting principles, and data processing mathematics. In the second year, the students concentrate heavily on computer programming, programming systems, operating systems, systems analysis, and systems applications. In addition, the students continue their related course study in areas such as technical report writing, economics, and statistics.

An important point to keep in mind is that the two year program emphasizes the practical, rather than the theo-

TABLE I—Some Statistics Regarding Purdue University and Its Regional Campuses

Institution	Location	Distance (Miles) to Lafayette Campus	Enrollment Fall 1972	Principal Computing Equipment
Purdue University	Lafayette	—	26,204	CDC 6500
Purdue Regional Campuses				
Calumet	Hammond	100	4,880	IBM 360/22
Fort Wayne	Fort Wayne	115	2,794	IBM 360/22
North Central	Westville	80	1,354	IBM 360/22
Indiana Univ. Regional Campus				
Indianapolis	Indianapolis	60	16,938	IBM 360/44*

* The IBM 360/44 at Indianapolis is linked to a CDC 6600 at Indiana University, Bloomington, Indiana. The other three IBM 360/22's are linked to the CDC 6500 at Purdue University, Lafayette, Indiana.

retical aspects of EDP. In addition, the program emphasizes the solution of EDP problems using the "hands on" approach to operate computers and other peripheral devices and to debug their programs.

Strong emphasis is placed on "real-life" laboratory exercises which are intended to reinforce the student's knowledge of data processing techniques by requiring the student to apply them in a broad spectrum of practical applications. In addition the "hands on" approach exposes students to a wider variety of applications and techniques than most programmers would receive in a year of on-the-job training since most of the latter training tends to focus on one language and a relatively narrow range of applications.

In the two year Computer Technology program, the curriculum is designed to prepare a person for the entry level position of a programmer and to perform the following functions:

(1) Analyze problems initially presented by a systems analyst with respect to the type and extent of data to be processed, the method of processing to be employed, and the format and the extent of the final results.
(2) Design detailed flowcharts, decision tables, and computer programs giving the computations involved and the sequences of computer and other machine operations necessary to edit and input the data, process it, and edit and output information.
(3) Utilize the programming languages of COBOL, RPG, FORTRAN, as well as a machine and assembly language to construct the necessary program steps, correct program errors, and determine the cause of machine stoppage.
(4) Verify the accuracy and completeness of computer programs by preparing sample data and testing it on the computer.
(5) Evaluate and modify existing programs to take into account changed requirements.
(6) Confer with technical personnel with respect to planning new or altered programs.
(7) Prepare full documentation with respect to procedures on the computer and other machines and on the content of the computer programs and their full usage.
(8) Devise more efficient methods for the solution of commercial or scientific problems.
(9) Comprehend the major concepts, types of equipment, programming systems, and operating systems related to EDP.

* The IBM 360/44 at Indianapolis is linked to a CDC 6600 at Indiana University, Bloomington, Indiana. The other three IBM 360/22's are linked to the CDC 6500 at Purdue University, Lafayette, Indiana.

After successfully completing the two year Computer Technology program, students are awarded an Associate Degree. A student graduating from the program not only has studied theories and principles but has had extensive practical experience in operating and applying data processing techniques on modern computing equipment. This combination enables the graduate to step into an entry level programming job and become productive in a short period of time.

The first Computer Technology program was initiated in 1963 at the Calumet and Indianapolis regional campuses, respectively. Course revisions and curricular changes have taken place during the past ten years in order to keep pace with the current state of the art. In addition, changes have been designed to "fine-tune" the two year programs while providing flexibility in individual courses and regional variation to meet the special needs at each community where the curriculum is taught.

Appendix A illustrates the two year program at the Purdue Calumet Campus that has undergone "fine-tuning" in order to deal with third generation computers. The curriculum in Appendix A is offered on a semester basis, and it can be used for illustrative purposes. (The sequence of courses in the two year program varies between regional campuses, and it is not the intent of this paper to debate the merits of different course sequences.)

The curriculum in Appendix A reflects the following changes that have taken place during the past ten years:

(1) Many of the original programming, business, and supporting courses in Appendix A have been assigned specific names so as to become readily identifiable and to reflect their status in the curriculum.

(2) The one-time importance of unit record equipment (tab equipment) has diminished. It is no longer necessary for a viable third generation program to concentrate mainly on "board wiring" and punched card applications. Hence, the priority of unit record equipment has been considerably reduced (not eliminated) in the curriculum.
(3) The importance of first and second generation computers has also diminished. Third generation computer hardware and software concepts are stressed by the curriculum in Appendix A.
(4) File organization techniques and disk/tape programming concepts are emphasized together with input/output control systems and the functions of an operating system.
(5) Two semesters of assembly language together with the compiler languages of COBOL, RPG, and FORTRAN are also stressed since these are the common tools that the programmer utilizes on the job.

THE FOUR YEAR COMPUTER TECHNOLOGY PROGRAM

This baccalaureate program is a two year "add on" curriculum which is open to associate degree graduates of Computer Technology or the equivalent in data processing. The program builds on the students' knowledge of computer programming acquired in the first two years, and emphasizes the practical aspects of such areas as computer systems analysis and commercial systems design. The inclusion of many elective courses enables the students to pursue areas of special interest. Graduates from this third and fourth year of study are prepared to fill a variety of positions related to data processing, computer systems, systems analysis, systems programming, and computer programming.

The objectives of an additional third and fourth year of study leading to a baccalaureate degree are summarized below:

(1) With regard to the student, the objectives of the curriculum are:

(a) General—To prepare a graduate who is:
1. Proficient in computing, information processing, and data management techniques;
2. Capable of developing computer programs in a wide variety of application areas and in a number of commonly used languages;
3. Capable of productive effort for the employer shortly after graduation;
4. Capable of remaining current with the changing technology.

(b) Technical Competence—To prepare a person who is knowledgeable concerning:
1. Mathematical concepts relating to computer programming;
2. Techniques used in the definition and solution of commercial systems problems;
3. Computer and peripheral equipment operations, computer operating systems, and data communications;
4. Fundamentals in the subject matter areas most closely related to computer applications.

(c) General Education—To broaden the individual through exposure to:
1. Humanities and social sciences;
2. Oral and written communications;
3. Business, management, and supervisory concepts;
4. Elective courses directed toward further individual development.

(2) With regard to the program, the objectives are to provide a curriculum which is:

(a) Viable and responsive to the changing technology;
(b) Based on a two year modular structure that encompasses both the commercial and technical options of an associate degree program in Computer Technology.

The format for identifying the requirements for the baccalaureate degree differs from that normally found in a standard college or university catalog. In addition to the usual semester-by-semester plan of study, the minimum requirements in the Computer Concentration Courses are specified as indicated in Appendix B. The baccalaureate program has been offered since 1968 at the Indianapolis regional campus and is scheduled to be officially offered at the Purdue Calument Campus during the 1973 Fall Semester.

The computer course requirements provide the students with a flexibility allowing for varied implementations. Thus, as indicated in Appendix B, one student may take course sequences emphasizing Computer Systems Analysis while another emphasizes Systems Programming. This flexible structure also allows the curriculum to remain current in a rapidly changing industry without requiring constant revision.

THE ROLES OF COMPUTER SCIENCE AND COMPUTER TECHNOLOGY*

Computer Technology training currently is provided to students at three levels; all levels stress in-depth practical experience. The two year associate degree program develops computer practitioners whose competency lies pri-

* The author wishes to thank the Computer Technology staff and the Computer Science staff at the Indianapolis regional campus for some of their ideas and thoughts as expressed in this section.

marily in programming and secondarily in systems. The learn-by-doing technique is heavily stressed. The baccalaureate program is broader-based and affords the students an opportunity to have a business, scientific, and communications exposure. The primary goal is the development of computer professionals well versed in the current state of the art. The technologist is provided with the perspective to apply his tools through an integrated systems approach to data processing problems. The third level concerns the direct charge of providing service courses. Certain offerings for the community at-large are non-credit, while credit courses are offered for other departments of the University to fill the need for computer-oriented electives in various curriculums. Each course is designed to meet the needs of the department in question.

Computer Science as a discipline is also concerned with the integrated use of the computer as a component in the overall solution to problems. Students are trained in the formulation of computer-oriented solutions peculiar to design-oriented problems. These persons have a mathematical background suited to the needs of their discipline as a framework within which to arrive at a solution. A computer scientist functions in an environment analogous to that of the theoretical mathematician while the technologist functions in an environment analogous to that of the applied mathematician.

In cases where the problem is so new and/or complex as to require new solution techniques or substantial modifications or new applications of existing techniques, a computer scientist acts in conjunction with the discipline-oriented professional and the computer technologist in developing the problem solution. The computer scientist has the depth of mathematical training and the breadth of theoretical knowledge to effectively contribute to the decision-making process and to the feasibility of developing new techniques such as the development of new numerical methods, algorithms, optimization techniques, simulation models, new higher-level languages, operating systems, or management information systems. In carrying out the results of such planning, the creativity of the computer scientist is his contribution to the problem solution; effective use of established methods is the contribution of the computer technologist.

In general, the computer scientist is a theorist with a broad interdisciplinary overview, while the computer technologist is a specialist in the techniques of analysis and implementation. The scientist is sought as one who can synthesize diverse information into an integrated solution approach, while the technologist is sought as a professional who can produce computer-solution results efficiently.

Accordingly, Computer Science and Computer Technology are each individually responsible for their respective degree programs, their implementation and development, as well as those service courses which meet particular needs. Therefore, even in those courses and offerings which appear similar in content, there is a difference in emphasis and orientation, reflecting the different roles of the two disciplines. The A.A.S. and B.S. programs in Computer Technology and the B.S., M.S. and Ph.D. programs in Computer Science are part of a continuum called computing and information processing.

PROBLEMS FACED BY THE COMPUTER TECHNOLOGY PROGRAMS

Summarized below are some of the problems faced by the two year and four year Computer Technology programs. Although some of these problems may be pertinent to Purdue University, others are general enough to apply to other institutions which have similar academic programs.

Staffing

A problem that the Computer Technology staff faces is the constant updating required in their field as compared to their colleagues in such fields as liberal arts or the humanities. It has been said that the half-life of one's computer-EDP knowledge obsoletes each five years due to the many new developments that are occurring in the field. Recognizing this problem, the staff has been periodically infused with new computer-oriented knowledge through attendance at summer institutes, computer manufacturers' education centers, technical meetings sponsored by professional organizations, and by various consulting assignments in industry. In addition, the campus library's selection of computer-oriented books and journals has been expanded so as to enable the staff to remain abreast with the latest developments in their field.

Another problem that has been experienced over the years concerns the difficulty in hiring experienced instructors who possess up-to-date knowledge about computers and data processing applications. One of the problems contributing to this difficulty has been the low starting salaries and fringe benefits commonly found in the teaching profession.

University administrators must be constantly made aware that computer-oriented staff members have unique problems and additional resources must be made available to compensate for these deficiencies. The demand for competent computer-oriented instructors is high and the supply has a long way to catch up.

Student transfer problems

No serious problems have been experienced by the Purdue graduates of the two year programs who have transferred to the baccalaureate Computer Technology program at the Indianapolis regional campus. This is due to the fact that at all regional campuses the computer equipment is from the same manufacturer and the courses are structured essentially in the same manner.

Some problems have been experienced whenever two year graduates from other schools which did not have similar computer equipment transferred into the bacca-

laureate program. The problems in these cases stemmed from the lack of familiarity of the operating system and the assembly language of the computer utilized in the baccalaureate program.

Problems have also been experienced whenever students from private commercial schools have tried to transfer into the two year program. These types of students have been found to be weak in EDP fundamentals, flowcharting, programming logic, and documentation. In some instances these students have had to retake some of the basic computer-oriented courses before they were fully admitted to the two year program.

Evaluation of students' computer programs

Currently various methods exist to evaluate students on their computer-oriented courses using such means as written and oral exams, quizzes, homework problems, and laboratory exercises. A problem faced by instructors in such courses involves the evaluation or grading of students' *computer programs* especially those programs of some complexity. Questions most often asked in this area are: What do you grade on? What constitutes a good (or bad) program? What parameters do you consider important—execution time, amount of storage utilized, or the number of unsuccessful attempts tried before completion? These are areas which bear further study and thinking by instructors since programming is an art and not an exact science.

Instructional materials

More than 1,000 books from more than 120 publishers have been published for the computer-EDP field. Currently good textbooks, student work manuals, and visual aids exist for introductory computer or data processing courses and for computer programming courses which appear in the freshman and sophomore level of the Computer Technology program. In fact one may state that there are too many books published for these courses at these levels.

Good textbooks, student work manuals, and visual aids for the junior and senior levels of the Computer Technology program are practically non-existent. The courses which require good textbooks are: Management Information Systems, Operating Systems, Commercial Systems Applications, Computer Graphics, Hybrid Computing Systems, and Data Communications. The text material for these courses usually consists of reference manuals from computer manufacturers, notes from the instructor, or handbooks oriented for experienced professionals rather than for students. More effort needs to be exerted by textbook publishers in producing student oriented textbooks for these courses.

Computer-oriented aptitude tests

There is a need for the development of good aptitude tests to predict if an entering student will be successful in graduating from the Computer Technology programs or whether a graduate from these programs will be successful in a programming or systems analysis job position. Our A.A.S. and B.S. Computer Technology graduates have reported that they face in many instances an aptitude test when they apply for a programming or systems position. It seems that interviewers confronted with the problem of predicting job success among applicants for these positions have come to rely heavily on aptitude tests especially the IBM PAT. It is unfortunate that in many instances the IBM PAT score is the sole factor used to determine whether an applicant qualifies for further consideration. The IBM PAT scores are not an accurate predictor of job success.

It is apparent that the computer-EDP field needs to give our psychological test developers additional qualities to base their tests on if they are to perform the task of predicting job success as many employers believe they now do. In addition, further work is necessary to develop *"third generation aptitude tests"* in order to be part of the computer hardware and software presently available. Hopefully some of these tests can also be utilized as entrance examinations for computer-oriented academic programs.

Funds

As far as funds are concerned, it seems there are two bottomless pits at academic institutions—libraries and computer centers. Adequate funds to purchase or lease modern computers and their peripheral equipment, especially I/O terminals, is a problem faced by all academic institutions including Purdue University. Funds from the National Defense Education Act are scarce especially for institutions such as Purdue University that once were funded from this Act in the early 60's. In addition, computer manufacturers no longer offer large educational discounts for new computer equipment as they once did in the past.

Currently, the emphasis at Purdue University is to share computer resources at all levels. At each regional campus a common third generation computer is shared for all academic, administrative, and research oriented tasks. In addition these computers are linked to a CDC 6500 at the Lafayette campus thereby providing economically and efficiently computing power and storage to many users at one time. Typical COBOL or FORTRAN type problems submitted by Computer Technology students are processed at a cost of 20¢ to 30¢ a program on the CDC 6500 with an "average" turn-around time of approximately 5 to 15 minutes during non-peak times.

A question often asked is: "Where will additional funds come from?" I don't think there will be any significant outlays of funds from federal and state government sources. Nor will there be any sizeable student tuition increases. Rather I expect that academic institutions will have to increase the efficiency of their computer centers and start actively looking for ways of stretching their funds such as third party leasing.

APPENDIX A—CURRICULUM OUTLINE FOR THE TWO YEAR COMPUTER TECHNOLOGY PROGRAM AT PURDUE UNIVERSITY*

	Hours per Week			
	Class	Lab	Total	Credits
First Semester				
Introduction to Data Processing	4	2	6	5
English Composition I	3	0	3	3
Introductory Accounting	3	0	3	3
Algebra	3	0	3	3
Elective	3	0	3	3
	16	2	18	17
Second Semester				
Data Processing Math	3	0	3	3
RPG Programming	2	2	4	3
FORTRAN Programming	2	2	4	3
Fundamentals of Speech Communication	3	0	3	3
Cost Accounting	3	0	3	3
	13	4	17	15
Third Semester				
Assembly Language Programming I	3	2	5	4
Statistical Methods	3	0	3	3
Systems Analysis and Design	3	0	3	3
COBOL Programming	2	2	4	3
Technical Report Writing	3	0	3	3
	14	4	18	16
Fourth Semester				
Assembly Language Programming II	3	2	5	4
Commercial Systems Applications	2	2	4	3
Computer Operating Systems I	2	2	4	3
Computer Seminar	2	0	2	1
Principles of Economics	3	0	3	3
Elective	3	0	3	3
	15	6	21	17

* For the description of the courses, consult the latest edition of the "School of Technology Bulletin", Purdue University, Lafayette, Indiana.

APPENDIX B—COMPUTER TECHNOLOGY CURRICULUM OUTLINE FOR A THIRD AND FOURTH YEAR OF STUDY AT PURDUE UNIVERSITY*

	Hours per Week			
	Class	Lab	Total	Credits
Fifth Semester				
Data Communications	2	2	4	3
Computer Concentration Courses (2)	4	4	8	6
Calculus I	3	0	3	3
Communications Elective	3	0	3	3
Elective	2	0	2	2
	14	6	20	17

APPENDIX B (Continued)

Sixth Semester				
PL/I Programming	2	2	4	3
Computer Concentration Course	2	2	4	3
Calculus II	3	0	3	3
Physical Science Elective	4	2	6	4
Elective	3	0	3	3
	14	6	20	16
Seventh Semester				
Computer Concentration Courses (2)	4	4	8	6
Physical Science Elective	4	2	6	4
Social Science Elective	3	0	3	3
Humanities Elective	3	0	3	3
	14	6	20	16
Eighth Semester				
Computer Concentration Courses (2)	4	4	8	6
Social Science Elective	3	0	3	3
Humanities Elective	3	0	3	3
Electives	4	0	4	4
	14	4	18	16

* For the description of the courses, consult the latest edition of the "School of Technology Bulletin", Purdue University, Lafayette, Indiana.

The *Computer Concentration Courses* are defined as follows: Any two of the following sequences plus one additional computer-oriented course.

(1) Commercial Systems sequence
  (a) Management Information Systems I
  (b) Management Information Systems II
  (c) Financial Accounting
(2) Computer Systems Analysis sequence
  (a) Systems Analysis of Computer Applications
  (b) Computer System Planning
  (c) Design of Data Processing Systems
(3) Systems Programming sequence
  (a) Introduction to Computer Systems
  (b) Computer Operating Systems II
  (c) Systems Programming
(4) Technical Systems sequence
  (a) Numerical Methods
  (b) Topics in FORTRAN
  (c) Hybrid Computing Systems

The *General Requirements* for the baccalaureate program are:

(1) Completion of an Associate Degree in Applied Science, in Computer Technology or the equivalent.
(2) Completion of the Core Requirements, plus additional courses as required to complete a minimum of 130 semester credit hours which includes credits earned toward the Associate Degree. The additional courses are free electives, except that not more than 9 semester credit hours may be taken in the Computer Technology Department.

APPENDIX B (Continued)

(3) A minimum of 40 semester credit hours must be 300 or higher level courses.

The *Core Requirements* for the baccalaureate program consist of 111 semester credit hours in the following areas:

	Semester Credit Hours
(1) General Education	
(a) Communications (English, Speech, Report Writing)	12
(b) Social Science (Economics, Political Science, Psychology Sociology)	9
(c) Humanities (Creative Arts, History, Literature, Philosophy)	6
(d) Business (Industrial Management, Industrial Supervision)	9
(e) Mathematics (Including Calculus, Finite Mathematics and Statistics)	17
(f) Physical Science (Biology, Chemistry, Physics)	8
	61
(2) Computing Principles	
(a) Data Processing Basics	6
(b) Assembly Languages	8
(c) Compiler Languages	9
(d) Computer Systems	6
	29
(3) Computer Concentration Courses	21

Computing studies at Farmingdale

by CHARLES B. THOMPSON

State University, Agricultural and Technical College
Farmingdale, New York

Farmingdale Agricultural and Technical College is part of the State University System of New York. The College is one of three public two year colleges serving Nassau and Suffolk counties. The school is located 25 miles east of New York City on the boundary line dividing these two counties.

The computing program at the school is an academic program in data processing. The program was started in 1967, under the direction of Dr. Harold J. Highland. The program has about 150 day and 300 evening students enrolled. Computing support for the program is an IBM 360/30, DOS system.

The objective of the program is to equip the student with the skills and knowledge to enter the data processing field as a junior programmer. A junior programmer is defined as one who has a strong command of one language, familiarity with two others, has extensive experience programming structured problems, and has a general knowledge of computing and data processing systems.

The overall philosophy of instruction is application. Students are assigned programming problems as a primary vehicle of learning. The "hands on approach" rapidly develops command of programming and the confidence to program or solve problems.

The day student has a choice of choosing the scientific or the commercial option of the program. The first year is a core year for a second year of concentrated study in scientific programming, FORTRAN, or commercial programming, COBOL. Upon completing the program, the student is awarded an AAS degree in data processing.

The enrolling day student is generally a first generation college student. The academic background of the students vary widely, but can be grouped into those with three or more successful years of secondary mathematics, those without, and those with some knowledge of data processing. Students in all three groups have completed the program. They have entered the field as an operator or junior programmer or continued their studies in Computer Science, Programming and Systems, or Business.

The evening students have diverse backgrounds, but almost all have some knowledge of computing, varying from operations to systems programming. These students enter the program to advance their careers in the commercial aspects of data processing.

By and large, the data processing program has been a success; those who have completed the program can and have succeeded. Trends are developing, however, which threaten the success of this or like programs.

The era of "Send me a warm body, I'll train him" is over. The recession, closing off entry jobs and causing a surplus of available and experienced personnel, has brought on a problem of locating meaningful junior programmer jobs for the graduates of the program. Although the predicted economic expansion will reduce this problem, the recession has brought to light the lack of professional recognition and unclear career patterns for the personnel in the information processing field.

The present and future student is aware and skeptical of entering a program which may equip him for a nonexistent job. The publicity and the increased attention to sociological/health careers has caused a significant reduction of potential students.

The era produced a proliferation of two and four year programs in computing science, data processing, and programs with minors in these subjects. This levelled the enrollment at a lower figure than had been anticipated, endangering future programs. Educational institutions, more than ever, must offer a variety of modern programs, supported with up-to-date hardware systems and faculty, and change these programs to meet the future, a challenge which is very costly and risk prone.

To meet this challenge, information is needed. Information which is authentic and available that can be used by students, educators, employees, and employers. Too many decisions are based on one's limited environment, not always objective or timely. A paradox, in that most computing programs are developing personnel who are to participate in supplying objective and timely information.

Information which will be considered authentic must come from a national organization which has as its purpose developing information processing personnel. This organization would publish statistics, local and national, about personnel needs and qualifications in greater depth and degree than is presently distributed. A natural outgrowth of such an organization's purpose would be to promote recognition of information processing personnel,

to conduct research in information processing instructional systems, and to develop programs of studies.

The statistics would be used by students and their counselors in deciding about their choice of careers. Educators would use the data in providing needed programs. Employees woud be able to select and choose alternative educational programs for advancement. Employers would be able to provide professional development programs to meet their future needs.

Other functions the organization could serve is the promotion of professional recognition, seeking scholastic aid, and distributing programs, formal, inhouse, or intensive, with recommended instructional systems which would provide effective and efficient education.

This organization could also serve another needed educational and development function, regional training centers. These centers would equip personnel with locally needed qualifications. Personnel attending the centers would be recent graduates of college programs and in-service personnel temporarily relieved from their assignments. These centers would conduct intensive up to the minute training.

Hundreds of thousands future positions which are forecasted can only be filled by a national effort. If trends threatening this highly technical profession continue, the nation will face a shortage of qualified personnel and over supply of obsolete skilled personnel. Only a national organization can prevent another Apalachia.

Computer education at Orange Coast College—Problems and programs in the fourth phase

by ROBERT G. BISE

Orange Coast College
Costa Mesa, California

The business and industrial growth that has been associated with Orange County (California) speaks for itself. New and relocating firms representing the entire spectrum of technologies are moving into the cities of Costa Mesa and Newport Beach daily. The Coast Community College District, and especially the staff of the Business Division of Orange Coast College have continually developed educational programs to support this environment.

In the past period of shifting technologies, we have been supported by stable patterns of human behavior. As we plan for the next shift, we no longer find these stable patterns of human behavior. Instead we find only revolving fragmentations of the past and undefined forces that may be in the future.

In 1973, we are undertaking the development of viable programs and curriculum in the areas of computing and management information systems. In this we will be continually trying to integrate the experience and intuitive judgment that we have gained during a decade of total submersion in the changing forces of computer technology. We understand that the total environment in which we will make our attempt has the potential for treachery of the senses.

Charles Poore of the New York Times has labeled these times the Karate Age where with one quick and deadly assault, a man, a university, a regime or a nation may be sent writhing in agony.

A review of the technological changes that have taken place in computing quickly reveals that those who have been in the field of computing over the past decade had experienced "Future Shock" somewhat before Alvin Toffler coined the phrase. A brief history of computing at Orange Coast College will serve as a vehicle for reviewing these changes in computer technology. At the same time we may review the continuous process of curriculum redevelopment and the demands that were made of the instructional staff at Orange Coast College. The history may be seen as comprising three distinct phases.

PHASE I

In 1958 Orange Coast Community College entered into its first phase of data processing. At that time our equipment consisted solely of leased Electro-Mechanical Tabulating Equipment. To this, we added computing power in the form of a General Precision LGP30 with 4K drum memory and paper tape/typewriter input-output devices in 1959. The curriculum was designed around the available hardware to include courses of instruction in Electro-Mechanical Wiring Principles, Basic Concepts of Data Processing, the Electro-Mechanical Application of sorting, collating, reproducing, interpreting, tabulating and calculating. It also included programming in machine language on the LGP30. In addition students were required to study the principles of accounting, cost accounting, and accounting systems.

PHASE II

Phase II was initiated through the acquisition in 1963 of second-generation computing hardware systems in the form of IBM 1401 and 1620 computers with disk storage. The curriculum shifted to keep pace with the hardware. Although the principles of Electro-Mechanical Wiring and Tabulating equipment were retained, additional hands-on experiences were provided in machine language, SPS, and FORTRAN on both machines and COBOL and AUTOCODER on the 1401.

The principles of Accounting, Cost Accounting and Accounting Systems continued to be a part of the program and a new emphasis was initiated in Management Information Systems.

The objective of the two-year vocational program in data processing at this time was to develop qualified entrance-level tab operators and application programmers through hands-on experience.

The California Department of Vocational Education in conjunction with the Federal government provided assistance to the program in the form of grants for the development of curriculum and training of the instructional staff. With the rush by other institutions to provide programs in data processing and computer science, another dimension was added to the program in the summer of 1962.

In conjunction with the State of California Department of Vocational Education, a summer institute program for the intensive training and retraining of instructors in data

processing was initiated. This program was to become an on-going part of the total program.

With the active participation of the instructional staff in the training of others (and also of cross-training themselves) a sense of mastery over conditions developed. The frantic rush to keep up with hardware and programming sophistication seemed likely to be a condition of the past.

That sense of mastery was short-lived when in 1964 IBM changed the game from checkers to chess with their announcement of the System 360.

PHASE III

In 1966-67 the State of California underwrote a proposal to defray the costs of training two OCC instructors in third-generation programming and concepts. In return for this training, the staff agreed to the development of a detailed report containing all of the necessary educational ingredients to make the transition from second to third-generation computing.

This report was made available to all institutions. The curriculum by the fall of 1968 presented the concepts of 360 programming through an understanding of the programming languages of RPG, FORTRAN, COBOL, PL/1 and ALC.

The concepts of operating systems, file design, file management, and job control were integrated into the programming classes. Cost Accounting became an elective in the program and a course in Management Information Systems Projects became a requirement for graduation. The latter class was designed to provide students with the background necessary to function in their fast-developing role as staff consultants to line management at all levels.

Through the generous contribution by Hunts Foods of computing time on their 360, we were able to introduce a third-generation curriculum in the spring of 1967. Third-generation computing hardware was available at the college by November of 1968 (IBM System 360/40). In January of 1969 teleprocessing terminals were added using APL as the computer language. There was one factor upon which we all agreed after the hectic year of 1969: one was only kidding oneself if he found security in technological expertise.

The concepts of the third generation increased the need for summer institute programs for the retraining of educators in the field, and the college offered the first summer institute in third generation programming in the summer of 1969.

Quickly we became aware of the fact that where in Phase II we were involved in a simple vocational program, with the sophistications of third generation, higher aptitudes, wider perspective, and greater perseverance would be required of the student. We could no longer provide mere vocational education but had to be involved in providing some measure of professional education and training. The offers that our graduates were receiving from the labor market required them to possess a much keener insight into the realities of the business environment and demanded a strong understanding of the organization and the part the computer played in the organization.

In the summer of 1970 our new facility was completed which doubled our capacity. We now had a designated room for our IBM 029 keypunches and IBM 2741 teleprocessing terminals. We attempted to maintain our philosophy of hands-on training through a student/reader/printer and the addition to our curriculum of a hands-on course in computer operation.

The program for the development of computer-assisted instruction initiated in 1969 necessitated the acquisition of an IBM 360/50 DOS System in the fall of 1970. The college having expanded to two colleges in 1965, changed the name of its district to the Coast Community College District in 1970. Through the foresight of the district administration, a policy of decentralizing computing power was implemented through the placement of the teleprocessing terminals throughout both campuses. This included the use of dial-up teleprocessing terminals. Both the added use of computing throughout both colleges and the additional administrative requirements to implement program budgeting systems allowed the Business Information Systems instructional program to receive the benefit of more sophisticated hardware systems.

The IBM 360/50 DOS system could not meet the demands for the additional computing requirements, and a change was made from DOS to OS with one megabyte of low-speed core in 1971. Through the use of CRT terminals a student file inquiry system became operational in 1972. This necessitated a further upgrading of the system to an IBM 370/155 OS MFT containing one megabyte of main memory.

With the two year program arriving at a somewhat stable position, new emphasis was placed upon developing courses of instruction to service the other disciplines of the college and to integrate all disciplines with the sense of the rapidly increasing rate of technological change. The ability to adapt was emphasized. Two courses were designed to meet this objective. A course of instruction using the language of FORTRAN and APL was developed to integrate programming concepts and applications with the respective discipline of the prospective transfer student to the four year college. Another course was developed using the interactive teleprocessing language of APL to provide instruction to all students of the campus.

With the changing of emphasis in the computing field came requests from the computing community for additional courses in Computer Operations, Data Communications Systems, Management of the Computer Effort, Operating Systems, and most recently Advanced COBOL. In order to further meet the needs of the rapidly-growing business environment, two one-day seminars were held in the areas of Management and the Computer and Data Communications for Management. We also held a two-day seminar for a visiting Japanese top-management

group. The title of this seminar was the use of computing by American managers.

Since September of this year we have been involved in the evaluation of our total curriculum and have attempted to make our program more flexible to the three basic student groups that we serve.

The first group is comprised of an increasing number of students who are transferring to four year colleges to complete their education.

Most of these four year colleges do not have as wide an offering of courses, and those that are offered are at the upper division level. Consequently, students must use much of their course work in Business Information Systems taken at our institution to fulfill elective lower-division courses. We have been able to obtain some relief from this problem through "one-to-one" articulation on an individual college basis, but this is a nagging problem causing a great deal of frustration to the student.

The second group we serve is that of the two year terminal student. These students can be classified into two general categories: those with a good aptitude for programming and systems work and those that have average aptitude and strong desire. We feel that the higher aptitude student would benefit by taking more advanced work in programming and systems. For the second group of students we see very fulfilling careers in the area of computer operations and possibly computer sales and allied fields. We encourage members of this group to take courses in computer operations and to broaden their general understanding of the field.

The third group is comprised of practicing professionals in the computer field, and managers and staff people from various fields of business. For this group we have added courses in Data Communications Systems, Managing the Computer Programming Effort, Advanced COBOL and Operating Systems.

In our attempt to meet the needs of these three basic segments of our student population, we have devised what we feel to be the basic minimum core requirements for our students.

The core requirements are intended to develop the technical base necessary to compete in the dynamic information and computer industry and in addition to provide each student with a macro view of the environment in which the function of computing is performed. We attempt to accomplish this through nineteen units of required courses consisting of Introduction to the Concepts of Information Systems, COBOL and PL/1, Assembly Language Coding, Management Information Systems and a Management Information Systems Projects class. Eight additional units are required in Accounting or in Calculus, and nine additional units are required from a group consisting of: Advanced COBOL, Computer Operations, RPG, Data Communications Systems, Managing the Programming Effort, FORTRAN/APL, Computer Science, Operating Systems, APL, Cost Accounting and Managerial Mathematics.

FACTORS TO BE CONSIDERED IN THE IMPENDING PHASE IV

The manufacturers promised that they would never do anything to us like they did with the complete change in architecture in 1964, but somebody forgot to get it in writing from the usurpers of the industry, that forward and vital mini-computer industry. Will the Volkswagen of the computer industry, the mini, make the big *one* of the computing field alter its competitive path? We can only wait and see. One thing we are sure of is that the Mini-Computer, Data Communications, Teleprocessing and Data-Based Management Systems are upon us.

We are told that the next major thrust of computing will be in manufacturing systems and the language of computing is going to be eventually reduced to the level of the user through the terminals and CRT. This is the picture of the 70's and we are told by John Diebold that the 80's will usher in the Cybernetic System in "intelligent machines," where Japan has every intention of dominating the market.

Before we attempt to define the problem of developing the curriculum for the last 1970's and 80's we might benefit by reviewing our societal framework over the past ten years or so.

The social upheaval over these recent years has shaken our institution to the very mantle of our earth.

The Civil Rights sit-ins in Greensboro, North Carolina, in 1960, were followed in 1963 by Martin Luther King's "I have a dream" speech to 200,000 civil rights demonstrators in Washington, D.C.

Polarization of social and political values were thereafter punctuated by an infamous series of assassinations and attempted assassinations. The Free Speech Movement at Berkeley in 1964 was followed by the Viet Nam protest from 1967 to the inauguration of the President on January 20th of this year. The energy of dissatisfaction and discontent has been registered through the vast disenchantment with our industrial military complex and the expenditure of great sums of money for space exploration. The result of all this has been that technology has been identified as one of the major sources of our society's problem.

The War on Poverty Program in the early 60's and the concern for the environment and health of our citizens brought about a new sense of social consciousness nonexistent in previous periods.

The dethroning of college president after college president because of a total inability to grasp what was taking place and make the required changes drove the point even deeper.

Suddenly in 1969 and 1970 a lionized profession (engineering) of the 1950's and 1960's suddenly found itself largely obsolete and unwanted. Thus a profession found itself in the same position that the blue collar worker had been faced with for decades.

Students following the path of success established by our society, acquired the training and education suppos-

edly designed to provide them with the "good life" of the future. The shock they received when they attempted to enter a labor market that could not utilize their skills, and an environment they did not understand destroyed much of their confidence in the ability of our economic system to meet the needs of the people.

The computer industry leaped off of the solid economic base established in 1958, and with the other industries of our economy grew rapidly during the early and mid-sixties. The constant pressure of supporting the war in Viet Nam and meeting the unprecedented demands at home finally forced our economy into a heated period of inflation and the eventual recession of 1969 and 1970. The fixed costs of computing were finally registered upon a management that had grown up in years of unparalleled growth.

Hopefully the recent fight for survival experienced by management has provided the necessary insights into what courses of action management is to take if we are not to repeat the mistakes of the 1960's. Whether management has been able to work through the archetypes of management past and sense the new needs of its employees only time will tell.

One thing seems certain, organizational needs are not yet synchronized with human needs and the pace of technology will only widen the gap. It appears that management does not know how to reward its employees for productive efforts within the framework of the new social consciousness.

To sense a real problem we have only to listen to personnel executives on one side lamenting the fact they are unable to find employees who can fit quickly into the work organization and become productive. On the other side, these same personnel experts are admonishing educators for developing people for a work environment that cannot adequately utilize their skills, thus bringing about employee dissatisfaction and turnover. There appears to be a mutual fuzziness both on the part of the executives defining the specifications of required skills for the near future and the part of the educator attempting to educate with such specifications in mind.

The atrophy that sets in as a misplaced individual exists in a state of unrelieved boredom only furthers the loss of identity and therefore raises frustration to a dangerous level. An impersonalization of organization that grows through a management strategy of merger and acquisition frequently spawns a hostile enemy behind an employee's mask of acceptance. Management will be using the computer to ever-increasing degrees to eliminate specific human procedures. However, it seems probable that for every problem solved in this too-obvious manner, there may be created a dozen more, for the approach ignores the basic root structure of men's needs.

All of the foregoing societal events that have transpired over the past decade have contributed two vital factors:

(1) There is a definite sense of social consciousness and a definite desire for real freedom. The Civil Rights Movement and the course of events that followed released untold amounts of human energy that is far from being coordinated in tandem.
(2) The power of our present and near future technology gives us unlimited capacity for the solution of high priority problems of our world.

Alone this technical competence is useless unless interwoven with the tapestry of human understanding. Such a process undertakes what Warren Bennis has referred to as the process of human revitalization. He identified the following four basic points in the process.

(1) An ability to learn from experience and to codify, store and retrieve the resultant knowledge.
(2) An ability to learn how to learn, the ability to develop one's own methods for improving the learning process.
(3) An ability to acquire and use feedback mechanisms on performance to become self-analytical.
(4) An ability to direct one's own destiny.

The program and curricula of the late 70's and 80's must especially develop the students' ability to learn how to learn and to direct their own destinies. It is difficult to perceive how such programs and curricula can be successful without the practice and consistent involvement of the business community, in both the development and implementation.

Sharp distinctions between campus and business arenas are already dulling. Work experience programs, on-site educational programs, educational TV and programmed instruction technology and concepts have made significant advances, and have an undeniable future. All we seem to need is the sense of urgency that will cause us to allocate resources toward the realistic assessment of the situation. Effective definition of objectives will require mutual contributions of time and intellectual resources on the part of both business and educational leaders.

Our problem today is one of breaking down our own archetypes and the archetypes of our institutions in order to develop those inner human qualities that men must integrate with future technologies.

Computing at Central Texas College

by ALTON W. ASHWORTH, JR.

Central Texas College
Kileen, Texas

ABSTRACT

Central Texas College has developed a post secondary curriculum in data processing in conjunction with the United States Office of Education. The program has been developed around the career education guidelines established by the United States Office of Education. The following list of program advantages will be discussed in some detail at the June meeting:

1. A complete unit of learning has been provided for the student in his first year and in his second year. At the end of his first year he will have received useful skills that are saleable in the market place. During the first year he will have had a balance of data processing courses, mathematics, business practices and effective communications. These subjects, combined with the learning of a basic programming language and systems analysis, will qualify him for many of the collateral jobs that exist in a data processing environment. He will have learned some advanced programming languages. He will have had applications courses. He will have learned some of the internal workings of the computers and programming. He will have been exposed to data management systems and transmission techniques providing him with an insight into the future of data processing. He will have had an elective during his last semester that could be an industry co-op program.
2. The curriculum is flexible enough so that the student will be able to change his educational objectives to a four year program without extensive loss of credit.
3. Through the new organization of courses, certain social and business objectives have been met as well as those of data processing. At specific points during education, well rounded educational objectives have been met.
4. A balance of traditional courses and special computer oriented courses exist between his two years of education. He will receive five data processing courses his first year and five data processing courses his second year, plus his elective co-op program with industry.
5. A balance of programming languages has been provided the student for his first and second year education. He will learn two programming languages his first, BASIC AND COBOL, and two programming languages his second year, FORTRAN and ASSEMBLY.
6. The curriculum is designed to develop people to become working members of society. In addition to data processing capabilities, communications skills and social awareness development courses have been provided.
7. Sufficient math has been provided in the curriculum to allow the student to advance his own studies of data processing after leaving school.
8. Considerable applications experience has been gained in both the educational and working environments.

The design of IBM OS/VS2 release 2

by A. L. SCHERR

International Business Machines Corporation
Poughkeepsie, New York

INTRODUCTION

The purpose of this paper is to give some insight into the design of IBM OS/VS2, rather than cover individual features of the release. Included are the overall objectives for the design, some of the system's key architectural features, and how these relate to the environments that the system is intended to be used in. The major objective is to show how the design of the system fits together and to provide an insight into the rationale of the design.

OBJECTIVES

Release 2 represents a major revision of OS to provide a new base for future application areas. The key thrust is to provide a new SCP base with increased orientation toward DB/DC applications and the additional requirements placed on an operating system because of them. Another key goal of the system is to support multiple applications concurrently in a single complex. This complex may include multiple CPUs, loosely or tightly coupled. The system must dynamically adjust itself to the changing loads in the various environments that it supports, as well as provide increased security and greater insulation from errors.

Maintaining a high level of compatibility continues to be a major objective for VS2. Extending the system, adding function, and changing its internal structure, while at the same time considering compatibility, represented a significant challenge to the designers of Release 2. Over the last few years we have learned a lot about the needs of our users, and during this time, the state-of-the-art in software design has moved forward. The system has been reoriented to incorporate these things into the system.

USE OF VIRTUAL STORAGE

The incorporation of virtual storage into OS has allowed the system to support programs whose size is larger than available real main storage. There are operational advantages; however, significant additional benefits can be realized. Using virtual storage to provide for an extremely large address space, allows program structures to be simpler, intermediate data files to be eliminated, and real main storage to be used to hold data that in the past was resident on a direct access device. This latter use can result in a significant performance advantage which will be discussed later.

MULTIPLE ADDRESS SPACES

Perhaps the most obvious new feature of Release 2 is the support of multiple address spaces. Each job step, TSO user, and operator STARTed program in the system has a private address space that is 16 million bytes, less the space taken by the operating system. Figure 1 is a comparison between Release 1 and Release 2 storage maps. Both maps extend from 0 to 16 million bytes. Release 1 and MVT actually look alike, with the only difference being that MVT's address space is limited to the size of real storage.

The Release 1 map, shows two TSO regions with several users in each. Users A and B, for example, cannot be in execution at the same time because only one of these users can occupy the region at a time. The others are swapped out. The transition from Release 1 to Release 2 can be understood very simply by considering the Release 1 system with a single TSO region the size of the total available virtual storage. What has been done in Release 2 is to remove the multiprogramming restriction between the users of the TSO region. On the other hand, Release 2 does not allow two jobs to share the same address space. One of the first implications of this design is that it is no longer necessary for the operator to get storage maps printed at the console so that he can manage main storage.

To show the effect of multiple address spaces on certain control program functions, TCAM will be used as an example. In Release 1, terminal input is read through a channel directly into the TCAM region. There it undergoes some processing and is then moved to the user's region or to the TSO control region. In Release 2, the input is read into a common system area buffer at the top of the map, and from there is transmitted under TCAM's control to the user. To programs that have done inter-

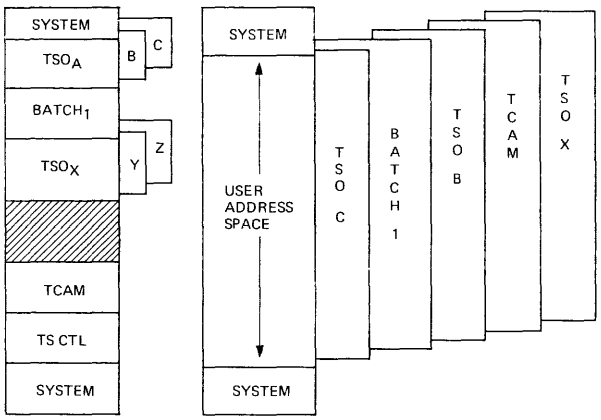


Figure 1—Multiple address spaces

region communication in previous systems, this new storage map represents a major difference.

In Release 2 V=R jobs no longer affect the virtual address space of V=V jobs. Since each job is assigned a 16 million byte address range, V=R jobs only affect the amount of real storage available. (See Figure 2).

STORAGE MAP

Figure 3 shows the storage map seen by a single job. This corresponds to an MVT or Release 1 region. At the top of the map in areas which are commonly addressable by all of the address spaces is the System Queue Area containing system control blocks, the pageable Link Pack Area, and the Common System Area for use in communicating between users. This area is used, for example, by TCAM and IMS for inter-region communication. At the bottom of the map is the Nucleus and that part of Link Pack Area which is to remain permanently in main storage.

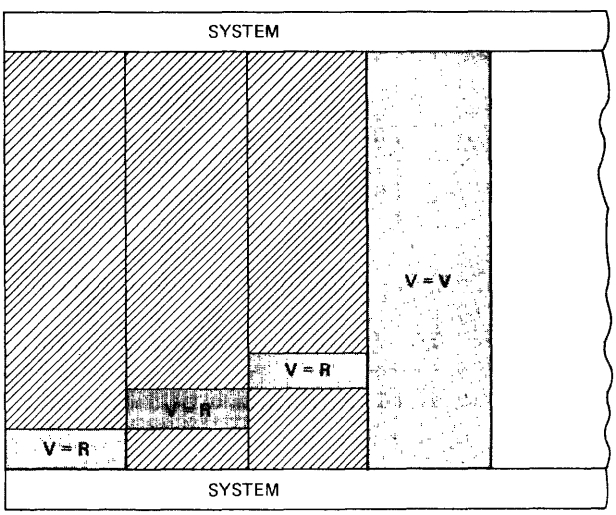


Figure 2—V=R, V=V

The area in between is the private address space for each user. User requests for storage in all subpools are allocated from the bottom of this private address space. Requests for Local Supervisor Queue Area and the Scheduler Work Area storage are satisfied from the top.

COMPATIBILITY

Compatibility is a major objective in Release 2. Object code and load modules from MVT and VS2 Release 1, not dependent on the internal structure of the system, will run with Release 2. JCL compatibility is maintained, and

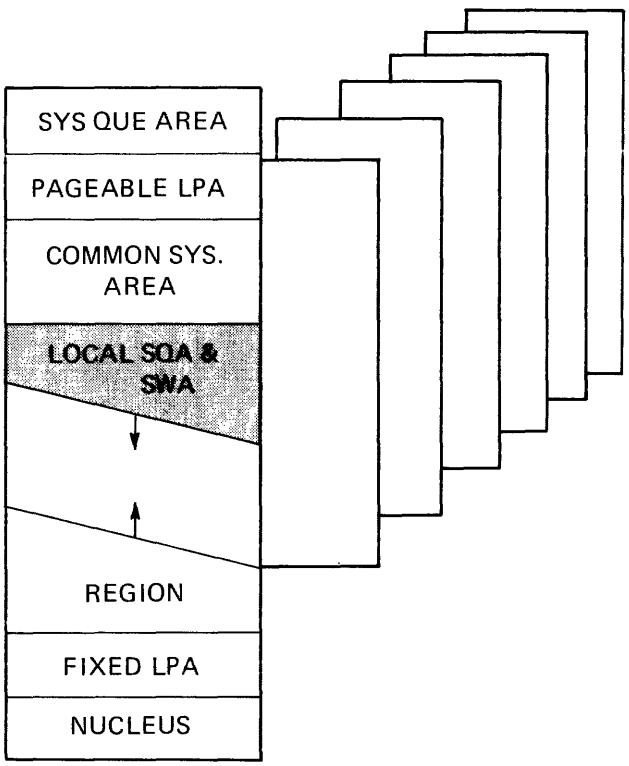


Figure 3—Storage map

the data sets and access methods of previous releases apply, as well as EXCP. SMF is compatible as well. However, it must be recognized that in moving from a non-virtual to a virtual environment, the usefulness of some of the measurements has changed; and in order to account completely for usage, it may be necessary to make some use of the new measurements that are provided.

Internal interfaces are the area of greatest concern because, in some cases, such interfaces have been extensively used. Generally, our approach has been to evaluate every change of this type to see what the effect is on the user community as well as our program products. Several proposed changes were not made because of their potential impact; but, on the other hand, some change is

required to make progress, and thus we have had to consider many difficult trade-offs.

The major differences that affect compatibility include the system catalog, which is now a VSAM based data set and requires conversion from the catalogs of existing systems. Forward and backward conversion utilities have been provided, as well as compatibility interfaces allowing the use of the original OS catalog macros. As mentioned earlier, the new storage map, will impact programs that have done inter-region communications. Also, IOS appendages run enabled in Release 2 and must use a new synchronization mechanism. Therefore, there is likely to be impact to user-written IOS appendages.

PARALLELISM

One of our major design goals in the system was to provide for as much parallelism of operation as possible. The reduction of software bottlenecks that prevented efficient multiprogramming is the major technique that we used. Listed are five of the main areas that we worked in. Each of these areas will be discussed.

- Job Queue
- Allocation
- Catalog
- TSO Region
- MP65 Disable Lock

Experienced OS users will recognize these as areas with a high potential for improvement.

JOB QUEUE

We have eliminated the Job Queue data set that OS has used since its beginning. With HASP or ASP in an OS system, there were really two job queues—the one kept by the support system relating primarily to information required to schedule jobs and the printing of output, and the OS job queue which contains similar information as well as information pertaining only to a job in execution. One type of information is really for inter-region communication between various parts of the scheduling functions of the system; the other, for intra-region communication between the scheduling components and data management in behalf of the executing job.

The inter-region information has now been placed entirely in the job queue maintained by the job entry subsystem, either JES2 or JES3. The intra-region information has been segmented and placed into the individual job's address space. In this way, the portions of the original OS job queue having the highest usage are now in each job's private address space. The less frequently used information relating to communication between various components of the scheduling function is now in the JES job queue. Thus, all of these elements of the job queue can be accessed in parallel. The JES job queue is also used to journal information required to restart jobs during warmstart or from a checkpoint. (See Figure 4.)

ALLOCATION

The component of the system that does data set and device allocation has been completely redesigned. Both batch and dynamic allocation are now supported by the same code and provide essentially the same function. The design is oriented toward virtual storage—no overlays are used, and all work areas are in virtual storage. Allocation of data sets to storage or public volumes can be done completely in parallel, regardless of other allocation activity.

This type of allocation represents probably the most common form in most installations, and, in general, the design of the new allocation provides shorter paths for these generally simpler cases. When it is necessary to allocate a device and perform volume mounting, these requests are serialized by device group. Therefore, a request for a disk need not be held up because another job is waiting for a card reader. Other improvements in this area include the ability to prevent a job from holding devices until its entire requirement can be met, and the ability to cancel a job waiting for devices.

CATALOG

The catalog has been converted to an indexed VSAM data set, primarily to allow for faster access to a large catalog. The curves in Figure 5 give the general idea of how access time should relate to catalog size with this new structure.

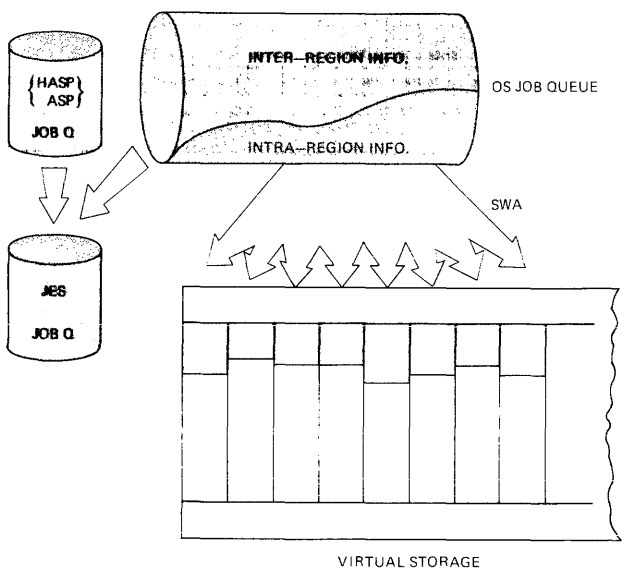


Figure 4—Job queue

- INDEXED VSAM DATA SET
- DESIGNED FOR FAST ACCESS TO A LARGE CATALOG

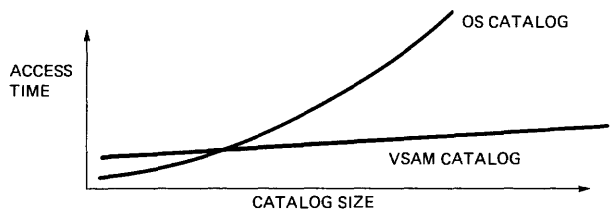


Figure 5—Catalog

TSO REGION

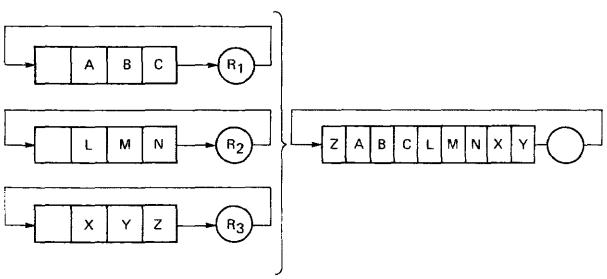
As previously stated, in MVT or Release 1, TSO users sharing the same region cannot be concurrently in execution. This restriction is eliminated in Release 2. Therefore, the situation shifts from one where each region serves a given set of users, to one where the entire system serves all of the users. Thus, any potential imbalance between regions is eliminated. (See Figure 6.)

Moreover, previous support placed a limit on the level of multiprogramming for TSO at the number of TSO regions. In Release 2, the level of multiprogramming can vary and is dependent upon the load placed on the system.

LOCKS

In a tightly-coupled multiprocessing system, it is highly desirable from a performance point of view to allow the control program to be executed simultaneously on both CPU's. However, some means is then required to synchronize or serialize the use of control information used by the control program.

System code in MVT disabled for interrupts prior to the updating or use of this type of control information; and when the operation was completed, the system was enabled. The MVT technique used for Model 65 multiprocessing was to use one lock which prevented both CPU's from being disabled at the same time. In environments with heavy usage of control program services, this lock becomes a significant performance bottleneck. (See Figure 7.)



- ELIMINATES:
  - UNBALANCE BETWEEN REGIONS
  - LIMIT ON LEVEL OF TSO MULTIPROGRAMMING

Figure 6—TSO region

In the Release 2 support of MP, we have used instead a number of specific locks, each relating to a particular function. Generally, the program obtains the appropriate lock relating to the data that it is going to update or use, performs the operation, and then frees the lock. Whether or not the system is disabled during this operation depends on whether or not interrupts can be handled.

The locks that are used include one per address space, a dispatcher lock, multiple IOS locks, a lock for real storage management, locks for global supervisor services, and locks for virtual storage management. This means that, for example, a GETMAIN can be performed in a user's

- MP 65 TECHNIQUE: ONE LOCK

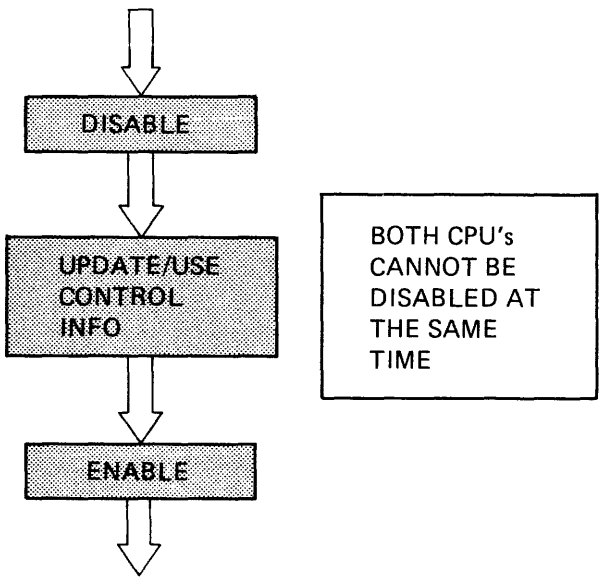


Figure 7—Locks

private address space at the same time that another GETMAIN is being done in another user's space, or an interrupt is handled by IOS. The net result is that the system is enabled for interrupts more often and more elements of the control program can execute in parallel. The primary advantages here are to a tightly-coupled multiprocessing system, but some of these carry over into other environments. (See Figure 8.)

MAIN STORAGE EXPLOITATION

Because of recent changes in the relative costs of various hardware components, the trade-off between main storage usage and other activity in the system has changed. In Release 2, our goal has been to exploit main storage by trading it for CPU and I/O activity wherever possible.

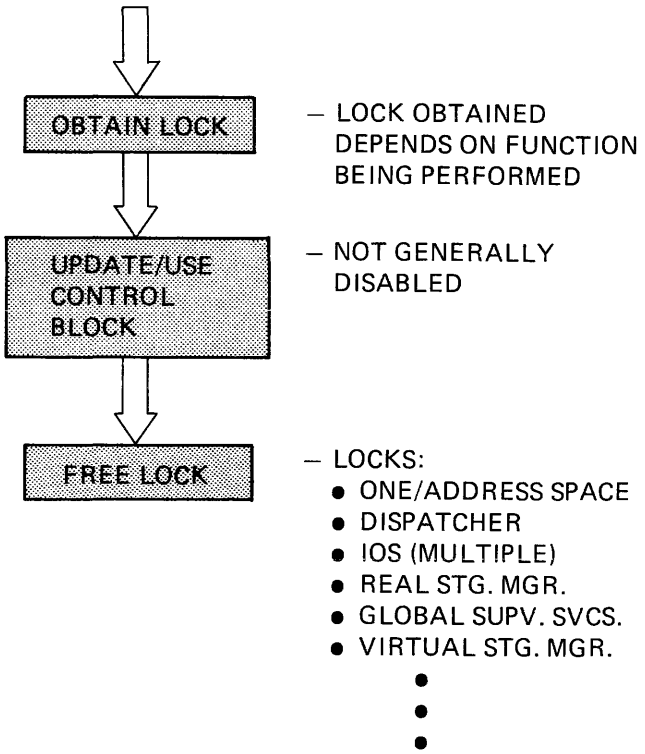


Figure 8—VS2/Rel 2 uses multiple locks

In MVT and VS2 Release 1, data sets are generally placed on a device and all access to this data must go to that device. Main storage content is limited to the images of programs.

Certainly, in many environments there is data whose usage is high enough to warrant at least a part of it being resident on a higher speed device or perhaps in main storage. In fact, there are environments where some blocks of data receive higher usage than some of the pages of the program, and ideally should receive preference for main storage occupancy. In Release 2, we have attempted to move in the direction of allowing data to be placed in the storage hierarchy dynamically, according to its usage. Therefore, certain data can be resident in main storage or on a higher speed device if it has high enough frequency of usage.

The whole idea is to allow more data, more information, to be resident in main storage. Thus, given a fixed amount of main storage, there is a better choice as to what to put there. More importantly, given more main storage, there is more useful information to put into it.

In Release 2 there are three facilities for this type of exploitation of main storage: virtual I/O, Scheduler Work Areas, and the large address spaces.

VIRTUAL I/O

Virtual I/O provides for placing data sets in the paging hierarchy. The net result of this is that if a page of data is resident in main storage, there is a reduction in I/O and CPU time. The CPU time is reduced because of the elimination of I/O interrupt handling, channel scheduling, and task switching. Because blocking is done automatically at 4K, greater efficiency may result. When I/O is done, it is performed by the paging mechanism, with generally more efficiency than with the conventional techniques.

An additional advantage of virtual I/O is that no direct access device space management is required, and therefore allocation time is faster. Because space is allocated in 4K blocks as needed, space utilization is also more efficient.

In Release 2, temporary data sets are supported for virtual I/O in a compatible way. No JCL or program changes are required for SAM, PAM, DAM, XDAP, and the equivalent operations in EXCP. Any program dependencies on direct access device characteristics are handled in a transparent way.

SCHEDULER WORK AREA

The Scheduler Work Area allows a job's job queue information to be contained in its own virtual storage. Thus access times are better for this information when it is required for allocation, termination, or OPEN/CLOSE-End of Volume processing. If usage is high enough, this information would be resident in main storage with the same advantages as with virtual I/O.

LARGE ADDRESS SPACES

The use of large address spaces to achieve greater performance has been described exhaustively in other places, however, several techniques which have been incorporated into portions of the control program should be highlighted. Overlay structures have been eliminated, and the use of the Overlay Supervisor, LINK, and XCTL services has been removed with a decrease in I/O activity as well as CPU time. *Spill files* have been eliminated; instead, large work areas in virtual storage have been used. The allocation redesign makes use of both of these techniques.

RESOURCE MANAGEMENT

In the resource management area, our goal has been to centralize all of the major resource control algorithms. The objective here is to achieve better coordination than is possible with decentralized algorithms. With a decentralized design, two uncoordinated algorithms can sometimes work at cross purposes. By having a centralized set of algorithms, more opportunity exists for optimization.

The system resource manager in Release 2 replaces the TSO driver, the I/O load balancing algorithm of Release 1, and HASP's heuristic dispatching. Further, it provides a new algorithm to control paging and prevent thrashing

by dynamically adjusting the level of multiprogramming. The rate at which users get service is controlled by the Workload Manager in accordance with installation specified parameters.

WORKLOAD MANAGEMENT

Priorities for this Workload Manager are not absolute, but rather are expressed in terms of a rate of service for each job. This allows a departure from the usual situation where a lower priority job gets only what is left over after the higher priority jobs have received all of the service they can get. In Release 2, two jobs can proceed at a relative rate of progress that can be set by the installation. These service rates are specified for different system loads so that the relative rate of service received by two jobs can change as the overall system load shifts. Finally, service rates can be specified for a given set of users or jobs, where a set can include as few as one user.

Figure 9 shows a sample of how this is done. There are five sets of users, A through E; and service rates varying from 0 to 1,000 service units per second. Service is expressed in terms of a linear combination of CPU time, I/O services, and main storage use. The number 1 curve, which might be considered for a light load, shows the users in groups A and B receiving high service rates, users in groups C and D slightly less service, and E even less. User sets A and B might be two types of TSO users, C and D, high turnaround requirement batch jobs; and E the rest of the batch jobs.

As the load gets heavier, the installation has specified that they would like more of the degradation to apply to the users in Sets D and E, and the least degradation to apply to sets A and B. Curve 4 represents the heaviest load where users in set A get significantly better service than anyone else, and users in sets C through E receive only what is left. The system attempts to operate on the lowest numbered curve; however, as the load gets heavier, it degrades the service seen by each of the sets of users proportionally to the way shown by the curves. That is, in going from curve 1 to curve 2, it reduces the service seen by users in category C more than for category A.

A set of reports is produced which the installation can use to determine the response time or turnaround time

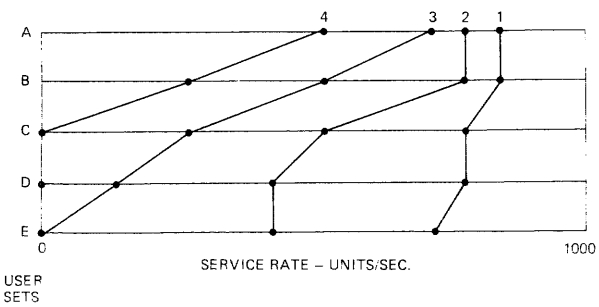


Figure 9—Workload management

and throughput that is being produced by the system for each user set. Should an adjustment be required, a higher rate of service specified for a set of users will yield better response time or turnaround time. Our objective here is to provide a relatively simple way to achieve discrimination between users and to provide the right level of service to each group of users.

RECOVERY

Recovery mechanisms in the system have also been overhauled in a major way. A significant amount of work has been done in this area. Our goal is to contain errors to the lowest possible level, and either to recover from an error so that the system can proceed as if the error never occurred, or at least to clean up so that the effect of the error is not felt outside of the function in which it occurred. In this area we have really recognized that it is not enough to have code with a minimum number of bugs, but rather to have a system that minimizes the effect of the failures that do occur.

The same approach for minimizing software failures is used for hardware error recovery as well, especially in the multiprocessing environment. Generally, the method is to provide specialized recovery routines that operate as a part of the main line functions, and which receive control whenever an error is detected by the system. There are approximately 500 such recovery routines in Release 2.

INTEGRITY

In Release 2 we have closed all of the known integrity loopholes in VS2. This means that unauthorized access or use of system facilities and data or user data, is prevented, both for accidental as well as intentional actions, and we will now accept APARs for integrity failures. Integrity is a prerequisite for adequate security, where security is defined as an authorization mechanism to distinguish between what various users can do. Moreover, integrity should also provide for an increased level of reliability.

SERVICE MANAGER

In Release 2, we have provided a new transaction-oriented dispatching mechanism which allows the efficient creation of new units of multiprogramming. Our goal here was to increase performance by trading off function. This new unit of multiprogramming differs from the OS task in that it is not a unit of resource ownership or recovery. The new facility, called the Service Manager, is used by the Release 2 supervisor, JES3, IOS, VTAM, and the version of IMS for use with VS2 Release 2. This mechanism can also be used by appropriately authorized user programs. For example, a VTAM application.

RELEASE 2 PERFORMANCE

Summarizing what has been done in Release 2 from a performance standpoint the following points are noteworthy. Because of the reduction in serialization and the tradeoffs that can be made between I/O activity and main storage, the system can better utilize the CPU.

Figure 10 shows conceptually the CPU and I/O overlap for an MVT job. The wait state time is comprised of I/O wait plus other waits caused by serialization on system resources. These wait times are reduced as a result of virtual I/O, scheduler work area, the new catalog, allocation, etc. However, this wait time may be extended due to paging. This is typically rather small, especially in a large main storage environment.

On the other hand, CPU time generally will be reduced as a result of virtual I/O activity since fewer interrupts are handled, etc. Other overhead is also reduced because the reduction in I/O and wait time generally allows the CPU to be fully utilized at a lower level of multiprogramming. On the negative side is degradation due to extra instructions required in Release 2 because of enhanced recovery, integrity, and new function. The overall effect is that individual jobs tend to look more CPU bound.

The general performance characteristics of Release 2 are significantly different than previous OS systems. The system now is generally more responsive, in that there is better consistency, with fewer long responses caused by the processing requirements of other jobs and the operator. Because fewer initiators can be used, and because of the reduction in bottlenecks, batch turnaround time can be improved. And, with the System Resource Manager, the installation has more control over the service seen by an individual user or job.

VS2 RELEASE 2 ENVIRONMENTS

The following summary shows how the features of VS2 Release 2 apply to various environments, such as

Multiple large applications,
Data base/data communications,
Time sharing,
Batch,
Multiprocessing, and finally,
Operations.

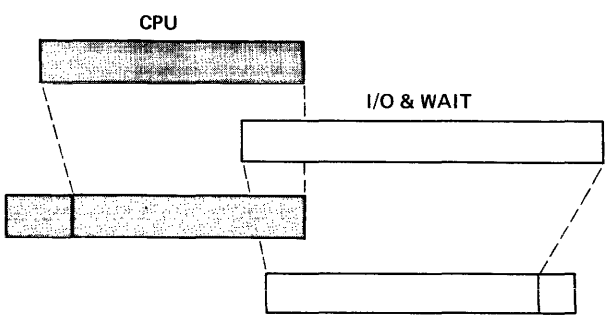


Figure 10—CPU and I/O overlap for an MVT job

MULTIPLE APPLICATIONS

One of our major goals is to allow multiple applications to operate effectively in a single system complex. This is theoretically highly desirable, but previous operating systems have had insufficient capabilities to shift resources dynamically from one application to another as the load changed. Perhaps even more important, failures in one application often brought down other applications, or even the entire system. There was also insufficient separation of applications from a security point of view. Release 2 provides both better isolation and integrity to address these problems. With virtual storage and other facilities in Release 2, more dynamic control and use of resources is also possible.

TELEPROCESSING

In the teleprocessing area, Release 2 is intended to be a base for high performance data base/data communications applications. VSAM, VTAM, Service Manager, Virtual I/O, Large Address Spaces, and the new Allocation all provide tools for such applications.

TIME SHARING (TSO)

For time sharing, a number of performance improvements have been made: SWA, the Catalog, etc. Compatibility between TSO and other areas of the system is more complete, primarily because the rest of the system is now more like TSO. Dynamic device allocation with volume mounting represents a new facility for TSO users that are authorized by the installation. SYSOUT data can be routed through JES2 or JES3 to a remote high speed work station to provide bulk output for a TSO user. Finally, large address spaces have been classically considered a time sharing function.

BATCH PROGRAMS

In the batch area there are a number of performance improvements as well. Dynamic data set and device allocation is provided for the first time for the batch programs. Among other things, this allows the ability to start printing SYSOUT data sets dynamically prior to the end of the job. This can be done with a minimal change to the JCL and with no programming change. Remote job entry is provided through the JES2 and the JES3 packages.

MULTIPROCESSING

Multiprocessing has traditionally placed a heavy emphasis both on reliability and availability as well as

performance. In the reliability area, a number of hardware improvements have been made. Certainly the increased integrity, both between the operating system and the user, as well as between the various parts of the control program, provides the potential for better reliability. Most important are the new recovery mechanisms in Release 2.

In the performance area, the complexity of a multiprogramming system is generally increased in the MP environment; however, the facilities for increased multiprogramming efficiency in Release 2 go a long way toward achieving good performance on MP systems. The exploitation of main storage is also important, since most MP systems are configured with large amounts of main storage. The multiple locks of Release 2 are aimed directly at minimizing contention for control program usage of the CPU's in a tightly coupled multiprocessing system.

OPERATIONAL CHARACTERISTICS

On the operational side of the system, our goal has been to have less dependence on the operator for performance. Generally, the system is significantly less serialized on the operators and their activities. The system, we feel, is simpler to operate. Tuning should be significantly easier as well. There are fewer bottlenecks to balance, fewer parameters to specify, and the system is more self-tuning. Moreover, the system can allow for more control over its own operation with the new integrity facilities, the System Resource Manager, and so on.

CONCLUSION

The purpose of this paper has been to provide some insight into how we arrived at the design of Release 2. Our objective was to provide some very significant increases in function and availability, with improved performance characteristics, and with a high degree of compatibility to previous OS systems.

We think that the system has done a good job of meeting these often conflicting objectives. OS/VS2 Release 2 represents a major step forward, but it is only a first step, since it provides the base on which we will build total support for advanced applications in the 1970's.

IBM OS/VS1—An evolutionary growth system

by T. F. WHEELER, JR.

International Business Machines Corporation
Endicott, New York

INTRODUCTION

A brief study of IBM OS/VS1 (Operating System/Virtual Storage 1) will reveal a system providing many faceted growth capabilities at all levels of user-system interaction. Additional meaningful function is provided on a stabilized base to assure this growth capability. It can be further seen that installation growth is achieved through new application work and not by a continual rework of existing programs. To assure the users ability to move to new work almost immediately, OS/VS1 is built on an IBM OS/MFT (Operating System/Multiprogramming with a Fixed Number of Tasks) base. Compatibility is defined to extend to most object programs, source programs, data and libraries from OS/MFT to OS/VS1, thus assuring a normal movement of existing programs to the virtual environment. Figure 1 graphically represents the areas of change between MFT and VS1.

In like manner, the transitional problem of education is greatly reduced for the programmer and operator alike. VS1 uses the MFT languages in all areas of programmer/operator contact to the system, and from the system generation procedure to the operator control language, VS1 incorporates, improves and extends the existing MFT language to support the virtual function.

As an OS compatible system VS1 becomes a vital part of the IBM family of new virtual systems which includes DOS/VS (Disk Operating System/Virtual Storage), OS/VS1, OS/VS2 and VM/370 (Virtual Machine/370). Each is based upon its predecessor system but each expands the horizon of support with virtual memory.

The virtual storage facility is the single most important new characteristic of VS1. It offers significantly longer address space for both application partitions and system functions by providing, with adequate equipment, a 16 million-byte addressing capability.

To provide this enhanced capability, OS/VS1 requires a System/370 central processing unit with the dynamic address translation facility. VS1 supports this facility on the System/370 Models 135, 145, 158, 168, and those 155's and 165's which have the DAT facility field installed. In addition to the hardware facility, significant changes were made to control programs code which I shall discuss later in this paper.

Significant enhancement was made to the job scheduling algorithms. The single most important addition has been the incorporation of Job Entry Subsystem and Remote Entry Services into the Release 2 scheduler. These functions provide efficient job entry from both local and remote users, providing a transparency of operation that enhances remote capabilities. I will also investigate these changes in detail at a later point in the paper.

Finally, VS1 will contain a new data management function—Virtual Storage Access Method (VSAM). This function and its new data set organization has been added, as an ISAM (Index Sequential Access Method) replacement, to better support more sophisticated and online applications. Significant improvements in the exploitation of relocate, data integrity and recovery, device independent addressing, and migration ability help to make VSAM an important base for data base development. Since VSAM is a topic in itself, it will not be discussed in this paper.

RELOCATE

General discussion

Virtual storage separates address space and real storage and then expands the address space to make it larger than real storage. In VS1, address space can be up to 16,777,216 bytes containing the control program, data, and normal application jobs within partitions. Virtual storage addresses are not related to real storage addresses, but both are broken into 2048-byte sections called in virtual storage, pages, and in real storage, page frames. A great deal of study went into determining the optimal page size for a VS1 environment. Involved in this study was a determination of the effective CPU time for instructions and data within a page and the time taken to move the page to secondary storage from real storage. The page size of 2K balances these considerations for optimal performance.

In like manner, the DASD (Direct Access Storage Device) mapping algorithm was considered critical in achieving both medium entry and performance in that entry level. The direct mapping of virtual to secondary space greatly simplifies the movement of data from real to secondary storage and reduces the logic size of the page input/output routines.

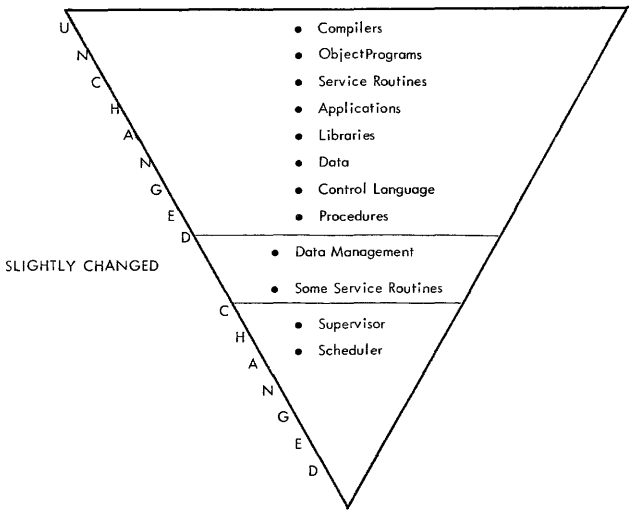


Figure 1—Areas of change between OS/MFT and OS/VS1

Page management

The key component in the management of virtual storage is page measurement. Page measurement is accessed directly by the System/370 hardware when a page exception occurs. A page exception occurs when the address translation feature is unable to resolve a virtual address to a real storage location. At the point of the exception, page management assumes responsibility for ensuring the addressability of the initial storage contents.

OS/VS1 uses a number of pointer queues to manage its least recently used page replacement algorithm and regulate the flow of pages to and from the external page storage. Some of these queues include:

1. In-Use Queues—The addresses in these queues point to locations of currently active page frames. These frames contain the most recently executed code and the most recently used data and tables. The number of in-use queues is a variable dependent upon the number of active partitions and active tasks including system tasks. Figure 2 shows four such in-use queues.
2. Available Page Queues—This queue contains the frames that are available for program servicing when a page fault occurs. At the initial program load, all RSPTE's (real storage page table entries) representing real storage blocks above the fixed nucleus appear on this queue. As execution occurs, this queue is maintained at a minimum threshold to minimize both lockout and thrashing possibilities.
3. Page Input/Output Device Queues—These queues are addresses of frames that are being used for page I/O. The input queue represents the list of frame addresses that are currently being filled from external page storage (SYS1.PAGE). The output queue contains the addresses of the least referenced pages that are about to be stored on external page storage (SYS1.PAGE).
4. Logical Fix Queue—This queue contains the addresses of both short-term fixed page frames and long-term page frames.

Keys to page frame arrangement are the change and reference bits. Both bits are set by hardware and reset in the process of paging activity by the page management routines. The change bit indicates whether the contents of a given page frame have been modified since the page was brought into real storage. This bit is reset only when the page is moved to the external page file. The reference bit is turned on when reference is made to the contents of a page frame.

At periodic intervals (between 3 and 9 task switches in Release 1), the status of the in-use queues page frames is adjusted. This process involves the migration of all unreferenced frames to the next lower queue and all referenced frames to the highest level queue. This migration enables the low reference level frames to move to the lowest level queue and eventually permit their replacement.

As we have noted before, when a referenced page is not contained in real storage, the hardware facility turns control over to page management. Page management immediately looks to the available queue to satisfy the request. If an adequate number of frames is available, the request is immediately satisfied. If there is an inadequate number to satisfy the request, the page replacement routine is entered. The page-frame release request formula is applied as follows:

$$A + HTV - APC = \text{Release Request Amount}$$

where:
A = Page Allocation Request
HTV = High Threshold on Available Page Queue
APC = Available Page Frame Count

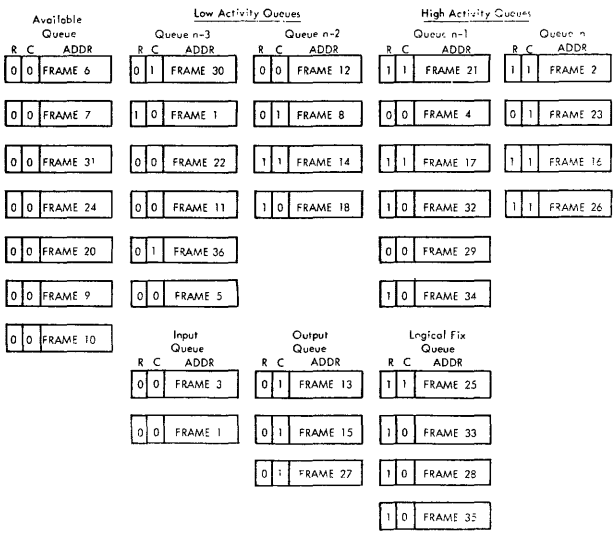


Figure 2—Four in-use queues

This calculation will indicate how many additional page frames should be released to maintain the available queue at an acceptable level. The page replacement routine will begin a scan of the low usage queues to determine what frames may be freed. Page frames that have the reference bit turned off can be released to the available queue. If a change bit is turned on, the frame must first be moved to the output queue where it is placed on the external page storage. Frames that have not been changed are moved directly to the available queue.

Let us again refer to Figure 2. An entry to the page measurement routine would move all frame addresses to the next lower level queue where N-3 is the lowest possible level. In like manner, all frames that have been referenced are moved to reference level N in the queue structure. (This includes frames 1, 14, 18, 21, 17, 32, 34, 2, 16, 26.) The reference bits are reset on all frame indicators on the N-queue. The change bit is not modified at this time nor is the reference bit pattern on the logical fix queue.

Similarly, the process of page release concentrates on the low activity queues moving in a right hand scan from the lowest to the highest queue. Once again referencing Figure 2, frames 30, 22, 11, 36, 5 are currently available on queue N-3 and are thus available for release if required. If we establish the low threshold as 3 and the high threshold at 6 for the available queue, any request for five or more pages would force the release routines to be entered. The frames on the output queue will be moved directly to the available queue when the interrupt is returned.

The page release routines will scan only the low activity queues. If an inadequate number of frames can be obtained from the low level queues, then partition deactivation is entered. Deactivation will deactivate a partition at a time to make available their frames for additional page requests. Partitions are deactivated from low order of priority to high order of priority.

Deactivation controls excessive paging rates known as thrashing. Thrashing, a classical problem in paging systems, is caused by a hyper-contention for available real storage. Deactivation reduces the contention by reducing the number of active tasks. Consequently, the severe contention is eliminated and performance is maintained at an adequate level.

The opposite performance problem is an insufficient number of active partitions. Reactivation must be entered in adequate time to permit a properly balanced range of CPU time. Periodic checks are made to determine the availability of resources for reactivation. Deactivated partitions are reactivated in order of highest priority when a task switch occurs. Release 2 of VS1 has expanded the facility for a user installation to monitor and control some of the deactivation parameters.

Excluded from the deactivation category are:

1. System functions that are necessary for continual execution.
2. Certain system tasks.
3. Jobs that are executing in virtual = real mode.
4. The last active user job.

In summary, the page management routines play a vital role in maintaining system performance at an acceptable level. Key in the achievement of these goals is the allotment of an adequate level of resources to the paging routines.

Input/output supervisor

Channels on the System/370 do not perform address translation on channel control word addresses. Since these are virtual addresses, they must be converted to real addresses for proper program execution; the input/output supervisor of VS1 must do the additional translation. In addition, certain information must be fixed in real storage to avoid a page exception in the middle of an I/O operation.

In the normal execution of an I/O request, therefore, the I/O Supervisor must first fix the frames that contain tables, buffers, and work areas. Once this information is fixed in storage, the real addresses will be placed in the appropriate locations in the channel control word. The Start I/O is issued to a chain that contains real addresses, and is thus referred to as a real channel program. Upon the return of the I/O interrupt, the frames are freed and returned to the normal processing queue. Programs that create self-modifying channel control word chains cannot be handled by the normal I/O Supervisor routines and should be run in a virtual = real mode.

OS/VS1 provides a function known as virtual = real mode. The address space assigned to the job step is placed in a contiguous real location below the V = R line. The size of the V = R area is specified on the REGION parameter and represents the actual size of the program to be executed. Since the V = R area must be contiguous, the job step execution must wait for a free contiguous space to be freed. In addition, the V = R job step is not deactivated during the entire job step execution.

Although the DAT feature is in use during the execution of the V = R job, no translation takes place. In like manner, software translation of the channel control word is avoided.

The virtual = real address space permits the execution of highly time dependent or self-modifying programs. In addition, certain high I/O activity job steps may be run in V = R mode to avoid the CCW translation. It is apparent, however, that the additional overcommitment of real storage may adversely affect other areas of the system.

Storage management

The advent of relocate served to modify the storage management algorithms of the operating system. Portions of the control program that were either optional or resi-

dent in the nucleus could be repackaged into pageable system modules, thus reducing the contention on critical real storage.

Similarly, the System Queue Space which became a critical resource in OS/MFT was broken into three positions dependent upon the area of required information.

1. The System Queue Area (SQA) was designated as a permanently fixed area in real storage that could be dynamically extended or contracted dependent upon its usage. The SQA is used for channel control word translation and for relevant system-oriented tables.
2. The Fixed Portion Queue Area (FPQA) is a permanently fixed area used primarily for partition page tables.
3. The Pageable Partition Queue Area (PPQA) is a protected portion of each partition that contains partition-relevant tables.

The fixed nucleus in VS1 contains the normal control program functions and is permanently in real storage. Strict control should be exercised over the nucleus in small systems to provide an adequate amount of real storage for the paging process.

User partitions must be defined in VS1 in 64K increments of virtual storage. The user may define up to fifteen user partitions and fifty-one system task partitions. Normally the supporting system modules such as data management would be located in the users partition. A user may define resident access methods in the pageable resident access method area for space and performance considerations.

We have discussed the major areas of change made necessary by the relocate function. Dynamic dispatching affected a change in the dispatching techniques of the system in Release 2. Additional changes were made to portions of Data Management and the Scheduler to reflect the hardware and supervisor changes. The Scheduler became the first user of relocate and, as such, repackaged certain portions to reduce branches and move subroutines in line. We will next investigate the major areas of change in Job Scheduling.

JOB SCHEDULING

A number of fundamental design decisions changed much of the VS1 Scheduler. These decisions ranged from the simple packaging choice for modules to the complex inclusion of spooling algorithms within the scheduler framework. Many of the changes were intended to improve user accessibility to the system, while others removed potential bottlenecks to improve performance. Some, such as I/O Load Balancing, are intended to perform total performance improvement as I/O utilization is spread intelligently across channels. The end result was a faster, cleaner component that provides a growth step to the seventies.

Basically the support scope of the relocate function could have limited the scheduler changes to those control card modifications and some internal changes in the Program Status Word and Set System Mask areas. It was recognized, however, that additional benefit could be derived from tailoring the scheduler to make use of relocate. The original base scheduler in MFT used two basic options to schedule jobs. The first was intended for any program with a partition in excess of 44K bytes. The second option was intended for small partition scheduling executed in a linear fashion in the 2K transient area. Investigation demonstrated that a performance and maintainability gain would result by changing the Scheduler to always execute in a 64K virtual partition. The need for a small partition scheduling algorithm is eliminated since the Scheduler can execute in a minimum number of real page frames.

In like manner, portions of the Scheduler such as termination routines were in part repackaged and in part recoded to better support relocate. This was done by moving high incident subroutines in line to avoid excessive paging activity. In addition, a regrouping of tables based upon reference rate and location of reference reduced the paging activity.

These changes did not affect the basic execution order for the Scheduler; however, other enhancements modified the functional structure of the component while maintaining the outward interface.

We will now look in detail at some of these major enhancements to the VS/1 Scheduler.

Central queue manager

An early analysis of the Job Queue usage indicated a need for a redefinition of the contents and structure. The MFT Job Queue Data Set (SYS1.SYSJOBQE) contained various forms of job control information including the actual job queue. Access to this queue was spread through a number of in-line routines to the 176-byte chained records. VS1 has broken the job queue into a number of specialized data sets to reduce the bottlenecks of the old structure. These data sets include:

1. Job Queue (SYS1.SYSJOBQE) retains the name of the MFT data set but it is only a fraction of the size. Relevant information to job queuing is stored on this data set. Disk entry records and accounting records are placed on the data set according to class and priority. When jobs terminate, an entry is made for SYSOUT information according to class. The job queue information is deleted following the processing of the last SYSOUT record.
2. Scheduler Work Area Data Set (SWADS)—this data set is created when an Initiator is started on a partition basis. A SWADS contains the Scheduler work tables that are created and maintained throughout the scheduling routines.

3. Spool Data Set (SYS1.SYSPOOL)—this data set contains the Job Control Information, commands and input from the reader. On the output side, the data set contains output and messages related to each job execution.

We will discuss in the following part of this paper the relevance of the spooling data set. It is apparent that the dichotomy of the queue information into a number of parts has reduced contention problems.

Job entry subsystem (JES)

One of the broadest functional changes to the VS1 Scheduler was the incorporation of the Job Entry Subsystem. JES incorporates a high-speed spooling mechanism into a pageable centralized routine for Scheduler usage. JES is so structured that apart from a minimum resident response routine, it is pageable in all or in part depending upon the frequency of usage.

The first external introduction to JES is through the input reader. The Job Entry Peripheral Services (reader and writer) handles all the system input (SYSIN) and output (SYSOUT). The JES reader is designed to read and immediately pass the input data to the Spool Manager. This changes the sequence of interpretation so that Job Control Language (JCL) interpreter now runs as a subroutine of the Initiator. This delayed interpretation can be prevented by entering a new parameter: TYPRUN = SCAN on the job card. In this case, a simple error scan is performed and the job is flushed through the reader. Once the errors are corrected, the job must be resubmitted.

The input from the JES reader is submitted to the Job Entry Central Service routines. An internal job name has been assigned at this time which is a combination of the user job name plus a unique system number. The central service routines will separate the input from the JCL and write both to the SYS1.SYSPOOL data set. In like manner, in-line procedures and entries from the procedure library (SYS1.PROCLIB) are placed in a special procedure SPOOL area. The division into separate areas enables the JES routines to minimize disk-access contention and thus improve performance. JES maintains an information directory to allow rapid retrieval from all areas of the spool file.

The JES reader for card devices does not terminate at the end of file as in MFT. This facility has become known as a "hot" reader facility. Another advantage of the JES readers and writers is the single reentrant copy maintained in the pageable system area assuring user access to all partitions.

In order to provide greater flexibility of use, the JES parameters are stored in the parameter library (SYS1.PARMLIB) and may be modified during the Initial Program Load (IPL) process. This facility is useful in modifying the number and size of the JES buffers and greatly reduces the need for a new system generation.

Remote entry services

An important adjunct to the Job Entry environment is Remote Entry Services (RES). RES is a logical terminal extension of Job Entry using the Remote Terminal Access Method (RTAM) to drive the terminal devices. Just as JES is intended as an incorporation of HASP techniques into the center of the system, RES is based on HASP Remote Job Entry routines and in fact uses modified HASP work-station support.

RES supports Point-to-Point (leased and dial-up) and the Two-Wire and Four-Wire Half Duplex lines. As with HASP, RES supports multi-leaving which is synchronized, two-directional transmission between two processing units. RES, like most of JES, uses relocate facilities and is therefore pageable.

The RES design is totally integrated into the JES structure so that RES is treated as a logical extension of the JES reader and writer and thus communicates to the system in the same manner.

MIGRATION

One of the first and fundamental questions that arises about the virtual systems is the ease, or lack of ease, of moving current programs to the new system. As I indicated in the introduction to the paper, many of the existing capabilities of current Operating System programs were moved to the relocate counterpart totally or largely intact. We should look at some specific areas to get a better comprehension of the differences.

The first introduction to a new operating system is the vast body of reference information intended to introduce the different competence levels to the software. In the case of VS1, this begins with the IBM System/370 System Summary (GA22-7001) which is intended as a general introduction to the system providing an overview of new and expanded capabilities. In like manner, the OS/VS1 Planning and Use Guide (GC24-5090), A Guide to the IBM System/370 Model 145 (GC20-1734) and the OS/VS1 Features Supplement (GC20-1752) are intended to introduce the reader to a more specialized and in-depth treatment of relevant system components. The general mode of the documentation has been very favorably received since it embodies a strong technical content with an easy to use style. The documentation has been based on the existing operating system documentation but style changes have had a favorable effect on the readers.

The documentation is presented in specialized packages that attempt to match relevance of information with the specialized need to know. Thus an operator does not have to dig through as much generalized information to find the fundamentals of operation.

The system generation process (SYSGEN) has been simplified by the reduction of options. As I indicated earlier, many of the former optional control program func-

tions have been incorporated into the virtual control program portion of the pageable space and thus removed from the option process. In like manner, the use of JESPARM feature represents a considerable advantage when the specific JES parameters should be modified.

The operator interface to the operating system is largely identical to MFT. Some additional work is done during the Master Scheduler Initialization routines causing some additional messages and perhaps responses on the part of the operator. In like manner, dumps of virtual memory will initially cause the operator to speculate as to what is happening, but by and large the operators to the VS1 system have not found the system more difficult to operate or for that matter really any different from their current procedures.

The migration of the programmers to virtual memory is largely straightforward. The programmers are faced with a removal of concern for memory management and overlay in their application substructure. Once again the virtual dump is usually larger but comparable to a large degree to the MFT dumps. Initially, experience has shown that some time is spent in locating some of the familiar tables that have been separated in the protected sections of the system tables. But in the long run, the job seems simplified by the grouping of partition-relevant information in a single place—the partition.

The adjustment to VS1 has been very rapid. Many users have compared the ease of migration to VS1 with the ease of moving to a new release of MFT. In like manner there has been genuine surprise at the increase in functional ability and the increased volume of job throughput.

Verification of a virtual storage architecture on a microprogrammed computer

by W. A. SCHWOMEYER

International Business Machines Corporation
Endicott, New York

INTRODUCTION

The verification of a virtual storage architecture shares many commonalities with the verification of any computer architecture. This paper will therefore define an approach to architecture verification in terms which are independent of virtual storage. It will then discuss the application of this approach to a virtual storage architecture. The paper will conclude with a discussion of a method for determining the completeness of an architecture verification process on a microprogrammed computer.

ARCHITECTURE

The functional description of a computer as observed by an external user or programmer is usually contained in an architecture for that computer. It is the purpose of an architecture to provide a functional description based upon the requirements of the external user or programmer, independent of any particular implementation. In fact, an architecture may serve as the functional description for several models of a computer family with each model implementing the same functions in different technologies or media. The architecture is the base for determining functional compatibility between models.

The architecture will include a definition of the priorities for handling all simultaneously occurring functions. Where the architecture justifiably defines the priority to be "unpredictable," the exact priority definition is left to each implementation.

An architecture defines two types of entities. The *computer state* entities define the state of the computer as it can be interrogated by the external user or programmer. Some examples of computer state entities are: main storage contents, programmable register contents, comparison and arithmetic computation result indicators, interrupt mask indicators, and supervisor or problem state indicators.

The *computer function* entities define processes which, when performed, result in a change, or a significant lack of change, in the computer state entities. The most familiar computer function entities are the excution of machine instructions. These instructions are usually grouped into one or more classifications based upon the type of operands on which they operate. Some examples of these classifications are: storage instructions, register instructions, decimal arithmetic instructions, floating point arithmetic instructions, emulation instructions, and timing facility instructions. Often these classifications are analogous to certain optional or standard computer features.

Other examples of computer function entities are: interrupt processing, manual operations, and timing facilities.

Architecture verification in the context of this paper is the process which verifies whether a particular computer implementation conforms to the functional definition in its architecture. In terms of the previously defined architecture entities, this means: execute the *computer function* entities and observe and verify the resulting change (or lack thereof) in the *computer state* entities. It is generally expected that the architecture has already been verified as adequately meeting the external user's requirements.

TEST PHILOSOPHY

To understand the principles discussed later in this paper it is necessary to be familiar with the circumstances under which an architecture verification test is performed. Architecture verification is only one of the many tests performed during the course of designing a computer, building a prototype, and manufacturing a finished product.

In addition, these tests, including the architecture verification, may be performed repeatedly at different times within a development cycle.

Architecture verification tests are usually conducted in two environments. In the first environment, each computer function entity is tested under static or quiescent conditions with no interaction from other function entities. Ironically, testing in this environment has been the most difficult to control.

Historically, it has been characterized as being greatly dependent on manual operations. Tests were entered into

the computer manually and the results recorded manually. The test was a slow and tedious process. Heavy reliance on human intervention created many exposures in the effectiveness of the test. Oversights and discrepancies in recording test results, misunderstandings in communicating exact manual procedures from one group to another, and oversights in performing required tests at all necessary stages of development are results of human error in performing the manual procedures.

Improvements in the control of testing in the static environment will be evident in the discussion under TEST OBJECTIVES and TEST APPROACH.

The second environment involves testing the computer function entities under dynamic conditions. Differing amounts of random interaction are introduced to verify that there are no problems when many function entities contend for CPU (central processing unit) resources. Test programs are written which strictly control and verify the execution of the various computer function entities.

These test programs execute under a supervisor program which allows them freedom in controlling all computer state entities. The supervisor program controls the initiation and termination of execution of the test programs. Interaction between the function entities tested by concurrently executing test programs occurs at random. The existence of the test and supervisor programs ensures the capability to repeat the test as needed and decreases the amount of manual procedures which must be communicated between various groups performing the test.

Deficiencies in the test result from the fact that it is still a manual procedure to verify the thoroughness of the test and the test programs usually employ only valid architecture entities. The latter deficiency results from the assumption that the computer correctly detects architecture violations. Enhancements to testing in the dynamic environment will be discussed under VIRTUAL STORAGE.

TEST OBJECTIVES

Based upon the previous discussion, the following desirable traits or objectives can be defined for a basic test approach to be used to verify an architecture in the static test environment.

- *Controlled Environment*—The test must be performed on a static or quiescent computer with no interaction between functions.
- *Ease of Use*—The test must be easy to perform (usually implying a minimum of manual intervention) and the results must be easy to interpret. It will be recalled that the test will be performed at different times in the development of a computer. As may be expected, there are differing users who will perform the test and have differing information requirements from the results of the test.

  In addition, the test will be performed on a variety of computers or models with different features installed. It must be easy to perform the test on each of these computers with little concern on the part of the user for the features installed.
- *Repeatable*—Because of the number of times the test will be performed, it must be repeatable with the expectation that the results of the previous tests will be duplicated each time it is performed.

  Often a simulator is used in the early stages of computer development. It is advantageous to develop an architecture verification process which not only can verify the architectural implementation of the simulator but also can verify the architectural implementation of the resulting computer. This will ensure that the computer accurately duplicates the design in the simulator.
- *Thorough*—It is the intent of the test to verify 100 percent of the architectural implementation. However, in practice some compromise is usually made between the thoroughness of the test and the time and cost required to achieve 100 percent verification.
- *Transparent*—The test must be performed in a way which does not alter the architectural implementation being verified.
- *Fast*—Due to the number of times the test is performed, it is desirable to perform the test as fast as possible. An increase in speed can be obtained by decreasing the amount of manual intervention required.
- *Model Independent*—The test should be model independent to allow the same test to be used in verifying multiple implementations of the same architecture. This will also verify the compatibility between the computer models.
- *Extendable*—While the test should be model independent, it should also be extendable to allow the testing of model dependent results (architecturally "unpredictable") when required. This capability will also allow extension of the test if additional architectural verification requirements are determined.
- *Portable*—The test tool must be easily transportable from one computer or model to another, whether they are located at the same computer installation or at locations separated by large geographic distances.

In addition to the above objectives, there is one additional objective not directly related to the effectiveness or usability of the test vehicle itself.

- *Ease of Implementation*—The test tool should be easy to implement, providing as much flexibility in the implementation process as possible. As indicated above, the test should be easily extendable to provide additional architectural verification as the requirements are determined.

TEST APPROACH

The test approach for verifying an architecture in the static test environment is divided into the following parts: control program, test cases, and support programs.

Control programs

The control program is a "stand-alone" software program implemented in the architecture to be tested. It *controls* the initialization of the computer state entities, the execution of the computer function entities, the interrogation and verification of the resultant computer state entities, and the indication, to the user, of the test results. It is not designed to explicitly verify any portion of the architecture implementation.

The number of failures which may be experienced during execution of the control program is reduced by minimizing the number of different computer functions and computer state entities utilized in implementing the control program.

Selection of the computer entities to be used is based on their simplicity. The simplest entities are usually those whose principles are common to the definition of most computer architectures and are usually included as standard features on the computer. Due to their simplicity and familiarity, these computer entities are less susceptible to design errors and any failures which do occur are more easily defined.

The control program systematically interrogates a computer state entity for each optional computer feature. The results of the interrogation indicate to the control program which optional computer features are installed. This information is used by the control program to determine which tests may be meaningfully executed. The user is also provided the capability to manually select the tests to be executed, overriding the selection of tests made automatically by the control program.

The control program indicates to the user the success or failure of the computer in conforming to its architecture. For most users it is sufficient to provide this indication in a brief message for each installed computer feature. The individuals responsible for defining and correcting a problem will have need to manually select the messages providing more details concerning the problem.

The time for execution of the test is reduced by minimizing the amount of manual intervention required for normal execution and by employing efficient, straightforward coding techniques in designing and implementing the control program. Equally important is the means by which the control program obtains the description of the test to be performed. The information required to describe the test is defined in the next section. The format in which this information is presented to the control program should be carefully selected to advantageously use the properties of the computer entities utilized by the control program in handling the information.

Test cases

A test case is a data record providing information to the control program completely *defining* a test to be executed by the control program. Each test verifies the action of one computer function entity upon a single combination of all the computer state entities. Consequently, many test cases are required to test the action of all computer function entities upon all combinations of all the computer state entities.

Each test case contains the following information:

1. Computer function entity
2. Computer state entities
   a. initialization values
   b. resultant values
3. Test case execution control information

The computer function entity is usually represented by a single computer instruction with a prescribed beginning and termination. Computer function entities, which are not executions of computer instructions, are represented in the test case by a sequence of one or more computer instructions which initiates execution of the computer function entity.

As indicated earlier, the computer state entities define a particular state of the computer, including contents of main storage and programmable registers. The test case defines the *initialization* values of the computer state entities for use by the control program in setting the computer to that particular state prior to execution of the computer function entity. The *resultant* values of the computer state entities are also defined in the test case for use by the control program in verifying that the execution of the computer function entity results in the expected changes in the computer state entities.

The test case execution control information defined in the test case provides a communication link from the individual writing the test case to the control program. When determining which tests are to be executed, the control program must be able to determine if a test case is dependent upon the presence or absence of a particular computer feature. The control program must also be able to determine if a test case is dependent upon a particular computer state entity having a specific value. Changes in the architecture and detection of errors in the test cases sometimes result in changes to the test cases.

The control program must have the capability to indicate, to the user, the change level of the test case being executed. In addition, the control program must have the capability of uniquely identifying each test case from all other test cases. All of the above situations require that specific information be communicated from the individual writing the test case to the control program.

Following are some principles which, if applied during the writing of test cases, tend to improve the overall effectiveness of the test:

- In general, each test case should be independent of the other test cases. This is achieved by initializing all pertinent computer state entities in each test case where they are used. An exception to this principle is the situation where several test cases require large amounts of main storage to be initialized to the same values and execution of the test cases is not expected to change the contents of this main storage.
- The test of the architecture should be as complete as possible. One method of enhancing the completeness of the test is to subdivide the entire architecture into manageable pieces for the purpose of test case writing. Usually the architecture can be satisfactorily subdivided for this purpose into the classifications discussed in the section entitled ARCHITECTURE.

Once the architecture has been subdivided, the completeness of the test can be further enhanced by systematically writing test cases which terminate at each architecturally defined termination point. Execution of these test cases will ensure that the computer correctly detects all architecture violations and all functional entity error conditions.

- The termination of a test case should not depend on the occurrence of any external event. This dependence can be avoided by ensuring that no continuous, never-ending program loops result when using computer instructions to test the computer functional entities.
- It is clear that no computer states which are architecturally "unpredictable" should be included in the architecture verification test which is performed on all the computer models. However, it may be beneficial to test these model dependencies and other implementation peculiarities as an extension to the architecture verification test.
- Usually the number of computer state entities makes it prohibitive to attempt to verify the value of all computer state entities for all possible executions of all computer function entities. Clearly, all computer state entities which are architecturally defined to change or remain constant must be verified. In addition, it may be equally important to verify that the values of certain other computer state entities remain unchanged.

It is clear from the above discussion that the number of test cases included in the architecture verification test will be quite large. It follows that some procedures should be established to control test case development, maintenance, and distribution to the many users.

- The independence of the test cases from each other and from the control program allow several individuals to simultaneously develop the control program and test cases for individual classifications of computer function entities (once the test case format has been defined). Procedures must be established to ensure that test cases are implemented for all classifications of computer function entities.
- Due to the large number of test cases, it is convenient to organize them into a library system. Procedures must be established for adding test cases, deleting test cases, and changing test cases already in the library.
- Many times architecture implementation problems are detected by other programs or other tests. Procedures must be implemented which ensure that a test case is written to adequately test the problem. The test cases resulting from these procedures will improve the completeness of the architecture verification test.
- Changes in the architecture and detection of errors in the test cases sometimes result in changes to the test cases. Procedures must be established which ensure that the library contains the latest level test cases for the latest level architecture.
- It was indicated earlier that the architecture verification test is performed many times by many different users. Procedures should therefore be established for distributing the test to those individuals having a need for it. Magnetic tape and disk are usually satisfactory media for transporting and executing the programs.

Support programs

As previously indicated, the format of the test case, when presented to the control program, should be carefully selected to allow simple and efficient use of it by the control program. This usually means that the test case format is closely associated with the architecture and the internal code of the computer.

Most often this format is not the easiest and most efficient with which an individual can work. Usually a "higher level" format is defined for use by the individuals writing the test cases. A *translator program* is required to translate the test case definition from the "higher level" format to the format accepted by the control program.

The remaining support programs are *library maintenance programs* These programs may take many forms in automating the library maintenance procedures discussed in the previous section.

All of the above support programs are written to execute on a computer whose architecture is not being tested. This procedure has the following advantages:

- Test cases can be developed in parallel with the computer they are to test.
- These programs can use existing computer facilities, such as interactive terminals and data management, to make the jobs of test case development and maintenance simpler and more efficient.

TESTING VIRTUAL STORAGE

Upon investigation of the facilities provided in a virtual storage environment, it would be apparent that most of them are implemented in software. Facilities such as paging, page fixing, resource allocation, and task selection are some examples. Since these functions are implemented in software, they will not be included in the architecture definitions of computer function entities and computer state entities. Therefore, the verification of these facilities is not included in the architecture verification process discussed by this paper.

The virtual storage environment does, however, contain several architecturally defined computer function entities and computer state entities. Some examples of these computer entities are: address translation, page fault detection, associative arrays, page boundary crossings, page referenced indicators, page changed indicators, and computer instructions to control and interrogate the virtual storage environment. All of the architecturally defined computer entities must be tested by the architecture verification process.

This testing is achieved by writing additional test cases for execution with the static environment approach discussed in the section entitled TEST APPROACH.

Test cases must be written to verify correct execution of the virtual storage computer function entities. The test cases previously generated for the basic architecture must be rewritten to include verification of the change in the virtual storage computer state entities and verification that the execution of the address translation entity is transparent in the execution of the basic computer function entities. In addition, test cases must be written to test any "special case" computer function entities which execute differently in the virtual storage environment than they do in the non-virtual storage environment. The virtual storage environment may also give rise to some model dependencies, such as virtual equals real translation, for which it is desirable to write additional test cases.

The section entitled TEST PHILOSOPHY discussed testing in a dynamic environment utilizing many test programs executing under control of a supervisor program. The same section discussed some deficiencies in the thoroughness of the test conducted in the dynamic environment. Most of these deficiencies can be eliminated by writing an additional test program which executes under control of the same supervisor program and utilizes the virtual storage computer entities.

Figure 1 depicts the new, simulator-test program in the dynamic environment. It is referred to as a "simulator-test program" because it actually simulates the architecturally defined computer entities associated with a supervisor or privileged mode of operation. Only those supervisor or privileged mode entities used by the static environment control program are simulated. In addition, this simulator-test program controls the virtual storage

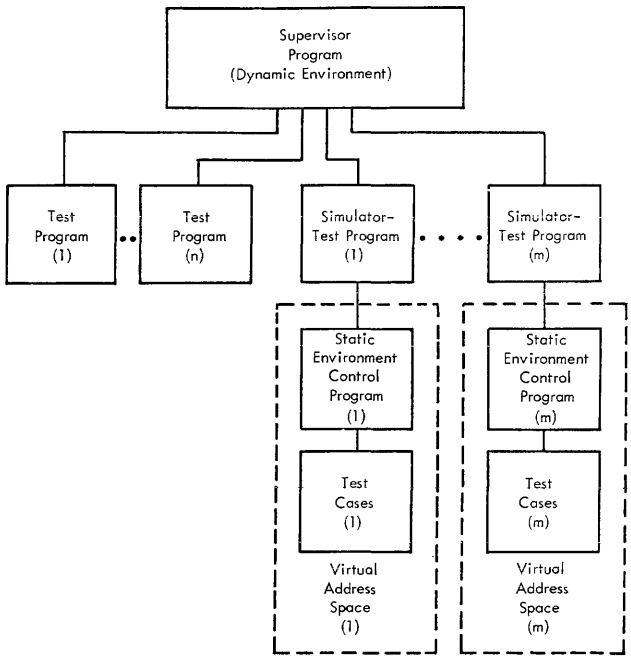


Figure 1—Dynamic test environment incorporating virtual storage capabilities

address translation entity in such a way that the absolute addresses of the architecture verification test control program are interpreted as virtual addresses and are translated to real addresses in the main storage controlled by the dynamic test supervisor program.

The combination of the two techniques allows the architecture verification test control program to execute in the dynamic environment controlled by the supervisor program as though it were the only program in the computer. The benefits of this arrangement are:

- The completeness of the architecture verification test in the dynamic environment is enhanced by execution of the basic test cases in this environment.
- The need to rewrite the basic test cases for execution in the virtual storage environment has been eliminated by actually executing the basic test cases for execution in the virtual storage environment.
- The concept of multiple virtual memories is verified by concurrent execution of multiple copies of the simulator-test program.

The following limitations imposed by the changes in the execution environment restrict which of the basic test cases can be correctly and meaningfully executed in the dynamic environment:

- Test cases depending upon specific timing considerations cannot be executed in the dynamic environment because of the interactions with the other test programs.

- Test cases requiring manual intervention may not be executable due to limitations in the manual operation-simulator-test program interface.
- It is not meaningful to execute test cases employing supervisor or privileged mode operations since these operations are simulated by the simulator-test program and do not result in an actual test of the architecture implementation.

TEST COMPLETENESS

One of the test objectives identified in the section entitled TEST OBJECTIVES is thoroughness. Some procedures have been discussed which can help ensure a thorough test. This section discusses an approach for measuring the completeness of an architecture verification test as applied to a microprogrammed computer.

In a microprogrammed computer, most of the external specifications of the architecture are implemented in microcode. As a result, some software debugging techniques can be employed. Specifically, a microcode trace could be obtained, using some hardware or software tool, and the results analyzed to determine which paths through the microcode have been exercised and which have not. However, the number of possible paths through the microcode and the number of test cases which would have to be individually traced make this approach prohibitive.

An alternative is to record an indication of which microinstructions have been executed and which have not. Additionally, an indication of which microinstruction branches have been executed can be recorded. Since only the total number of microinstructions and branches executed is of importance (and not the order in which they were executed), a separate record is not required for each test case. Only a composite record reflecting the execution of the entire set of test cases is required. However, for informational purposes, separate records of the static and dynamic environments may be desirable.

The completeness of the architecture verification test is then expressed as the percentage of the total number of microinstructions which were executed and the percentage of the total number of microinstruction branches which were taken. In analyzing the results of completeness verification, two pitfalls should be avoided:

- The total percentage coverage figures can overshadow the fact that one or more computer function entities may not be tested at all.
- An individual writing test case to test a specific computer function entity may desire to know what the completeness of his test case development activity is at a given point in time. For the percentages of a partial completeness report to be meaningful, it should be ensured that they are based on the number of microinstructions and branches which the test cases are designed to test.

With some knowledge of the implementation of the architecture and the information as to which microinstructions and branches have been covered, the individual writing test cases can determine which test cases are required to fill the holes in the test. Discretion should be used here, for the effort involved in generating test cases usually follows the law of diminishing returns. That is, it generally will take more effort per microinstruction to move from a 90 percent to a 95 percent coverage than it will to move from a 50 percent to a 75 percent coverage.

SUMMARY

This paper has described in general terms the techniques for implementing a basic architecture verification process and applying that process to a virtual storage architecture. This will allow the reader to mold the techniques and apply them to a given, specific architecture. The exposures and principles indicated throughout the paper will aid the reader in implementing a thorough and efficient test.

ACKNOWLEDGMENTS

The author acknowledges Mr. F. W. Ball, Mr. E. M. Shimp, and Mr. D. A. Shiner as the initial creators of the static test approach. Numerous other individuals have used both the static and dynamic approaches. Their discussions with the author have resulted in the approaches defined in the paper.

Especially appreciated were the efforts of my coworkers in patiently and diligently reviewing and critiquing this paper.

On a mathematical model of magnetic bubble logic

by EIJI YODOKAWA

Musashino Electrical Communication Laboratory, NTT
Tokyo, Japan

INTRODUCTION

In 1967, A. H. Bobeck[1] showed that, under suitable magnetic conditions, small discrete cylindrical magnetic domains can be stably supported in thin platelets of certain orthoferrite materials, and these domains can be moved within the material by the application of suitable external magnetic fields.

These cylindrical magnetic domains, commonly referred to as bubbles, have an application in performing memory function, since the presence or absence of a bubble at a particular location can be treated as binary information. Furthermore, these bubbles can be utilized to perform logic functions, since they can be manipulated by the application of external magnetic fields as well as by the magnetostatic repulsion between adjacent bubbles.

Some remarkable features of bubble devices are the realization of both memory and logic functions in the same platelet of bubble materials, the very low power dissipation, the high storage density and the low cost. For these reasons, bubbles have been investigated extensively in recent years. Especially, the design of a bubble computer[4] is very interesting.

This paper is concerned with a mathematical model of bubble logic. A simple model using only the bubble transfer operation was studied earlier by R. L. Graham,[2] who showed that there exist combinational functions of 11 or more variables that cannot be computed by his model. A. D. Friedman and P. R. Menon[3] extended Graham's model by introducing different types of operation which enabled the computation of all combinational functions, and then discussed the problem of efficient computation from the viewpoints of time and space requirements and geometrical requirements imposed by the fact that bubble interactions can occur only between physically adjacent locations.

This paper first briefly reviews Graham's results. After that further properties of his model are investigated from the viewpoints of the realization of combinational functions and geometrical requirements mentioned above. Next, Graham's model is extended by introducing a different type of bubble interaction, the magnetostatic repulsion between adjacent bubbles, which seems to be more practical. The computational capabilities are especially investigated in the present model.

The features of the model are the use of only the two types of operation, i.e., bubble transfer and conditional bubble transfer, and the restriction on space used for the computation, which meet both the engineering and economical requirements.

DESCRIPTION OF THE MODEL

Operations on bubbles

Let S be a finite set of n discrete locations at which bubbles may lie. Let X be any subset of S which is actually occupied by bubbles. It is assumed, without loss of generality, that all locations of S are adjacent to one another so that interaction between any pair of locations is possible. The following two types of operation are introduced on the set S. The first type of operation is the transfer denoted by the form $e=(a, b)$, and the second one is the conditional transfer denoted by the form $e=(ab, c)$. These operations are called commands. The command $e=(a, b)$ or (ab, c) transforms the locations of the bubbles from X to X^e by the following definitions.

transfer:

$$X^{(a,b)}=\begin{cases}(X-\{a\})\cup\{b\} & \text{if } a\in X,\ b\in X,\\ X & \text{otherwise}\end{cases}$$

conditional transfer:

$$X^{(ab,c)}=\begin{cases}(X-\{b\})\cup\{c\} & \text{if } a, b\in X,\ c\in X,\\ X & \text{otherwise}\end{cases}$$

A program P (of length $k\geqq 0$) is defined to be a finite sequence $P=e_1 e_2 \ldots e_k$ of commands. The length of any program P is denoted by $lg\ (P)$. A program of length zero, called the empty program, is denoted by λ. For any bubble location X and any program $P=e_1 e_2 \ldots e_k$, X^P is defined by

$$X^P=(\ldots(X^{e_1})^{e_2}\ldots)^{e_k}.$$

Therefore, in general, a program P maps the set of all 2^n subsets of S into itself.

Two dimensional bubble logic

Let us now consider the problem of computing the Boolean functions of m variables with appropriate programs. Following Graham,[2] two locations, a_1 and $\bar{a}_1$, are used to represent

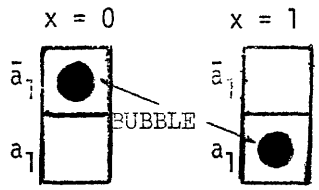


Figure 1—Configurations which represent $x=0$ and $x=1$

the value of a binary variable x, as shown in Figure 1. For m variables, a set of $2m$ locations is used which is imagined to be arranged in pairs, as illustrated in Figure 2.

A restriction is imposed on the locations, which can be used for the computation of the Boolean functions of m variables, namely $S=\{a_1, \bar{a}_1, \ldots, a_m, \bar{a}_m\}$.

Note that the space which can be used for the computation of the Boolean functions is not unbounded, but is defined by the input space.

For m variables, $x_1, x_2, \ldots, x_m$, let $W_1, W_2, \ldots, W_2m$ be all 2^m inputs, and let $X_1, X_2, \ldots, X_2m$ be the corresponding bubble locations. Let $f(x_1, x_2, \ldots, x_m)$ be a Boolean function of m variables. A Boolean function $f(x_1, x_2, \ldots, x_m)$ is said to be realized by a program P if the following condition is satisfied:

There exist some location $\rho \in S$ and some program P such that for $i=1, 2, \ldots, 2^m$,

$$\rho \in X_i^P \text{ if } f(W_i)=1,$$

$$\rho \notin X_i^P \text{ if } f(W_i)=0.$$

Note that the value of a Boolean variable is represented by using two locations, but the value of a Boolean function is represented by the presence or absence of a bubble in a particular location.

Let $f_a(i)$ be the value of the function which is represented in a location a, after application of the ith command e_i of a program $P=e_1e_2 \ldots e_i \ldots e_k$. If $e_{i+1}=(a, b)$, then

$$f_a(i+1)=f_a(i)\cdot f_b(i)$$

$$f_b(i+1)=f_a(i)+f_b(i)$$

Where $\cdot$, $+$ are logical AND and logical OR, respectively. We usually write $f_a(i)f_b(i)$ instead of $f_a(i)\cdot f_b(i)$. If $e_{i+1}=(ab, c)$, then

$$f_a(i+1)=f_a(i)$$

$$f_b(i+1)=f_b(i)f_c(i)+f_b(i)\bar{f}_a(i)$$

$$f_c(i+1)+=f_a(i)f_b(i)+f_c(i)$$

where $\bar{f}_a(i)$ is the complement of $f_a(i)$.

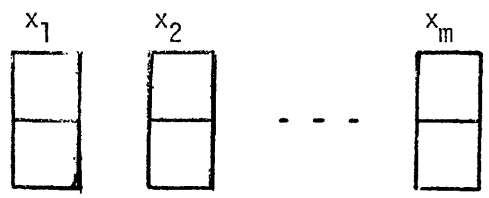


Figure 2—Symbolic arrangement of locations for computing Boolean functions of M variables

In the present model of bubble logic, two types of commands can be used, i.e., transfer and conditional transfer, but in Graham's model, only transfer commands can be used. Thus, the present model is an extension of Graham's model. The main object of this paper is to examine the computational capabilities of the proposed model.

SOME BASIC PROPERTIES OF GRAHAM'S MODEL

In this section, Graham's results are briefly reviewed in order to compare them with the present results. After that, further properties of his model will be investigated from the viewpoints of the realization of the Boolean functions and geometrical requirements imposed by the fact that bubble interactions can occur only between physically adjacent locations.

The following non-decreasing overlap (NDO) theorem accredited to W. Shockley plays a fundamental role in Graham's model.

Theorem 1 (NDO Theorem): Let X_1 and X_2 be two arbitrary initial sets of locations of bubbles and let P be any program using only bubble transfer commands. Then

$$|X_1^P \cap X_2^P| \geqq |X_1 \cap X_2|,$$

where $|X|$ denotes the cardinality of set X.

The first important consequence of the NDO theorem is the following result.

Proposition 2: There is no replicating program P^* using only the bubble transfer commands.

A program P is said to be a replicating program if the following conditions are satisfied:

Let S_1 and S_2 be two fixed subsets of S, such that $S_1 \cap S_2=\phi$, and let θ be a one to one mapping from S_2 onto S_1. Then, for each X, such that $|X| \geqq 2 |X \cap S_1|$,

1. $X^P \cap S_1 = X \cap S_1$

and

2. $\theta(X^P \cap S_2) = X \cap S_1$.

In other words, the replicating program P^* creates a "copy" of $X \cap S_1$, in S_2 without disturbing $X \cap S_1$.

Another consequence of the NDO theorem is the following result.

Proposition 3: There is no binary addition program P^+ using only the bubble transfer commands.

A binary addition program P^+ performs binary addition in the following way.

The definition of a binary addition program in Graham's paper[2] is repeated here.

Assume S_1 denotes a set of $m \geqq 1$ pairs of locations of S, S_2 denotes another set of m pairs of locations disjointed from S_1, and S_3 denotes a set of $m+1$ pairs of locations, disjointed from S_1 and S_2. These sets can be imagined as arranged as shown in Figure 3.

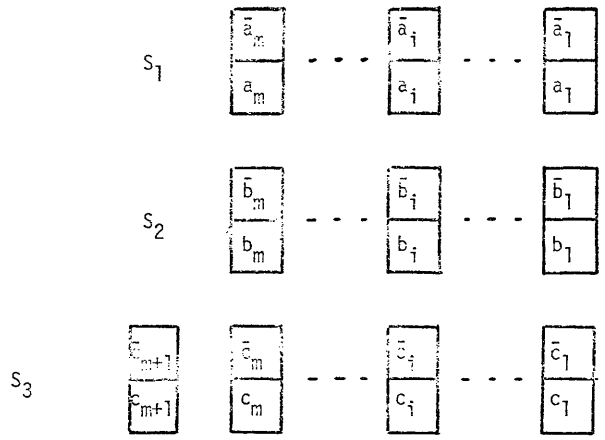


Figure 3—Symbolic arrangement of locations for binary addition

An integer M, $0 \leqq M \leqq 2^m$, can be represented in the m pairs of S_1 by letting the jth pair of S_1 denote the jth binary digit in the binary expansion of M. This can be done using the configurations shown in Figure 1. Thus, for $m=5$, the configuration shown in Figure 4 would denote the integer $10010_{(2)}=18$.

The binary addition program P^+ would operate by starting with S_3 in some fixed configuration and with arbitrary integers A and B loaded into S_1 and S_2, respectively, to form the initial state X; after applying P^+ to X we should get the sum $A+B$ in S_3.

The following theorem is the main result obtained by Graham, and indicates one of the limitations of his model for realizing Boolean functions.

Theorem 4: There exists a Boolean functions of 11 variables which cannot be realized by a program of bubble transfer commands.

At present, it is known that, for $m=1$, 2, 3 and 4, all Boolean functions of m variables can be realized by his model, but it is not known whether all Boolean functions of 5 variables can be realized.

Some properties of programs using only transfer commands are now investigated.

Let $P(f)$ denote a program which realizes a Boolean function f, and let $\rho(P, f)$ denote the location at which the value of f is represented by the presence or absence of a bubble, after applying a program P. Hereafter, the same character shall be used to represent both a variable (or its complement) and its corresponding input location, since these are not confused.

For example, first consider a Boolean function $f(x, y)$ defined by Table I. Thus, in this case, $\{x, \bar{x}, y, \bar{y}\}$ will be taken to be the set of S as shown in Figure 5.

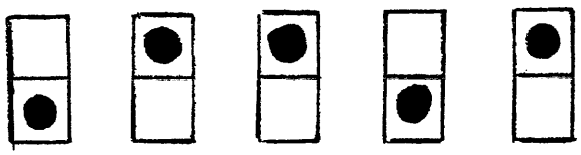


Figure 4—Configuration representing an integer $10010_{(2)}=18$

Table I. Truth table for $f(x, y)$.

x	y	f (x,y)
0	0	0
0	1	1
1	0	1
1	1	0

It is easy in this case to find an appropriate $P(f)$. For example, we can take

$$P(f) = (\bar{y}, \bar{x})\,(y, x)\,(\bar{x}, x)$$
$$\rho(P, f) = \bar{x}$$

For any Boolean function $f(x_1, x_2, \ldots, x_m)$ of m variables, we define $f_{\bar{x}_i}$, $f_{(x_i,x_j)}$, and $\bar{f}$ by

$$f_{\bar{x}_i} = f(x_1, x_2, \ldots, \bar{x}_i, \ldots, x_m)$$

$$f_{(x_i,x_j)} = f(x_1, x_2, \ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots, x_m)$$

$$\bar{f} = 1 - f(x_1, x_2, \ldots, x_m)$$

Proposition 5: If a program $P(f)$ exists, then $P(f_{\bar{x}_i})$ is obtained from $P(f)$ by complementing xi, and letting

$$\rho(P, f\bar{x}_i) = \begin{cases} \bar{x}_i \text{ if } \rho(P, f) = x_i \\ x_i \text{ if } \rho(P, f) = \bar{x}_i \\ \rho(P, f) \text{ otherwise} \end{cases}$$

Proposition 6: If a program $P(f)$ exists, then $P(f_{(x_i,x_j)})$ is obtained from $P(f)$ by permuting x_i and x_j, and letting

$$\rho(P, f_{(x_i,x_j)}) = \begin{cases} x_i \text{ if } \rho(P, f) = x_j \\ x_j \text{ if } \rho(P, f) = x_i \\ \rho(P, f) \text{ otherwise} \end{cases}$$

These results follow directly from the definition of the program.

Proposition 7: If $P(f) = (a_1, a_2)\,(a_3, a_4)\,\ldots\,(a_{2r-1}, a_{2r})$ and $\rho(P, f) = a_{2r-1}$ or a_{2r}, then

$$P(\bar{f}) = (\bar{a}_2, \bar{a}_1)\,(\bar{a}_4, \bar{a}_3)\,\ldots\,(\bar{a}_{2r}, \bar{a}_{2r-1})$$

$$\rho(P, \bar{f}) = \begin{cases} \bar{a}_{2r-1} \text{ if } \rho(P, f) = a_{2r-1} \\ \bar{a}_{2r} \text{ if } \rho(P, f) = a_{2r} \end{cases}$$

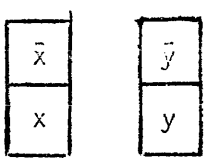


Figure 5—Symbolic arrangement of locations for computing Boolean functions of two variables

Proof: The proposition is proved by induction on r.

By the application of the program $P(f) = (a_1, a_2)$ or $P(\bar{f}) = (\bar{a}_2, \bar{a}_1)$, it follows that $f_{a_1}(1) = a_1 a_2$, $f_{a_2} = a_1 + a_2$ or $g_{\bar{a}_2}(1) = \bar{a}_2 \bar{a}_1$, $g_{\bar{a}_1}(1) = \bar{a}_2 + \bar{a}_1$ respectively. By De-Morgan's theorems, $\bar{f}_{a_1}(1) = g_{\bar{a}_1}(1)$ and $\bar{f}_{a_2}(1) = g_{\bar{a}_2}(1)$. Thus, for $r = 1$, the proposition is true.

Next, assume that the proposition is true for $1, 2, \ldots, r-1$. By the application of the command (a_{2r-1}, a_{2r}) or $(\bar{a}_{2r}, \bar{a}_{2r-1})$, it follows that

$$f_{a_{2r-1}}(r) = f_{a_{2r-1}}(r-1) f_{a_{2r}}(r-1),$$

$$f_{a_{2r}}(r) = f_{a_{2r-1}}(r-1) + f_{a_{2r}}(r-1)$$

or

$$g_{\bar{a}_{2r}}(r) = g_{\bar{a}_{2r}}(r-1) g_{\bar{a}_{2r-1}}(r-1),$$

$$g_{\bar{a}_{2r-1}}(r) = g_{\bar{a}_{2r}}(r-1) + g_{\bar{a}_{2r-1}}(r-1).$$

By the induction hypothesis,

$$g_{\bar{a}_{2r-1}}(r-1) = \bar{f}_{a_{2r-1}}(r-1),$$

$$g_{\bar{a}_{2r}}(r-1) = \bar{f}_{a_{2r}}(r-1).$$

Therefore,

$$g_{\bar{a}_{2r}}(r) = \bar{f}_{a_{2r}}(r-1) \bar{f}_{a_{2r-1}}(r-1) = \bar{f}_{a_{2r}}(r),$$

$$g_{\bar{a}_{2r-1}}(r) = \bar{f}_{a_{2r}}(r-1) + \bar{f}_{a_{2r-1}}(r-1) = \bar{f}_{a_{2r-1}}(r).$$

This completes the proof.

Let $f(x_1, x_2, \ldots, x_m)$ be a Boolean function. Taking account of the operations $\cdot$ and $+$, we write

$$f(x_1, x_2, \ldots, x_m; \cdot, +) \text{ instead of } f(x_1, x_2, \ldots, x_m).$$

By the duality in the effect of a command, we easily obtain the following result.

Proposition 8: If $P(f(x_1, x_2, \ldots, x_m; \cdot, +)) = (a_1, a_2) \ldots (a_{2r-1}, a_{2r})$ and $\rho(P, f(x_1, x_2, \ldots, x_m; \cdot, +)) = a_{2r-1}$ or a_{2r}, then a program which realizes the Boolean function obtained from $f(x_1, x_2, \ldots x_m; \cdot, +)$ by permuting $\cdot$ and $+$, is given as follows.

$$P(f(x_1, x_2, \ldots, x_m; +, \cdot)) = (a_2, a_1)(a_4, a_3) \ldots (a_{2r}, a_{2r-1})$$

$$(\rho P, f(x_1, x_2, \ldots, x_m; +, \cdot)) = \rho(P, f(x_1, x_2, \ldots, x_m; \cdot, +))$$

In the preceding section, it was assumed that interactions are possible between any pair of bubble locations. Since interactions can occur only between bubbles in physically adjacent locations, this implies that bubbles which are required to interact must be brought into adjacent locations without affecting the bubbles in other locations prior to the application of the command. Therefore, the time taken for computing a function depends not only on the number of commands in the program but also on the layout of the bubble locations.

There are several open problems associated with the design of efficient programs for computing Boolean functions in Graham's model.

(1) For a given Boolean function f, find a simple algorithm for determining whether a program $P(f)$ exists.

(2) If a program $P(f)$ exists, find an algorithm for constructing it effectively.

(3) If a program $P(f)$ exists, find a program with the smallest length for computing f.

(4) For a given program, find the assignment of locations so as to minimize the total number of locations required for its implementation.

The programs realizing all Boolean functions of 3 variables and the corresponding input memory layouts with the minimum total locations are now presented.

Note that by Propositions 5, 6 and 7, it suffices to give the programs for one representative for each of 14 equivalence classes of functions of 3 variables under the operations of complementing and permuting the variables, and of complementing the functions. Table II consists of these representatives. Figure 6 shows the corresponding programs and the input memory layouts. Also note that, in each case, no locations used only as transit points are required.

For all Boolean functions of one variable and of 2 variables, we can easily obtain such programs and input memory layouts as shown above, but for 4 variables, it is not known whether such programs and input memory layouts exist.

COMPUTATIONAL CAPABILITIES OF THE PROPOSED MODEL

This section deals with some of the computational capabilities of the proposed model. First, it is shown that a binary addition program exists, while there is no binary addition program in Graham's model. Next, by an argument similar to that of Graham in deriving Theorem 4, it is shown that Boolean functions of 19 or more variables exist that cannot be realized by the proposed model.

Proposition 9: A binary addition program P^+ exists in the model.

Proof: Used here is the symbolic arrangement of locations for binary addition, shown in Fig. 3. A program P_i^+ can be constructed which performs one-bit addition, as follows.

$$P_i^+ = (\bar{a}_i, \bar{b}_i)(\bar{b}_i \bar{c}_i, a_i)(a_i b_i, \bar{c}_{i+1})(a_i, b_i)(\bar{c}_{i+1} \bar{c}_i, c_i)(\bar{c}_i b_i, \bar{c}_{i+1})$$
$$\cdot (\bar{a}_i b_i, c_i)(\bar{b}_i b_i, \bar{c}_i)(\bar{c}_{i+1} b_i, c_i)(\bar{a}_i \bar{b}_i, c_i)(c_i, \bar{c}_i)(\bar{b}_i \bar{c}_i, c_i)$$
$$\cdot (\bar{c}_{i+1}, c_i, \bar{c}_i)(b_i \bar{c}_i, c_i)$$

Table II. The representatives for Boolean functions of 3 variables.

x_1	x_2	x_3	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	1	0	1	1	0
1	0	0	0	0	0	1	0	0	0	0	1	1	1	0	1	0
1	0	1	0	0	0	0	0	1	0	1	1	1	0	1	0	1
1	1	0	0	0	1	0	0	1	1	1	1	0	0	1	1	1
1	1	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1

P(f_1) = ($\bar{x}_1$, x_1), ρ(P, f_1) = $\bar{x}_1$
P(f_2) = ($\bar{x}_1$, $\bar{x}_2$) ($\bar{x}_1$, $\bar{x}_3$), ρ(P, f_2) = $\bar{x}_1$
P(f_3) = ($\bar{x}_1$, $\bar{x}_2$), ρ(P, f_3) = $\bar{x}_1$
P(f_4) = (x_2, x_3) ($\bar{x}_2$, $\bar{x}_3$) ($\bar{x}_2$, x_2) ($\bar{x}_1$, x_2), ρ(P, f_4) = x_2
P(f_5) = (x_2, x_3) ($\bar{x}_2$, $\bar{x}_3$) (x_1, x_2) ($\bar{x}_1$, $\bar{x}_2$) ($\bar{x}_1$, x_1), ρ(P, f_5) = x_1
P(f_6) = ($\bar{x}_2$, $\bar{x}_3$) ($\bar{x}_1$, $\bar{x}_3$), ρ(P, f_6) = $\bar{x}_1$
P(f_7) = ($\bar{x}_3$, x_2) (x_1, $\bar{x}_3$) ($\bar{x}_1$, $\bar{x}_2$) ($\bar{x}_1$, x_1), ρ(P, f_7) = x_1
P(f_8) = ($\bar{x}_3$, $\bar{x}_2$) (x_3, x_2) ($\bar{x}_2$, x_2) (x_1, $\bar{x}_3$) ($\bar{x}_1$, $\bar{x}_2$) ($\bar{x}_1$, x_1), ρ(P, f_8) = x_1
P(f_9) = λ, ρ(P, f_9) = $\bar{x}_1$
P(f_{10}) = (x_1, $\bar{x}_2$) ($\bar{x}_1$, x_2) ($\bar{x}_1$, x_1), ρ(P, f_{10}) = x_1
P(f_{11}) = (x_2, x_3) ($\bar{x}_2$, $\bar{x}_3$) (x_2, $\bar{x}_2$) ($\bar{x}_3$, x_3) (x_1, $\bar{x}_3$) ($\bar{x}_1$, $\bar{x}_2$) (x_1, $\bar{x}_1$), ρ(P, f_{11}) = $\bar{x}_1$
P(f_{12}) = ($\bar{x}_2$, $\bar{x}_3$) ($\bar{x}_1$, $\bar{x}_2$) (x_1, $\bar{x}_3$) ($\bar{x}_1$, x_1), ρ(P, f_{12}) = x_1
P(f_{13}) = (x_1, $\bar{x}_3$) (x_1, $\bar{x}_2$) ($\bar{x}_2$, x_3) ($\bar{x}_1$, x_3) ($\bar{x}_1$, x_1), ρ(P, f_{13}) = x_1
P(f_{14}) = ($\bar{x}_3$, $\bar{x}_2$) (x_2, x_3) ($\bar{x}_1$, $\bar{x}_2$) (x_1, x_2) ($\bar{x}_1$, x_1), ρ(P, f_{14}) = x_1

$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$
x_1	x_2	x_3

Input memory layout for P(f_1), P(f_2), P(f_3), P(f_4), P(f_5), P(f_6), P(f_7), P(f_9), P(f_{10}) and P(f_{14}).

$\bar{x}_1$	$\bar{x}_2$	x_2
x_1	$\bar{x}_3$	x_3

Input memory layout for P(f_8), P(f_{11}) and P(f_{12}).

$\bar{x}_3$	x_1	$\bar{x}_1$
x_2	$\bar{x}_2$	x_3

Input memory layout for P(f_{13}).

Figure 6—Programs and input memory layouts for the representatives for Boolean functions of 3 variables

Consequently, in general, a program $P^+(m)$ which performs m-bits addition, is given by

$$P^+(m) = P_1{}^+P_2{}^+ \ldots P_m{}^+(\bar{c}_{m+1}, c_{m+1})$$

where $(\bar{c}_{m+1}, c_{m+1})$ comes from the carry.

The capability of Boolean function realization by the proposed model is now considered. Let $f(x_1, x_2, \ldots, x_m)$ and $g(x_1, x_2, \ldots, x_m)$ be two Boolean functions. If $f(x_1, x_2, \ldots, x_m) \leqq g(x_1, x_2, \ldots, x_m)$ for all 2^m inputs, then it is said that f implies g, and written as $f \subseteq g$.

Let $f_a(t)$, $f_b(t)$ and $f_c(t)$ denote the Boolean functions realized at locations a, b and c, respectively, after applying the t-th command e_t of program P. It is immediately apparent that, if $f_a(t) \subseteq f_b(t)$, then the application of the command (a, b) as the $(t+1)$-th command of the program P changes nothing. Hence, it can be assumed that only the command (a, b) is used for which, at the time of their application, $f_a(t) \nsubseteq f_b(t) \nsubseteq f_a(t)$ (it is said that the pair $(f_a(t), f_b(t))$ is command-applicable). Similarly, if $f_a(t)f_b(t) \subseteq f_c(t)$, then the application of the command (ab, c) or $ba, c)$ as the $(t+1)$-th command of program P changes nothing. Hence, it can also be assumed that only commands (ab, c) and (ba, c) are used for which, at the time of their application, $f_a(t)f_b(t) \nsubseteq f_c(t)$ (It is said that the triplet $(f_a(t), f_b(t), f_c(t))$ is command-applicable). Define $D(t) = \{f_{x_i}(t) \mid 1 \leqq i \leqq k\}$. It is assumed that exactly r of the $\binom{k}{2}$ $(k-2)$ triplets are not command-applicable, and $f_{x_i}(t)f_{x_1}(t) \subseteq f_{x_2}(t)$, for some i, $i \neq 1, 2$. Then, after the application of the command (x_ix_1, x_2) as the $(t+1)$-th command of the program P, we get

$$f_{x_i}(t+1) = f_{x_i}(t)$$

$$f_{x_1}(t+1) = f_{x_1}(t)f_{x_2}(t) + f_{x_1}(t)\bar{f}_{x_i}(t)$$

$$f_{x_2}(t+1) = f_{x_i}(t)f_{x_1}(t) + f_{x_2}(t)$$

Therefore,

$$f_{x_i}(t+1) = f_{x_1}(t+1) \subseteq f_{x_2}(t+1).$$

Let

$$D(t+1) = \{f_{x_1}(t)f_{x_2}(t) + f_{x_1}(t)\bar{f}_{x_i}(t), f_{x_i}(t)f_{x_1}(t) + f_{x_2}(t), f_{x_3}(t), \ldots, f_{x_k}(t)\}.$$

Now, let us examine how many triplets of the functions of $D(t+1)$ are command-applicable.

Four cases arise:

For any $j \neq 1, 2$,

(1) $f_{x_j}f_{x_i} \subseteq f_{x_2}, f_{x_j}f_{x_2} \subseteq f_{x_1}$.
Then, $f_{x_j}(f_{x_1}f_{x_2} + f_{x_1}\bar{f}_{x_i}) \subseteq f_{x_i}f_{x_1} + f_{x_2}$
$f_{x_j}(f_{x_i}f_{x_1} + f_{x_2}) \subseteq f_{x_1}f_{x_2} + f_{x_1}\bar{f}_{x_i}$

(2) $f_{x_j}f_{x_1} \subseteq f_{x_2}, f_{x_j}f_{x_2} \subseteq f_{x_1}$.
Then, a) if $f_{x_j} = f_{x_1} = f_{x_i} = 1$ and $f_{x_2} = 0$,
$f_{x_j}(f_{x_1}f_{x_2} + f_{x_1}\bar{f}_{x_j}) \subseteq f_{x_1}f_{x_i} + f_{x_2}$
b) otherwise,
$f_{x_j}(f_{x_1}f_{x_j} + f_{x_2}) \subseteq f_{x_1}f_{x_2} + f_{x_1}\bar{f}_{x_i}$.

(3) $f_{x_i}f_{x_1} \subseteq f_{x_2}, f_{x_i}f_{x_2} \nsubseteq f_{x_1}$.
Then, $f_{x_j}(f_{x_1}f_{x_2} + f_{x_1}\bar{f}_{x_i}) \subseteq f_{x_1}f_{x_i} + f_{x_2}$

(4) $f_{x_1}f_{x_2} \subseteq f_{x_j}$.
Then, $(f_{x_1}f_{x_2} + f_{x_1}\bar{f}_{x_i})(f_{x_1}f_{x_i} + f_{x_2}) \subseteq f_{x_j}$.

Consequently, at least $r+1$ triplets of the functions of $D(t+1)$ are not command-applicable. It is known that, if exactly r of $\binom{k}{2}$ pairs of the functions of $D(t)$ are not command-applicable, after the application of the command (a, b) as the $(t+1)$-th command of the program, at least $r+1$ pairs of the functions of $D(t+1) = \{f_{x_1}(t) \cdot f_{x_2}(t), f_{x_1}(t) + f_{x_2}(t), f_{x_3}(t), \ldots, f_{x_k}(t)\}$ are not command-applicable, where it is assumed that the pair $(f_{x_1}(t), f_{x_2}((t))$ is command-applicable.[2] Therefore, we have the following result.

Lemma 10: For any Boolean function f of m variables, it is possible to put

$$\lg(P(f)) \leqq \left[\binom{2m}{2} - m\right](2m-2) + \binom{2m}{2}$$

Proof: In the initial state, the number of triplets which are command-applicable is $[\binom{2m}{2} - m](2m-2)$ and that of the pairs is $\binom{2m}{2}$.

Now we show the main result:

Theorem 11: Boolean functions of 19 or more variables exist which cannot be realized by the model.

Proof: Let

$$N=\left[\binom{2m}{2}-m\right](2m-2)+\binom{2m}{2}.$$

Consider any program $P=e_1e_2 \ldots e_r$. In choosing the i-th command e_i of P, there are, at most, $N-i+1$ possibilities for e_i, since, after $e_1e_2 \ldots e_{i-1}$ has been applied, at least $i-1$ of the pairs and the triplets are not command-applicable. Therefore there are, at most,

$$\prod_{i=1}^{N}(N-i+1)=N!$$

choices for the sequence of e_i, since $t \leqq N$ by Lemma 10. Furthermore, by the application of one command, at most two new Boolean functions are generated. Hence there are, at most,

$$2N(N!)+2m$$

Boolean functions which can be generated. On the other hand, the total number of Boolean functions of m variables is 2^{2^m}. These expressions are listed for $m=18$ and $m=19$ in Table 3. Hence the theorem follows.

Table III. Bounds on the number of Boolean functions which can be generated.

m	2N (N!) + 2m	2^{2^m}
18	$>10^{83,520}$	$<10^{78,914}$
19	$<10^{100,536}$	$>10^{157,826}$

SUMMARY

Some properties of a simple bubble model proposed by Graham have been investigated and the programs realizing all Boolean functions of 3 variables and the corresponding input memory layouts with the minimum total locations have been presented. This paper has extended Graham's model by introducing a different type of command, i.e., the conditional bubble transfer. It has been shown that, in the proposed model, a binary addition program exists, and also Boolean functions of 19 or more variables exist which cannot be realized.

ACKNOWLEDGMENTS

The author would like to thank Dr. K. Noda for his support of the writing of this paper, and Dr. N. Ikeno and Dr. G. Nakamura for their valuable suggestions and many discussions.

REFERENCES

1. Bobeck, A. H., "Properties and Device Applications of Magnetic Domains in Orthoferrites," *Bell System Technical Journal*, Vol. 46, No. 8, 1967.
2. Graham, R. L., "A Mathematical Study of a Model of Magnetic Domain Interactions," *Bell System Technical Journal*, Vol. 49, No. 8, 1970.
3. Friedman, A. D., Menon, P. R., "Mathematical Models of Computation Using Magnetic Bubble Interactions," *Bell System Technical Journal*, Vol. 50, No. 6, 1971.
4. Minnick, R. C., Bailey, P. T., Sandfort, R. M., Semon, W. L., "Magnetic Bubble Computer Systems," *Proceedings of Fall Joint Computer Conference*, 1972.

The realization of symmetric switching functions using magnetic bubble technology

by H. CHANG,* T. C. CHEN and C. TUNG

IBM Research Laboratory
San Jose, California

INTRODUCTION

Since its debut in the late sixties, magnetic bubble technology has quickly evolved to become a promising alternative to semiconductor technology for computer construction.[1-7] While emphasis has been placed upon the application of bubble technology to storage, its application to logic has thus far been largely limited to the implementation of simple basic operators, such as AND, OR, etc.[5,7] An exception is the excellent work recently reported in References 12 and 13. This limitation to simple basic operators, however, is not in keeping with the high density and low connectivity requirements of LSI (Large-Scale Integration), and it has become increasingly important to find powerful multi-input switching functions. The symmetric function is a natural candidate. The realization of symmetric functions using magnetic bubble technology has been found to be very simple.

The second part of this paper provides some basic information for a qualitative understanding of the bubble technology. Part three briefly reviews symmetric functions and also introduces residue threshold functions. Part four describes the mechanism for realizing symmetric functions, and part five presents an implementation. Some concluding remarks are made in part six.

MAGNETIC BUBBLES

Fundamentals

Basic to magnetic bubble devices is the existence of magnetic domains in uniaxial magnetic materials over a range of bias field. The magnetic domain is a cylindrical region in a garnet film or an orthoferrite platelet—hence the name bubble—with magnetization perpendicular to the plane of film or platelet and opposite to that in the surrounding region. This configuration, Figure 1, is achieved when the film or the platelet has uniaxial magnetic anisotropy to orient the magnetization perpendicular to the plane, has sufficiently low magnetization to prevent the demagnetizing field to force the magnetization into the plane, and has a bias field opposite to the bubble magnetization direction to prevent the bubble from expanding into serpentine domains—the natural demagnetized state.

Several features are noteworthy for device applications:

(i) Stable bubbles exist over a range of bias field strengths, thus exhibiting storage capability.
(ii) A bubble can be deformed by lowering the bias field for further manipulation, e.g., bubble generation, replication, etc.
(iii) A bubble can be annihilated by raising the bias field.
(iv) Bubbles interact with one another like magnets when they get closer than about three diameters. These interactions limit storage density, but are necessary for logic circuits implementation.

In the past, more attention has been given to the application of bubble technology to data storage than to data processing. The most popular configuration of bubble storage, by far, is the shift register with bubbles (representing ONE's) and voids (representing ZERO's) propagating along fixed tracks.

Propagation

The transmission and manipulation of information rely, directly or indirectly, on the propagation of bubbles.

There are two basic methods of producing movement of bubbles in the plane. The first method employs the current in a conductor loop to produce a field for attracting an adjacent bubble. A sequence of bubble positions may be propagated by exciting a series of conductor loops wired to carry current pulses. This is referred to as "conductor propagation."

The second method, "field access propagation," depends on the alternating magnetic poles in a patterned permalloy overlay; the poles are induced by a rotating field in the plane. A permalloy bar is easily magnetized by this field along its long direction. When suitable permalloy patterns are subjected to this rotating field, the induced magnetism in the form of a moving train of poles pulls the attracted bubbles along. Since field access propagation is more suitable for the implementation discussed in this paper, it will be examined in more detail.

The integers 1, 2, 3 and 4 will be used to denote the four phases of the rotating field, counterclockwise starting from the first quadrant, as shown in Figure 2.

* With T. J. Watson Research Center, Yorktown Heights, New York.

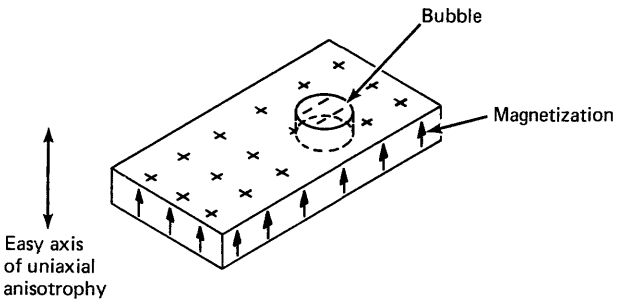
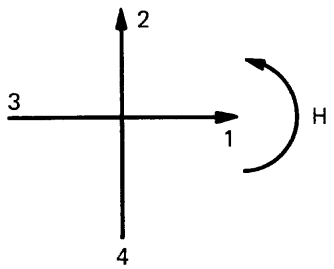


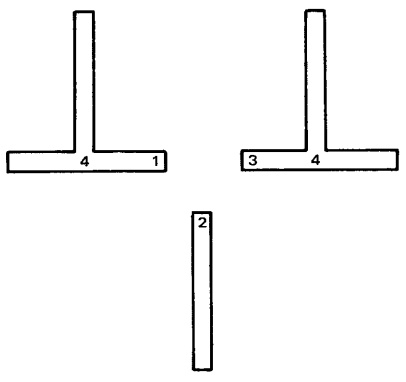
Figure 1—A bubble is a cylindrical magnetic domain with magnetization opposite to that of its surrounding. It exists in a thin film of uniaxial anisotropy, under proper bias field

The permalloy pattern shown in Figure 3 will guide the bubble propagation from left to right. As the field rotates from 1 to 2, for instance, the upper end of the vertical I-bar to the right of the current bubble position will be magnetized positively and thus be able to attract the negative end of the bubble toward the right. The entire bubble moves as a result.

Information is carried by a stream of bubbles and voids (vacancies), conventionally designated to represent 1 and 0, respectively. As each bubble in the stream moves by a unit



(a) Rotating Field



(b) Corresponding Positions of Induced Positive Magnetic Poles

Figure 2—Labelling convention for the four phases of a rotating field and their corresponding positions of induced positive magnetic poles

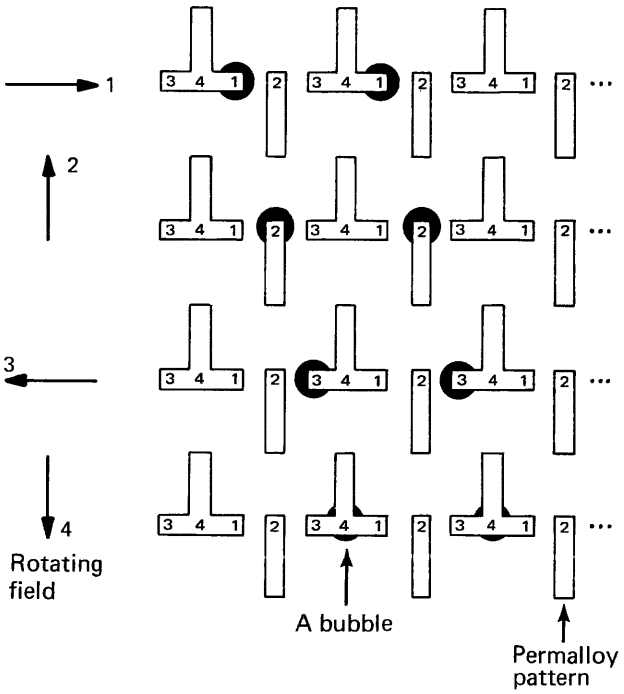


Figure 3—Bubble propagation—A *T-I* bar permalloy pattern propagates bubbles by moving magnetic poles induced in a rotating field

distance, the voids in between, if any, will have an apparent movement also by a unit distance. Thus the entire stream flows at a regular rate in response to the periodic magnetization of the regular TI permalloy pattern. When the permalloy pattern like the one shown in Figure 3 is arranged in a loop, a shift register memory results.

An idler is a cross-like permalloy pattern as shown in Figure 4. In the absence of extraneous influence, the bubble in an idler will circulate indefinitely; it is movable by, for example, a suitably positioned repelling bubble or magnetism induced by a wire loop nearby. Thus, without the external influence of magnetic force other than the rotating field, a vacant idler position serves as a bubble trap, and a filled

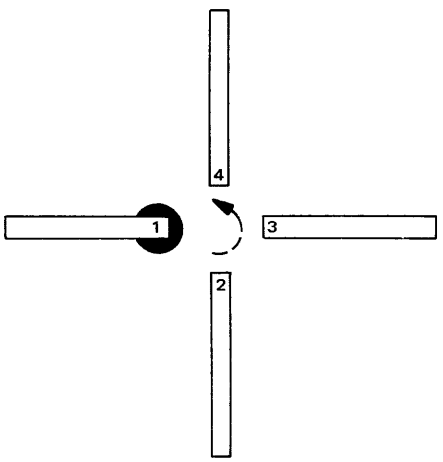


Figure 4—An idler circulates a bubble within a permalloy cross

idler position appears like a "short-circuit" insofar as bubble propagation is concerned; a stream of bubbles and voids in a tandem of idlers may thus be able to remain stationary in the presence of the rotating field.

Other forms of permalloy patterns can be used, notably *Y*-patterns and "angelfish." [8]

Interaction

The magnetostatic interaction between bubbles is essential to the generation of logic functions. Bubble domain media such as orthoferrite platelets and garnet films have very low wall motion threshold. Hence the flux lines emanated from a bubble are adequate to move an adjacent bubble as far as two or three bubble diameters away.

Figure 5 shows how bubbles can interact with each other to generate two logic functions.[5] The device shown in Figure 4 has two streams of input variables, namely A and B. The presence or absence of a bubble (also respectively called bubble and void) represents the true or false value of a variable. The lower bubble, representing A, will propagate toward the output terminal labelled $A \vee B$, independent of bubble B. If bubble B is currently at the position shown, then one quarter cycle later it may take one of the two possible paths, depending on the presence of bubble A. With bubble A being where it is as shown, the interaction of these two bubbles will force bubble B to propagate to 4′ rather than to 4, one quarter cycle later. The information that both A and B bubbles are present is conveyed by the propagation of bubble B toward the output terminal labelled $A \wedge B$. With the absence of bubble A, bubble B will propagate downward to the output terminal labelled $A \vee B$. In addition to the interaction of bubbles, one can clearly see that the precise timing of bubble propagation is crucial to the proper generation of logic functions.

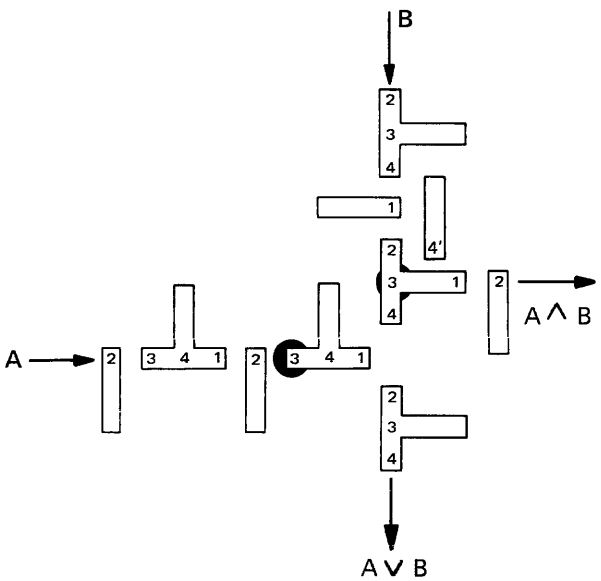


Figure 5—Bubble logic—A permalloy pattern brings bubbles together to interact magnetostatically and thereby generates logic functions

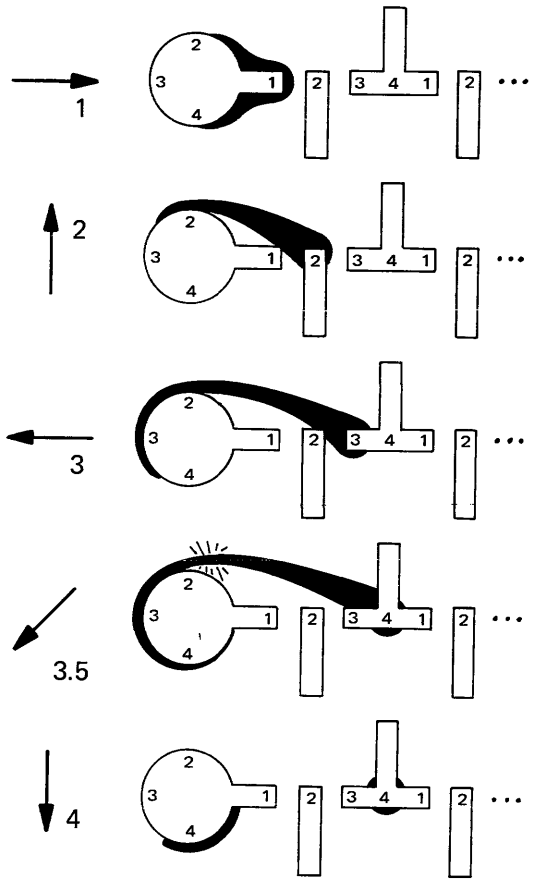


Figure 6—Bubble generation—A permalloy pattern generates new bubbles by stretching and then severing a mother bubble into halves

The realization of other logic elements such as binary full adder has been demonstrated to be feasible.[8]

Generation and annihilation

A permanent bubble associated with the generator disk, shown in Figure 6, is forced to stretch when one end becomes trapped during phase 2 of the planar rotating field. As the planar field keeps rotating, the bubble is further stretched. Between phases 3 and 4 its thinly stretched position will sever into two, leaving a newly formed bubble to the right of the generator disk.

The annihilation of a bubble can be achieved by arranging the permalloy pattern as shown in Figure 7. During phases 3 and 4, the bubble remains essentially in the same position. During phase 1, the bubble is severely weakened because the attracting pole of the permalloy pattern is remote; yet the repelling one is near and strong, thus annihilating the bubble.

SYMMETRIC SWITCHING FUNCTIONS

Symmetric switching functions

The function

$$S(A \mid X) \equiv S(a_1, a_2, \ldots, a_m \mid x_1, x_2, \ldots, x_n)$$

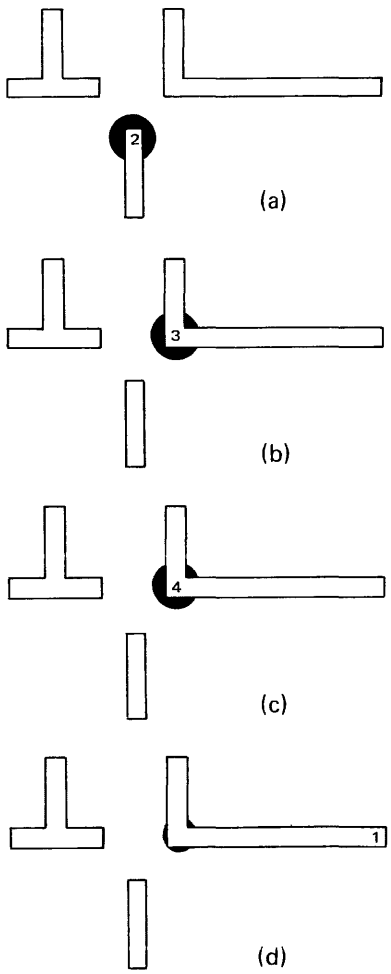


Figure 7—Bubble annihilation—A permalloy pattern propagates a bubble to a trapping position where the bubble is collapsed upon field reversal

with

a_i: an integer and $0 \leq a_i \leq n$

x_j: 0 or 1 (switching variable)

is said to be a symmetric switching function when and only when the sum over X equals one of the values in A, i.e.,

$$S(A \mid X) = 1 \text{ if } \sum_i X_i = \text{one of the values in } A$$
$$= 0 \text{ otherwise.}$$

The values in A are commonly called a-numbers. It is clear that a permutation of the vector X does not change the value in S, hence the term "symmetric." [9-10]

Symmetric switching functions can be used to synthesize other logical functions with a proper selection of a-numbers. In fact, the symmetric function is a universal logical connective since the commonly used universal set of logical connectives, AND, OR, and NOT can be synthesized trivially:

$$X_1 \wedge X_2 = S(2 \mid x_1, x_2)$$
$$X_1 \vee X_2 = S(1, 2 \mid x_1, x_2)$$
$$\bar{X}_1 \quad = S(0 \mid x_1).$$

As further examples, the popular NAND and NOR functions, each of which is a universal logical connective in its own right, can be synthesized by symmetric functions, at no increase in complexity, as follows:

$$\overline{X_1 \wedge X_2} = S(0, 1 \mid x_1, x_2)$$
$$\overline{X_1 \vee X_2} = S(0 \mid x_1, x_2).$$

As examples of synthesizing practical circuits, the binary adder with x, y, z as its input operands and input carry can have its outputs, carry and sum, defined by:

$$\text{carry} = S(2, 3 \mid x, y, z)$$
$$\text{sum} = S(1, 3 \mid x, y, z).$$

Residue threshold functions

A subset of symmetric functions, called residue threshold functions, has been recently studied.[11] Given n switching variables $x_1, x_2, \ldots, x_n$, m a positive integer, and t a non-negative integer, the residue threshold function is defined as

$$R(t, m \mid x_1, \ldots, x_n) \equiv R(t, m \mid X) \equiv t \leq (\sum_i X_i) \text{ Mod } m$$

that is,

$$R(t, m \mid X) = 1 \quad \text{if and only if} \quad (\sum_i X_i) \text{ Mod } m \geq t.$$

Here, $(\sum X_i)$ Mod m is defined to be the least positive remainder of $(\sum X_i)/m$, which is a number between 0 and m-1 inclusively.

The relationship between the symmetric switching function and the residue threshold function is very simple.

$$R(t, m \mid X)$$

is equivalent to

$$S(A \mid X),$$

with A containing all positive integers, a_i's $(a_i \leq n)$ such that

$$t \leq a_i \text{ Mod } m.$$

As noted before, the symmetric function derives its powerful capability of synthesizing any other logical function from the "personalization" of its a-numbers. In practice, the a-numbers are usually much more structured than a mere enumeration would indicate, and a common structure is the cyclicity. An example here may help clarify this point. To find the parity of a sequence of bits, one needs only to know whether the number of ONE's in the sequence is even or odd. The exact number of ONE's is immaterial. Thus, instead of specifying

$$S(1, 3, 5, 7, \ldots \mid X),$$

one needs only to specify

$$R(1, 2 \mid X).$$

The underlying structure permits a significantly simplified implementation as will be seen in a later section of this paper.

STEPS TOWARD REALIZING SYMMETRIC SWITCHING FUNCTIONS

Based on economical considerations, the progress of LSI of solid state devices is measured in terms of the ever in-

creasing device density and chip area, hence the number of devices per chip. One should note that as the area for devices is increased by a factor of m^2, the periphery for interconnections is increased only by a factor of m. Thus the merit of LSI can best be improved by increasing the versatility of devices to permit self-sufficiency within the chip and to reduce the number of external interconnections required for input/output, control and power. Bubble domain devices with shift-register type of memory capability and symmetric switching function type of logic capability appear to be an attractive candidate for LSI.

Here we consider the mechanisms required for the realization of symmetric switching functions using magnetic bubble technology. Implementation based on these observations will be discussed in the next section.

Given a sequence of bubbles and voids, X, consider:

(i) Bubble sifting: since the symmetric function is invariant to permutation, one can sift the bubbles in X to result in a new sequence Y in which bubbles gravitate to one end (say, left end) of Y. For instance, if X is 0 1 0 1 then Y would be 1 1 0 0.

(ii) Leading bubble detection: there exists a simple relationship between the position of the leading bubble (rightmost bubble) in Y and the number of bubbles in either X or Y. This relationship is

$$m=n+1-p, \quad \text{or} \quad p=n-m+1,$$

where m is the number of bubbles, n the length of X or Y, and p the position of the leading bubble (1-origin, left indexed) in Y. For the case $m=0$, p will be considered $n+1$ as the above formula dictates. In practice, one can augment Y into Z by appending a 1 to the left of Y, in order to accommodate the $m=0$ case. At the end of this leading bubble detection stage, one obtains a bubble stream W in which there is one and only one bubble.

(iii) Interaction with the control bubble stream: a control stream of bubbles and voids can be constructed with the positions of bubbles representing the a-numbers. That is, $A=a_1, \ldots, a_m$ is represented by

$$B=b_0, b_1, \ldots, b_n$$

such that

$$b_i=1 \quad \text{if} \quad i \quad \text{is in } A$$

$$=0 \text{ otherwise.}$$

By proper timing, the information coming out of the stage of leading bubble detection (with bubble/void representing that the leading bubble has/has not been detected) is required to AND with the components of B. At any time during ANDing a 1 output indicates that the number of ONE's in X agrees with one of the a-numbers. Therefore, $S(A \mid X) = 1$ if and only if ANDing of the control stream and the output from the leading bubble detection stage yields a true output.

The mechanism described above is summarized in a block diagram shown in Figure 8.

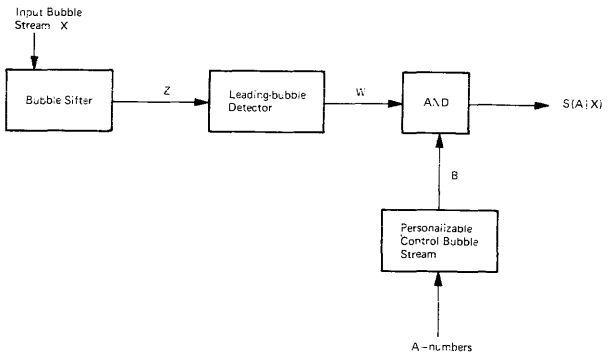


Figure 8—A block diagram showing required bubble mechanisms to realize symmetric switching functions

Example 1:

$X=$ 0 1 0 1 $\quad A=0, 2, 3$
$Y=$ 1 1 0 0 $\quad B=1\,0\,1\,1\,0$
$Z=1\,1\,1\,0\,0$
$W=0\,0\,1\,0\,0$
ANDing between W and B
$W=0\,0\,\boxed{1}\,0\,0$
$B=1\,0\,\boxed{1}\,1\,0$
$\downarrow$
T hence $S(A \mid X)=1$

Example 2:

$X=$ 0 0 0 0 $\quad A=3$
$Y=$ 0 0 0 0 $\quad B=0\,0\,0\,1\,0$
$Z=1\,0\,0\,0\,0$
$W=1\,0\,0\,0\,0$
ANDing between W and B
$W=\boxed{1}\,0\,0\,0\,0$
$B=\boxed{0}\,0\,0\,1\,0$
$\downarrow$
F hence $S(A \mid X)=0$

IMPLEMENTATION

Figure 9 shows the detailed permalloy pattern implementing the scheme depicted in Figure 8. It consists of four re-

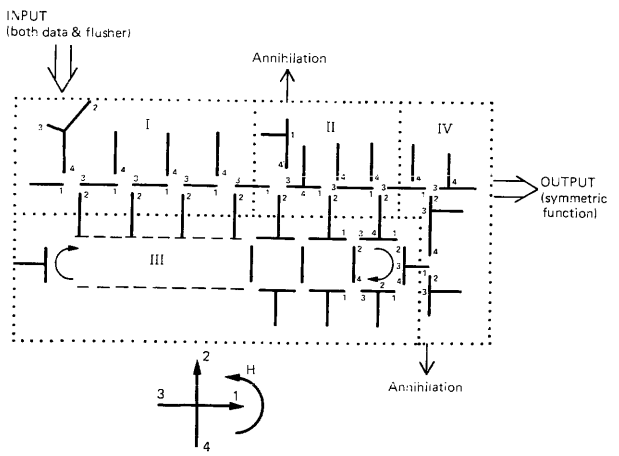


Figure 9—A permalloy pattern to implement the block diagram in Figure 8

gions, with region I being the sifter, region II the leading bubble detector, region III the control bubble stream, and region IV the AND gate.

The bubble sifter (region I) contains n idlers in tandem, with the leftmost one connected to the vertical input channel. The idlers within the sifter are slightly offset in such a way that it favors bubble movement toward the right. A bubble in the idler tandem, however, will not move to the right unless all the idler positions to its left are filled with bubbles, and there is a bubble entering at the input channel trying to push all the bubbles in the sifter. The net effect is that voids will skip all preceding bubbles. The input stream X as defined before will, after n cycles, become the n-bit stream Y residing in the sifter with, say, m bubbles juxtaposed with $(n-m)$ voids to their right.

Without further external influence, the Y stream will stay in the sifter indefinitely. The entering of an $(n+1)$-bubble flushing stream at the input channel at this time will drive the Y stream to the right for leading bubble detection. The first bubble in the flushing stream and the Y stream form the augmented stream designated as Z in the previous section.

Initially, the leading bubble detector (region II) contains a resident bubble in its right idler. As the very first bubble from the sifter arrives at the center idler, the resident bubble will be repelled to the right into the AND gate (region IV), conveying the information that the leading bubble of the Z stream has arrived. The bubble in the center idler is trapped there and, through its magnetostatic force, will divert all the following bubbles from the sifter upward to an annihilator.

A bubble generator (not shown in Figure 9) will issue bubbles, according to the a-numbers of the given symmetric function, into region III which is a closed loop shift register. This circulating bubble stream is the personalized control bubble stream B discussed previously.

The description of detailed operations in terms of bubble interactions with the permalloy pattern is now in order. The sequence of events together with the time intervals is shown in Figure 10. For illustration, we assume that the input is 0 1 0 1 with the rightmost position being the leading position, followed by a sequence of five flushing bubbles. The bubbles in the input stream (X or Y), called data bubbles,

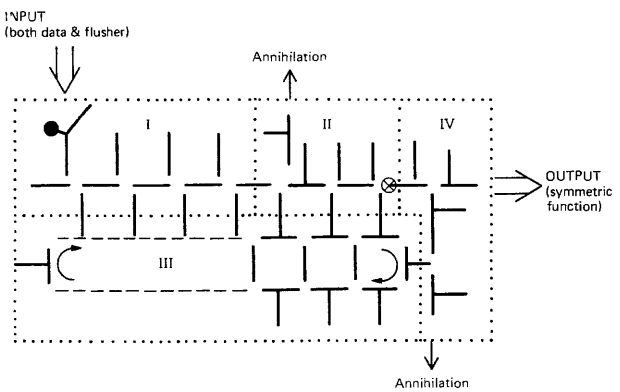


Figure 10—Sequence of key events in the symmetric-function bubble device: (a)—$t=0$. The device is cleared. A resident bubble is loaded into the first bubble detector. The first data bit (a ONE, i.e., a bubble) of the input bubble stream (0101) is ready to enter the sifter.

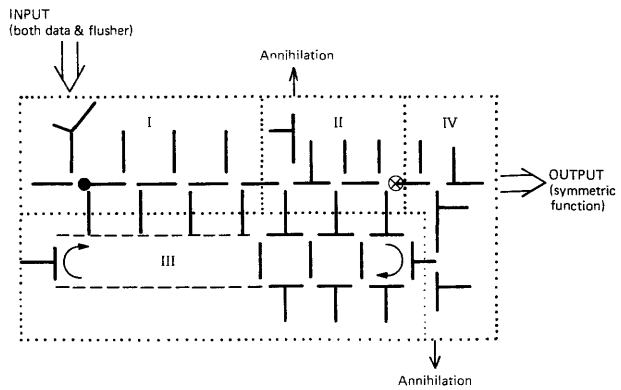


Figure 10(b)—$t=2$. The first data bit is trapped at the first position (leftmost idler) while the second data bit (a void) has skipped by. This demonstrates the sifting function.

are represented by solid dots, the flushing bubbles by circles, and the resident bubble by a circle with a cross in it.

The initial ($t=0$) status is shown in Figure 10-a, the corresponding field phase is 3 (see Figure 2). At $t=1$, the first data bubble has advanced to the leftmost idler, and at the input channel is the void. One cycle later, $t=2$, the first data bubble still remains at the same position; thus it can be said that the void has skipped the bubble preceding it and now resides at the second idler from the left. At the same time, there comes the second data bubble at the input channel. At $t=2\frac{1}{4}$, the second data bubble moves downward and repels the first data bubble toward the right, causing it temporarily out of synchronization. At $t=2\frac{3}{4}$, the first data bubble is attracted to position 1 of the second idler from the left, thus re-synchronized with the rotating field. The above sequence of operations in the last three field phases of a cycle is shown in Figure 10-c.

Figure 10-d shows the status at $t=4$; the two data bubbles are residing at the two idlers at the left and the two voids can be thought to have skipped the bubbles and advanced to the two idlers at the right. The first flushing bubble is now at the input channel. For this example, it takes two more cycles to fill up the sifter and the input channel, and two additional cycles for the leading data bubble to be flushed out of the sifter and propagated to the center idler of the

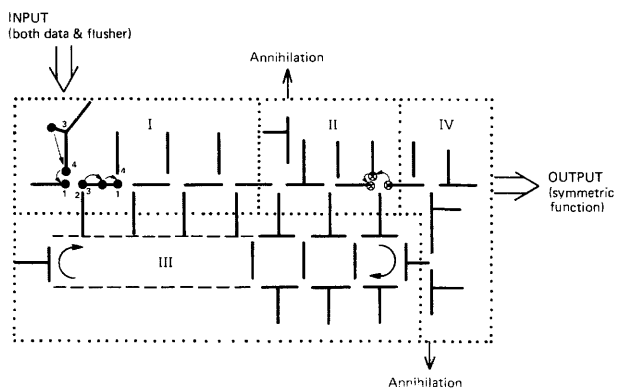


Figure 10(c) · $t=2,2\,1/4, 2\,2/4$. The third data bit (a bubble) pushes the first data bit (a bubble) to the second position

detector. The interactions at $t=8$ and $t=8\frac{1}{4}$ are shown in Figure 10-e. As time advances from $t=8$ to $t=8\frac{1}{4}$, the presence of the first data bubble in the detector causes the resident bubble to move to position 4′ in region IV, and the bubble following it to move upward to position 4′ in region II which leads to an annihilator. As the leading data bubble is trapped in the center idler, all following bubbles will be diverted upward to the annihilator. Consequently, during the whole operation one and only one bubble will leave region II and enter region IV.

Half a cycle later, $t=8\frac{3}{4}$, the resident bubble is at position 2 in region IV, shown in Figure 10-f. If, at this instant, there is a bubble in region III at the position indicated by a small square, the resident bubble will be forced to move to position 3′ in region IV at $t=9$, giving an indication that the symmetric function is true. Otherwise, the resident bubble will move downward in region IV and be annihilated eventually. It is clear now that the upper portion of region IV behaves as an AND gate.

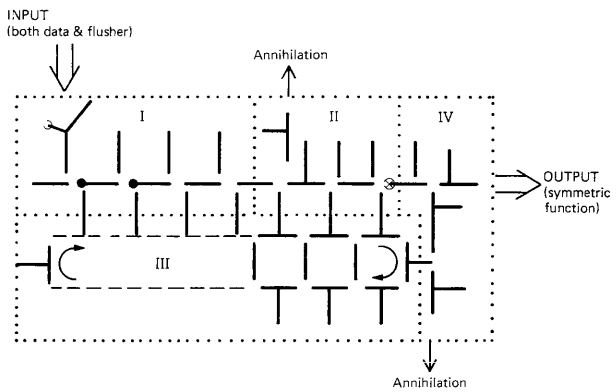


Figure 10(*d*)—$t=4$. The flusher bubbles are ready to enter the sifter

In general, with an n-bit (bubble/void) input having m bubbles, the critical ANDing time, t_a, ($t_a=8\frac{3}{4}$ in the case discussed above) between the resident bubble and the control bubble stream is

$$t_a=n+(n-m)+2+\tfrac{3}{4}=(2n-m)+2\tfrac{3}{4},$$

of which n cycles are required to load and sift the data bubbles and voids in the sifter (region I), $n-m$ cycles required to fill the sifter, 2 more cycles required to propagate the rightmost bubble in the sifter to the center idler of region II, and finally $\frac{3}{4}$ cycle required to move the resident bubble to position 2 of region IV.

It can be easily deduced from the above formula that if the resident bubble cannot be found at position 3′ in region IV before or at

$$t_a\,|_{m=0}+\tfrac{1}{4}=2n+3,$$

then the symmetric function is false. In other words, the operation time, excluding initialization, of this device is $2n+3$ cycles.

We have shown a bit-serial implementation. If each idler in the sifter is directly connected with an input channel, the parallel input operation can be performed to gain speed. This

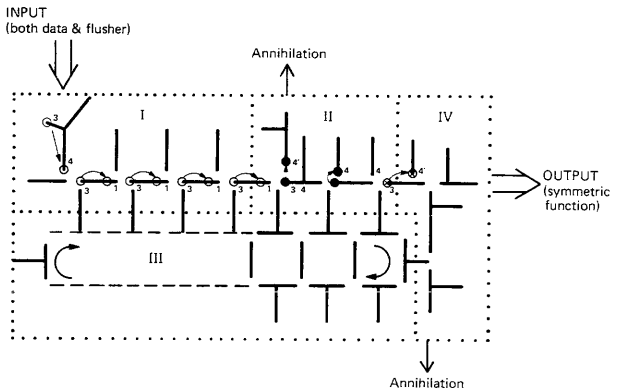


Figure 10(*e*)—$t=8$, 8-1/4. The leading data bubble has entered the center idler of the leading bubble detector, pushing the resident bubble into the AND-gate. The leading data bubble is trapped in the center idler, diverting the following bubbles to an annihilator

is because the sifting can be performed during flushing time; no data bubble can leave the sifter until all idlers in it have been filled. Assuming that the flushing bubbles are allowed to enter at the leftmost input channel, we find

$$t_a=1+(n-m)+2+\tfrac{3}{4}=(n-m)+3\tfrac{3}{4}.$$

and the operation time for this parallel input is thus $n+4$.

In many applications, the a-numbers are well structured and thus help simplify the control bubble stream significantly. As we discussed earlier, the parity check of a sequence of bits, X, can be expressed as

$$R(1, 2 \mid X).$$

To realize this, the control bubble stream needs only to consist of one bubble and one void in a tight loop, saving much space.

CONCLUSION

We have shown how to realize symmetric switching with magnetic bubble devices. The implementation is simple, yet

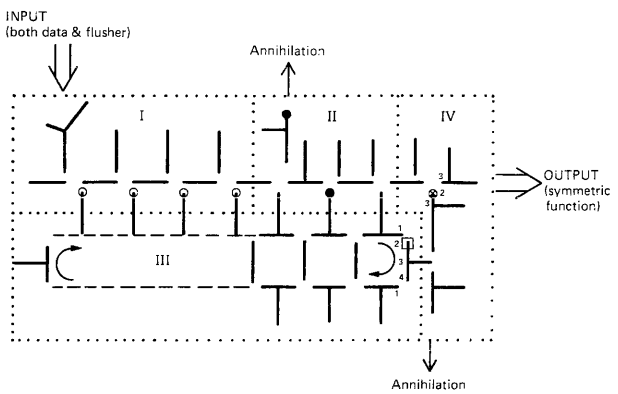


Figure 10(*f*)—$t=8$-3/4. The resident bubble is at a position to interact with the control bubble streams. The presence of a control bubble at the square will force the resident bubble to move to 3′ leading to the output, indicating that the symmetric function is true. The absence of a control bubble will permit the resident bubble to move to 3 and then the annihilator, indicating that the symmetric function is false

with the easily achievable personalization of the control bubble stream it produces a versatile and powerful logic device. Our a-numbers stream is a simple example of the personalization through a circulating shift register memory. The personalization persists as long as the vertical bias magnetic field is present.

Both bubble memory and bubble logic devices are implemented with very similar permalloy patterns, hence it is possible to have a mixture of memory and logic at a very local scale. Such a mixture is particularly attractive because of its low cost and low power dissipation. Note that traditionally memory and logic are in separate units. For example, the ferrite core memory and semiconductor central processing unit are separate, because of different technologies. In semiconductors memory and logic are separate, partly because of the density contrast of repetitive cells in memory versus a great variety of cells in logic; and more importantly because read-write memory is volatile, and logic must use a more nondestructive implementation. Thus circuit logic resembles read-only memory, and tends to be different from read-write memory in construction. The magnetic disks, drums, and tapes simply do not have any resident logic capability, and must rely on external logic circuits (control unit, channels, etc.) for data routing and data management. With the capability of an intimate mix of memory and logic, much of the previous demarcation lines can be removed. The design optimization should be greatly facilitated. In fact, the hardware capability may induce revolutionary changes in computer organization and architecture.

REFERENCES

1. Bobeck, A. H., Fischer, R. F., Perneski, A. J., Remeika, J. P., Van Uitert, L. G., "Application of Orthoferrites to Domain Wall Devices," *IEEE Trans. Magnetics 5*, 3, September 1969, pp. 544-553.
2. Perneski, A. J., "Propagation of Cylindrical Magnetic Domains in Orthoferrites," *IEEE Trans. Magnetics 5*, 3, September 1969, pp. 554-557.
3. Thiele, A. A., "Theory of Static Stability of Cylindrical Domains in Uniaxial Platelets," *J. Appl. Phys. 41*, 3, March 1970, pp. 1139-1145.
4. Bonyhard, P. I., Danylchuk, I., Kish, D. E., Smith, J. L., "Application of Bubble Devices," *IEEE Trans. Magnetics 6*, 3, September 1970, pp. 447-451.
5. Sandfort, R. M., Burke, E. R., "Logic Functions for Magnetic Bubble Devices," *IEEE Trans. Magnetics 7*, September 1971, pp. 358-360.
6. Ahamed, S. V., "The Design and Embodiment of Magnetic Domain Encoders and Single-Error Correcting Decoders for Cyclic Block Codes," *B.S.T.J. 51*, 2, February 1972, pp. 461-485.
7. Garey, M. R., "Resident-Bubble Cellular Logic Using Magnetic Domains," *IEEE Trans. Computers C-21*, 4, April 1972, pp. 392-396.
8. Bobeck, A. H., Scovil, H. E. D., "Magnetic Bubbles," *Scientific American 224*, 6, June 1971, pp. 78-91.
9. Harrison, M. A., *Introduction to Switching and Automata Theory*, McGraw-Hill, New York, 1965.
10. Kohavi, Z., *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1970.
11. Ho, I. T., Chen, T. C., "Multiple Addition by Residue Threshold Functions," *IEEE CompCon Proceedings*, September 1972.
12. Minnick, R. C., Bailey, P. T., Sanfort, R. M., Semon, W. L., "Magnetic Bubble Computer Systems," *AFIPS Proceedings*, Vol. 41, December 1972, pp. 1279-1298.
13. Minnick, R. C., Bailey, P. T., Sanfort, R. M., Semon, W. L., "Magnetic Bubble Logic," *WESCON Proceedings*, 1972.

The Control Data STAR-100 paging station

by W. C. HOHN and P. D. JONES

Control Data Corporation
St. Paul, Minnesota

INTRODUCTION

The Control Data STAR-100 is a large capacity, high speed, virtual memory[1,2] computer system whose input/output for storage and access is managed by "Stations"[3] separate from the main computer. This modularizes the total computing function into independent, asynchronous tasks which operate in parallel with the central processor. The approach also simplifies the central processor design and provides for fan out to a large number of storage devices and terminals. A station consists of an SCU (station control unit) and an SBU (station buffer unit). An SCU is a mini-computer with small drum and display console existing with power supplies and cooling in its own cabinet. An SBU consists of $64K$ ($K=1024$) bytes of high bandwidth core memory.

A STAR-100 system is shown in Figure 1 with the performance goals noted on each storage station. The M/P station manages maintenance and performance analysis of the STAR-100 mainframe. The media, working and paging stations consist of tapes and disk packs, large disk and drums respectively. Figure 2 shows the layout of a user's virtual storage which covers all the program's active files whenever they are stored. Each user has four keys, which reside in the program's control package and provide four levels of access protection in virtual memory.

In STAR-100 there is one global page table for all users with one entry for each core page. There are two page sizes, normal (4096 bytes) and large (128 times normal). The page table has 16 associative registers at the top and the rest of the table is in core memory (1008 entries in the case of the typical four million byte memory). The translate time in the 16 associative registers is one minor cycle (40 nanoseconds) and the search time is approximately 50 entires per microsecond. When a hit is made that entry jumps to the top of the table, thus most frequently referenced blocks have entries near the top of the table and conversely, the best candidates for removal from memory are at the bottom of the table. This paging mechanism was basically chosen to give the machine a virtual memory capability without degrading performance (100 million results per second). The memory access mechanism is illustrated in Figure 3.

When the virtual address is not found in the page table an access interrupt occurs and control is switched to the monitor. What happens then is flow diagrammed in Figure 4. The paging station contains the overflow pages from main memory and as such is critical in the overall performance of the system.

HARDWARE CONFIGURATION

The STAR-100 paging station, as illustrated in Figure 5, presently consists of two Control Data 865 drums and a page table search mechanism, called a comparator, connected to an SBU; the whole is controlled by an SCU. One half of the 16 page SBU contains the virtual page table and the other half (minus some drum control space) is used as buffer space for the drum/central page transfers. In order to ease the SBU memory conflict situation the SBU memory is hardwired such that it operates as two independent phased (4 bank) memories each with a bandwidth of four 16 bit words every 1.1 microsecond. By this means the comparator has sole access to its memory half and the drums and channels compete in their half, with the drum being top priority.

Figure 6 depicts the functional aspects of the station. The block labelled CONTROL represents the minicomputer within the SCU. All hardware interfaces—drums, comparator, channels—are controlled by routines residing in the SCU. The SBU provides a data freeway for page flow from the drum to central memory. Having the SBU between central memory and the drum reduces channel design complexity in central by not having to interface to critical, real time, rotating devices.

DRUM QUEUE AND DRIVER

Requests for drum transfers are not made to the driver directly but instead to the queue program. This program translates the drum block address (from the comparator search) into a head and sector address. If the resulting sector position is free in the associative queue the request is placed in the queue, otherwise it is placed in retry mode when it is offered to the queue program periodically until accepted. As the number of requests increase, the probability of filling more associative queue slots increases (assuming random drum block addresses) thus raising drum throughput.

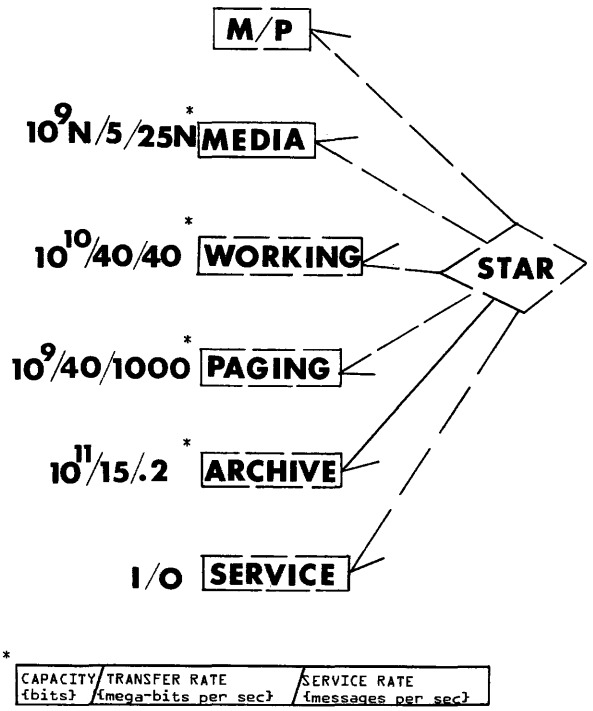


Figure 1—STAR-100 system

Control of the drum is illustrated in Figure 7. The driver program scans the drum angular position and its associated queue slot and if a request exists for that angular position the necessary control information is sent to the drum hardware. The drum control hardware has the ability to stack functions thus making it possible to "stream" contiguous sectors from the drum.

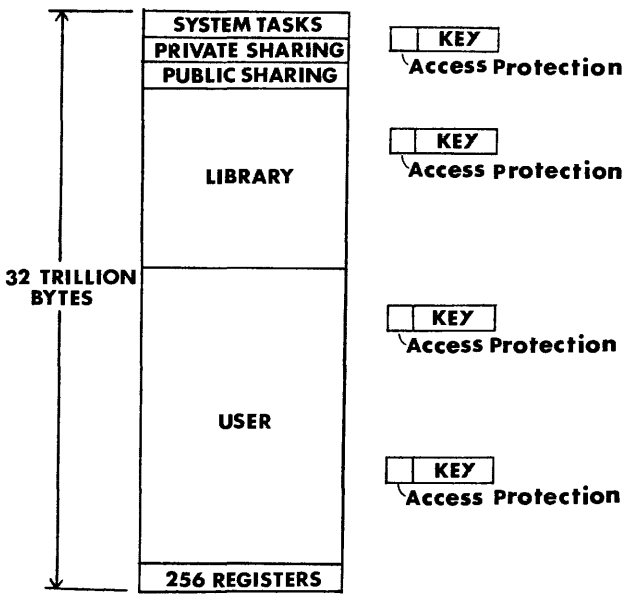


Figure 2—STAR-100 virtual memory

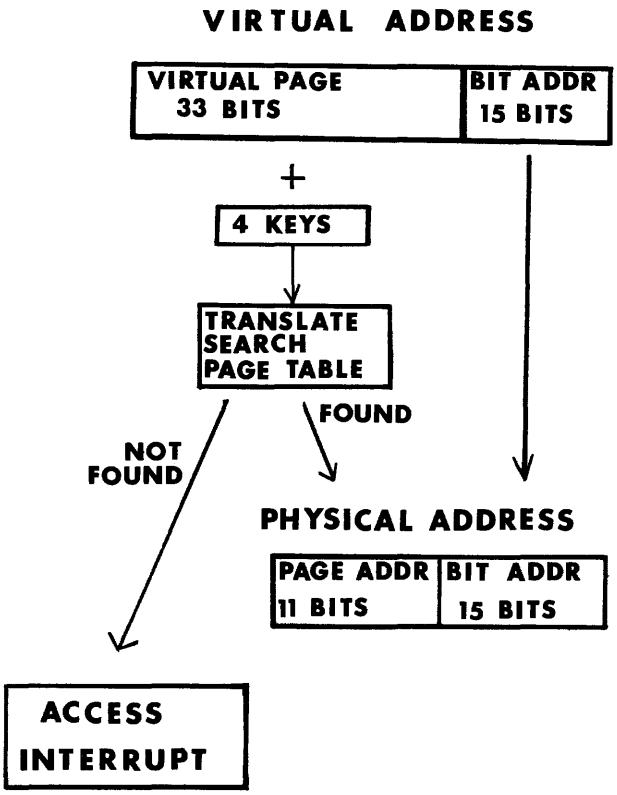


Figure 3—Memory access mechanism

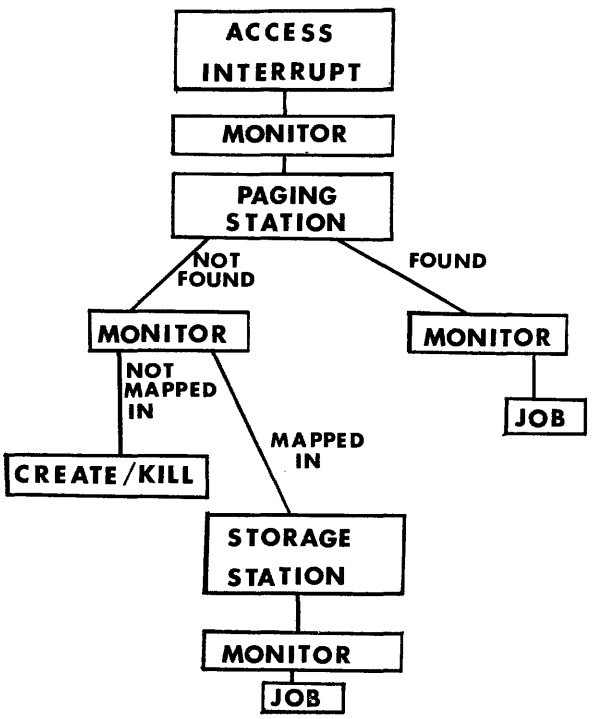


Figure 4—Access interrupt control

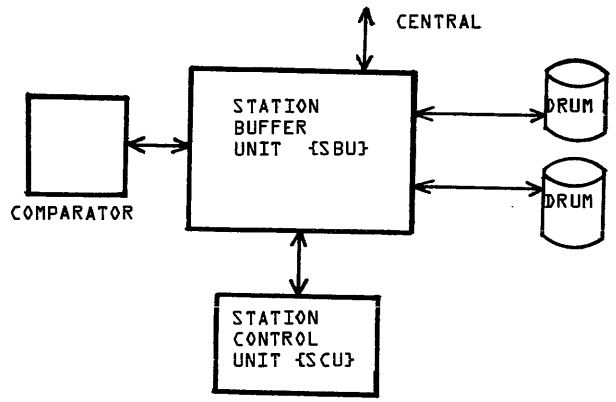
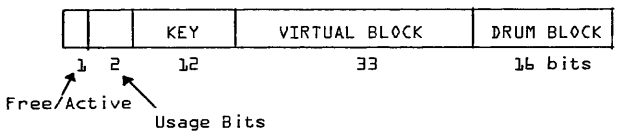


Figure 5—Paging station hardware configuration

COMPARATOR

The virtual page table maps the drum(s) one entry per drum block. Each 64 bit entry contains a unique drum block address and is flagged as either free or attached to a virtual address. The entry format is



The comparator is a hardware unit which compares selected virtual addresses against the page table entries. The hardware "ripples" the table as it searches for a compare. That is, all entries move down as the search passes by them unless a match is found. The entry which matches is placed in the now vacant slot at the top of the list, thereby generating in time a list topped by the most active entry and arranged thereafter in order of descending activity.

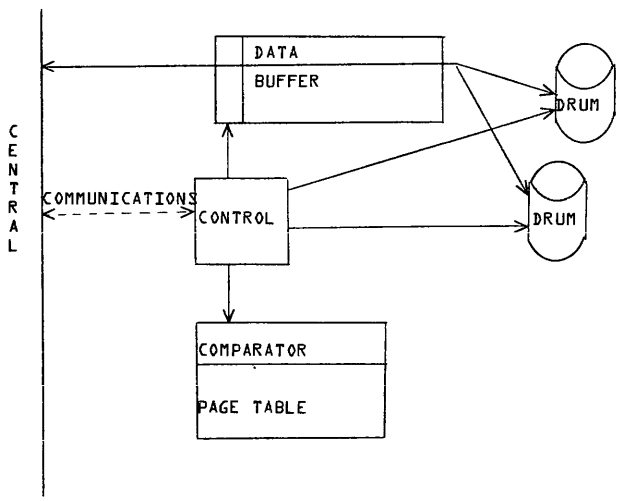


Figure 6—Paging station functional configuration

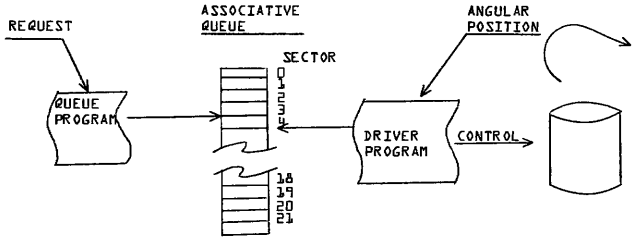


Figure 7—Drum queue and driver

This form of page table maximizes performance because the table is both compressed (all active entries at the top) and ordered by activity, two characteristics which minimize search time.

The table scan rate is one entry every 1.1 microsecond or 910,000 entries per second. Two compares are simultaneously made against a table entry (if only one virtual address request is active the second search is for a null).

The average search time, if both virtual addresses exist in the table, is two thirds of the table length times memory speed.

$$S=2/3L \cdot M$$

If either request is absent from the table then the average search time becomes table length times memory speed.

$$S=L \cdot M$$

In both cases table length, L, is the dynamic length of active table entries, (the table entries attached to a virtual address). If the ripple search were replaced by a straight linear search, the search time would be the reserved table length times memory speed. Table I lists some search rates reflecting the dynamic table length characteristic of the rippled page table.

MESSAGES

The paging station is driven by messages from the central processor. The paging messages are listed in Figure 8. A brief discussion on the message switch mechanism is contained in the next section on performance. Essentially

Table I—Comparator Search Time

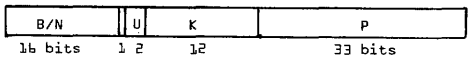
Active Table Length (Entrys)	Search Time (Millisecond)	Search Rate (2/S) (Per Second)
1000	1.1	1820
2000	2.2	910
3000	3.3	606
4000	4.4	455
Page Table Memory Speed — 1.1 Micro-second		

PAGING MESSAGES

Function Code	Function Name	Parameters	Format
200	Read page	B, K, U, P	2A
201	Write page	B, K, U, P	2A
202	Rewrite page	B, K, U, P	2A
203	Delete N pages	N, K, P	2A
204	Delete key {N = number of pages deleted}	N, K	2A
205	Read most active block with given key, then delete. {Page name and usage bits returned}	B, K, U, P	2A
206	Read least-active block with given key, then delete. {Page name and usage bits returned}	B, K, U, P	2A
207	Read and delete page	B, K, U, P	2A
208	Read drum page table	B, N, S, E	2B
209	Swap page	B, K_R, U_R, P_R, K_W, U_W, P_W	2C

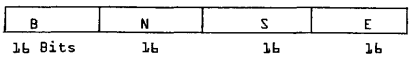
Parameters underlined are returned with the response

FORMAT 2A



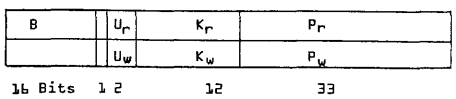
B/N	Block address or number of pages
U	usage bits: These are stored in the drum page table on write and rewrite and returned in this position on read.
U {bit 1}	0/1, unmodified/modified since initial access.
K	Key
P	Virtual page address

FORMAT 2B



B	starting block for drum page table
N	number of blocks to be read
S	word index {64 bit word} to first active entry
E	index {64 bit word} to last +1 active entry

FORMAT 2C



B, U, K and P are the same as in Format 2A. The subscripts R and W denote the pages to be read and written respectively.

MESSAGE HEADER

Response Code	Length & Priority	Used By	
Sender	To Zipcode	From Zipcode	Function Code
16 Bits	16	16	16

Figure 8—Paging messages and formats

the paging station polls central memory for messages, on finding a group of active messages they are read to the SCU where they are processed. The meaning of the messages is self-evident and the following sample message is typical:

0000 0001 XXXX XXXX
XXXX 1300 0100 0200
} --Header

0060 0040 000 4800} --Message Parameters
(Hexadecimal numbers)

The header tells us the message parameter length is one 64 bit word, the message is being sent from station #0100 (central) to station #1300 (paging). The message is to read (function code 200) virtual block #4800, key #80 (#40 shifted one place left) to central memory block #60. The usage bits of this page, that is whether it has been modified or not by the central processor, will be returned with the response.

MESSAGE PROCESSING

The steps involved in processing a typical paging message are shown in Figure 9. All code is reentrant and many messages can be processed simultaneously. The work space for each message is contained in its control package. The number of messages which can be processed simultaneously is a station parameter and is set to ensure a deadlock situation cannot arise, that is, where no message can proceed because of a lack of resources. The comparator search, drum transfer and channel transfer are asynchronous operations which operate in parallel with the SCU processor. For each function code there is one unique message process routine which makes heavy use of subroutines to fulfill its role. The message routines are not permanently resident in core memory but float in and out as required from the mini-drum.

PERFORMANCE PARAMETERS

There are a number of parameters which have a first order effect on station performance and these are now discussed.

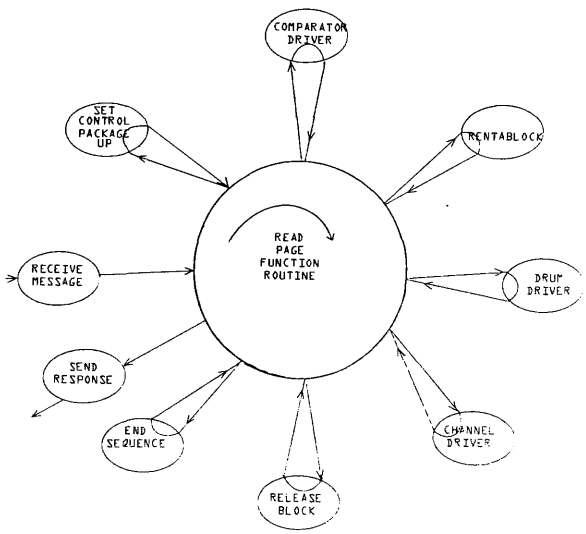


Figure 9—Read page flow diagram

Storage device performance

Large capacity, fast drums (or fixed head disks) are normally used at present for the paging devices. The goal in the STAR system is for a paging device with 10^9 bit capacity, 40 megabit transfer rate and a capability of delivering 1000 random page transfers per second. The initial paging station has twin Control Data 865 drums each with 1408 pages (1 page = 4096 bytes), made up of 64 head groups of 22 pages per 34 millisecond revolution time. The maximum transfer rate is approximately two times 660 pages per second.

Processor speed

The average number of memory cycles to process a message such as read page is 3000. Table 2 lists the

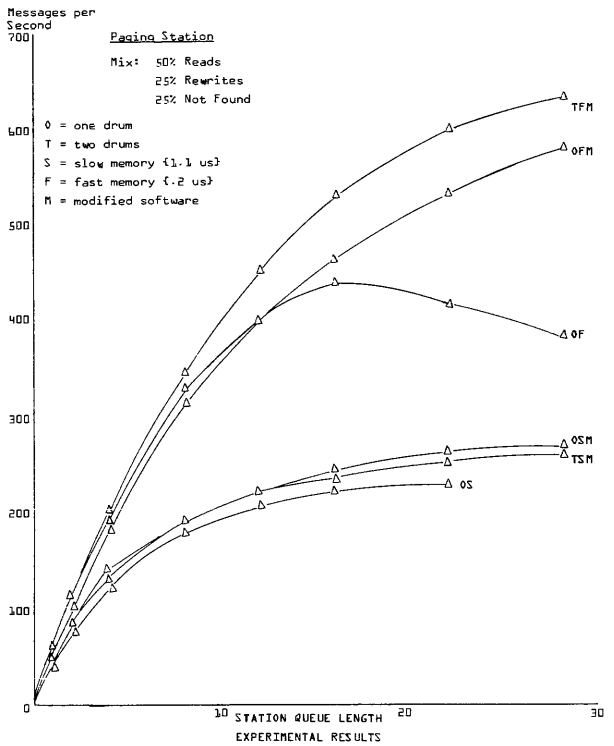


Figure 10—Experimental results

maximum throughput one can expect with varying memory speeds. Figures 10 and 11 show performance curves for two memory speeds, 1.1 and 0.2 microseconds. Experiments were conducted to obtain essentially a curve of station performance versus processor speed; the maximum throughput in Table II was not obtained since when the processor time required to drive a block from drum to central memory became larger than the drum page transfer time (about 1.5 milliseconds) it was not possible to stream pages from the drum. Drum revolutions were lost and there was a sudden drop in performance.

Data buffer size

With the present SBU's there are 7 blocks available to be rented as buffer space for drum transfers. Initially

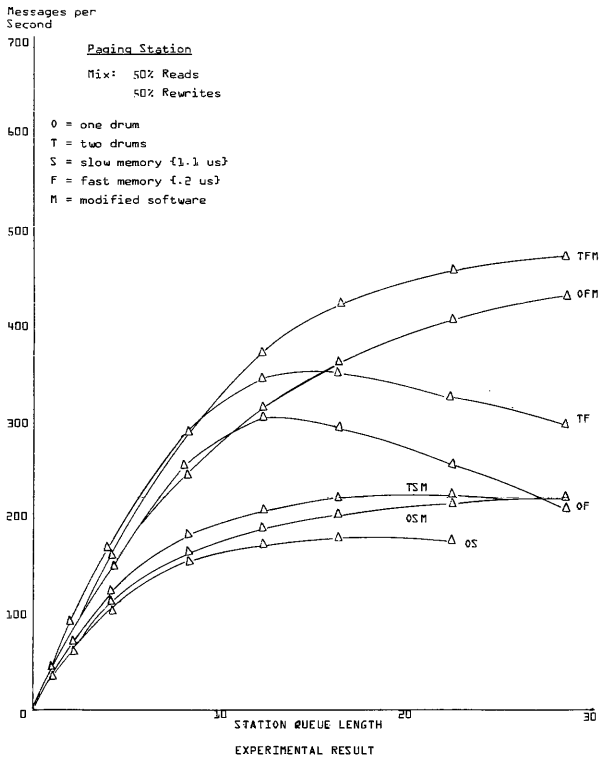


Figure 11—Experimental results

these were rented at the beginning of a message and released at the end (Figure 9). This resulted in a falling off in performance as the queue length became large. The software was modified to enable the drum driver to rent SBU space only for the drum and channel transfer time thus making better use of this resource. The difference in performance is shown in Figures 10 and 11.

SBU bandwidth

As mentioned earlier the paging station SBU operates as two independent phased memories with approximately 58 megabits of bandwidth in each half. Table III illustrates the channel bandwidth available when zero, one and two drums are active. Clearly, just as continuity of flow applies in aero- and hydro-dynamics it also applies

Table II—Message Throughput Versus Cycle Time

Memory Cycle Time	Maximum Throughput
0.1 Micro-seconds	3333 Messages/Second
0.2	1667
0.5	666
1.0	333
1.1	300
2.0	167
Average number cycles per transaction = 3000	

Table III—Channel Transfer Times

DRUMS ACTIVE	CHANNEL TRANSFER RATE
0	40 Mega-bit
1	24
2	10
Maximum Bandwidth = 58 Mega-bit	

in data-dynamics. If channel transfers cannot keep up with drum transfers then drum revolutions are lost and performance curves flatten off.

Message switch

The message handling mechanism between central memory and the SCU vitally affects station performance and already has undergone three distinct developments. Initially, there was a circular queue with pointers to the various messages. This scheme involved too many channel transfers, $3+N$ in fact (read the queue pointers, read the message pointers, read N messages, reset the queue pointers), and was discarded. Apart from the channel transfer time itself there is considerable processor overhead in driving a channel. In the second scheme messages were collected in channel "boats" (Figure 12) and one transfer brought over N messages. The drawback with this scheme was that only one "boat" was brought over at a time, and it was returned only when all messages were complete. This resulted in poor performance and very poor response times. The third scheme was, in effect, a sophisticated boat scheme. Boats could be processed

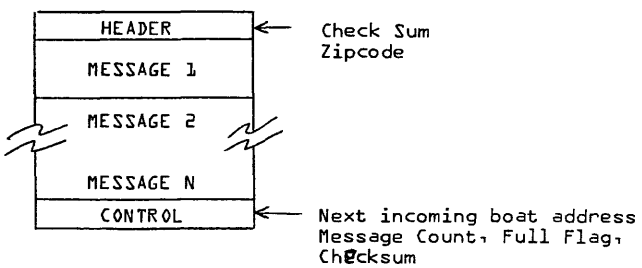


Figure 12—Message boat

before they were full, multiple boats were handled and responses were sent back in the first available boat. Figures 10 and 11 show performance curves obtained using this last scheme with a constant queue of messages maintained in the station. In this case the response time is given by queue length divided by the number of messages serviced per second.

Other factors

SCU memory size for message and control package buffers is another performance limiting parameter, but $16K$ ($K=1024$) bytes appears adequate for the present. The comparator search takes less time than drum latency and is not a limiting factor. Channel bandwidth to central is 40 megabits and is not as much a limit as the SBU bandwidth. The total input/output bandwidth in STAR memory is 3,200 megabit and a 400 megabit data path is available in the future for any new high speed paging devices.

CONCLUSION

As with other STAR stations[3] it is believed a successful attempt has been made to identify part of the general purpose computing function, in this case the paging function, and separate it from the central processor to operate as an independent, asynchronous, self-optimizing unit. Indications are that heavily timeshared, large capacity, high speed virtual memory computers will require paging rates of the order of 1000 pages per second and it appears an upgrade of the paging station with faster drums and twin SBU's will meet this goal.

ACKNOWLEDGMENTS

This work was performed in the Advanced Design Laboratory of Control Data and thanks are due to all members of that laboratory but in particular, J. E. Thornton, G. S. Christensen, J. E. Janes, D. J. Humphrey, J. B. Morgan, R. W. Locke, C. L. Berkey, R. A. Sandness, W. D. Marcus and R. R. Fetter.

REFERENCES

1. Curtis, R. L., "Management of High Speed Memory on the STAR-100 Computer," *IEEE International Computer Conference Digest*, Boston, 1971.
2. Jones, P. D., "Implicit Storage Management in the Control Data STAR-100," *IEEE Compcon '72 Digest*, 1972.
3. Christensen, G. S., Jones, P. D., "The Control Data STAR-100 File Storage Station," *Fall Joint Computer Conference Proceedings*, Vol. 41, 1972.

The linguistic string parser*

by R. GRISHMAN, N. SAGER, C. RAZE, and B. BOOKCHIN

New York University
New York, New York

The linguistic string parser is a system for the syntactic analysis of English scientific text. This system, now in its third version, has been developed over the past 8 years by the Linguistic String Project of New York University. The structure of the system can be traced to an algorithm for natural language parsing described in 1960.[1] This algorithm was designed to overcome certain limitations of the first parsing program for English, which ran on the UNIVAC 1 at the University of Pennsylvania in 1959.[2] The UNIVAC program obtained one "preferred" grammatical reading for each sentence; the parsing program and the grammar were not separate components in the overall system. The 1960 algorithm obtained all valid parses of a sentence; it was syntax-driven by a grammar consisting of elementary linguistic strings and restrictions on the strings (described below). Successive implementations were made in 1965,[3] in 1967,[4] and in 1971.[5] The system contains the largest-coverage grammar of English among implemented natural language parsers.

Implementation of a large grammar by several people over a period of years raises the same problems of complexity and scale which affect large programming projects. The main thrust in the development of the current version of the parser has been to use modern programming techniques, ranging from higher-level languages and subroutine structures to syntax-directed translation and non-deterministic programming, in order to structure and simplify the task of the grammar writer. In this paper we shall briefly review the linguistic basis of the parser and describe the principal features of the current implementation. We shall then consider one particularly thorny problem of computational linguistics, that of conjunctions, and indicate how various features of the parser have simplified our approach to the problem. Readers are referred to an earlier report[6] for descriptions of unusual aspects of the parser incorporated into earlier versions of the system.

Our approach to the recognition of the structure of natural language sentences is based on linguistic string theory. This theory sets forth, in terms of particular syntactic categories (noun, tensed verb, etc.) a set of elementary strings and rules for combining the elementary strings to form sentence strings.

The simplest sentences consist of just one elementary string, called a *center string*. Examples of center strings are *noun tensed-verb*, such as "Tapes stretch." and *noun tensed-verb noun*, such as "Users cause problems." Any sentence string may be made into a more complicated sentence string by inserting an *adjunct string* to the left or right of an element of some elementary string of the sentence. For example, "Programmers at our installation write useless code." is built up by adjoining "at our installation" to the right of "programmers" and "useless" to the left of "code" in the center string "programmers write code." Sentences may also be augmented by the insertion of a *conjunct string*, such as "and debug" in "Programmers at our installation write and debug useless code." Finally, string theory allows an element of a string to be replaced by a *replacement string*. One example of this is the replacement of *noun* by *what noun tensed-verb* to form the sentence "What linguists do is baffling."

The status of string analysis in linguistic theory, its empirical basis and its relation to constituent analysis on the one hand and transformational analysis on the other, have been discussed by Harris.[7] More recently, Joshi and his coworkers have developed a formal system of grammars, called string adjunct grammars, which show formally the relation between the linguistic string structure and the transformational structure of sentences.[8] The string parser adds to linguistic string theory a computational form for the basic relations of string grammar. In terms of these relations the arguments of grammatical constraints (i.e., mutually constrained sentence words) can always be identified in the sentence regardless of the distance or the complexity of the relation which the words have to each other in that sentence.[9]

Each word of the language is assigned one or more word categories on the basis of its grammatical properties. For example, "stretches" would be classed as a *tensed verb* and a *noun*, while "tape" would be assigned the three categories *tensed verb*, *untensed verb*, and *noun*. Every sequence of words is thereby associated with one or more sequences of word categories. Linguistic string theory claims that each sentence of the language has at least one sequence of word categories which is a sentence string,

* The work reported here was supported in part by research grants from the National Science Foundation: GN559 and GN659 in the Office of Science Information Services, and GS2462 and GS27925 in the Division of Social Sciences.

i.e., which can be built up from a center string by adjunction, conjunction, and replacement.

However, not every combination of words drawn from the appropriate categories and inserted into a sentence string forms a valid sentence. Sometimes only words with related grammatical properties are acceptable in the same string, or in adjoined strings. For example, one of the sequences of word categories associated with "Tape stretch." is *noun tensed-verb*, which is a sentence string; this sentence is ungrammatical, however, because a singular *noun* has been combined with a plural *tensed-verb*. To record these properties, we add the subcategory (or attribute) singular to the category *noun* in the definition of "tape" and the subcategory plural to the category *tensed-verb* in the definition of "stretch." We then incorporate into the grammar a restriction on the center string *noun tensed-verb*, to check for number agreement between noun and verb.

The number of such restrictions required for a grammar of English is quite large. (The current grammar has about 250 restrictions.) However, the structural relationship between the elements being compared by a restriction is almost always one of a few standard types. Either the restriction operates between two elements of an elementary string, or between an element of an elementary string and an element of an adjunct string adjoining the first string or a replacement string inserted into the first string, or (less often) between elements of two adjunct strings adjoined to the same elementary string. This property is an important benefit of the use of linguistic string analysis; it simplifies the design of the restrictions and plays an important role in the organization of the grammar, as will be described later.

IMPLEMENTATION

As the preceding discussion indicates, the string grammar has three components: (1) a set of elementary strings together with rules for combining them to form sentence strings, (2) a set of restrictions on those strings, and (3) a word dictionary, listing the categories and subcategories of each word. Component 1. defines a context-free language and, for purposes of parsing, we have chosen to rewrite it as a BNF grammar.

The approximately 200 BNF definitions in our grammar can be divided into three groups. About 100 of these are single-option *string* definitions; each of these corresponds to one (or occasionally several) strings. For example,

ASSERTION: : = <SA><SUBJECT><SA><VERB>
<SA><OBJECT><RV><SA>

contains the required elements SUBJECT, VERB, and OBJECT (corresponding to the three elements in such center strings as *noun tensed-verb noun* and *noun tensed-verb adjective*) and the optional elements SA, indicating where an adjunct of the entire sentence may occur, and RV, for right adjuncts of the verb appearing after the object. SA and RV are two of the approximately 20 adjunct set definitions; these definitions group sets of strings which may adjoin particular elements. The remaining definitions, including those for SUBJECT, VERB, and OBJECT, are collections of positional variants; these define the possible values of the elements of string definitions.

Once component 1. has been rewritten in this way, it is possible to use any context-free parser as the core of the analysis algorithm. We have employed a top-down serial parser with automatic backup which builds a parse tree of a sentence being analyzed and, if the sentence is ambiguous, generates the different parse trees sequentially.

The parse tree for a very simple sentence is shown in Figure 1. A few things are worth noting about this parse tree. Most striking is the unusual appearance of the parse tree, as if it had grown up under a steady west wind. We have adopted the convention of connecting the first daughter node to its parent by a vertical line, and connecting the other daughter nodes to the first by a horizontal line. This is really the natural convention for a string grammar, since it emphasizes the connection between the elements of a string definition. More interesting is the regular appearance of "LXR" definitions: a <LNR> below the subject, a <LTVR> below the verb, and a <LAR> below the object. Each LXR has three elements: one for left adjuncts, one for right adjuncts, and one in the middle for the *core* word. The *core* of an element of a definition is the word category corresponding to this element in the associated elementary string in the sentence; e.g. the core of SUBJECT (and of LXR) in Figure 1 is the noun "trees"; it is the one terminal node below the element in question which is not an adjunct. In some cases the core of an element is itself a string. LXR definitions and linguistic string definitions play a distinguished role in conjoining, as will be described later.

Each restriction in the grammar is translated into a sequence of operations to move about the parse tree and test various properties, including the subcategories of words attached to the parse tree. When a portion of the parse tree containing some restrictions has been completed, the parser invokes a "restriction interpreter" to execute those restrictions. If the restriction interpreter returns a success signal, the parser proceeds as if nothing

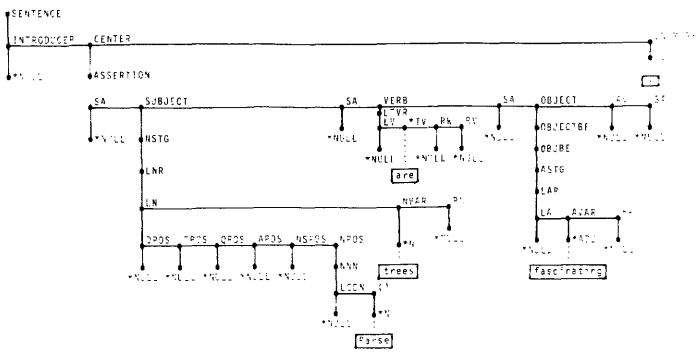


Figure 1—Parse tree for "Parse trees are fascinating"

had happened; if it returns a failure signal, the parser backs up and tries to find some alternate analysis.

The first version of the system was written in the list-processing language IPL-V for the IBM 7094. Because IPL-V was an interpretive system rather than a compiler, this implementation proved to be too slow for parsing a large corpus of sentences, and it was replaced by a system written in assembly language for the IBM 7094 (FAP). The speed of these systems was considerably enhanced by a mechanism which saved the subtrees below certain specified nodes the first time they were constructed, so that they would not have to be repeatedly built up. If restrictions on the saved subtrees had accessed nodes outside the subtree, these restrictions were re-executed when the subtree was re-used. With this saving mechanism the second version of the parser was able to obtain a first parse in a few seconds and all parses in under a minute for most sentences.

In both systems, the entire grammar (context-free component, restrictions, and word dictionary) was encoded in a uniform list-structure format. This format was easy to convert to internal list structure, but, particularly for encoding the restrictions, it left something to be desired with regard to perspicuity and brevity of expression. As the grammar was gradually refined and expanded, and especially when the restrictions were modified to handle conjunctions, these deficiencies increasingly burdened the grammar writer.

Therefore, when work was begun on a new version for the CDC 6600 in 1969 we set as our goal, in addition to the creation of a relatively machine-independent FORTRAN implementation, the development of a higher-level language suitable for writing restrictions. Because we realized that the language specifications would evolve gradually as new features were added to the system, we decided to use a syntax-directed compiler in translating the language into the internal list structure required by the program. This decision actually simplified the design of the overall system, since several modules, including the basic parser, could be used in both the compiler and the analyzer of English.

The restriction language we have developed looks like a subset of English but has a fairly simple syntax. The basic statement form is subject-predicate, for example

THE OBJECT IS NOT EMPTY.

This hypothetical restriction might be "housed" in ASSERTION (whose definition is given above); that is, it would be executed when the parse tree below ASSERTION was completed, and it would begin its travels through the parse tree at the node ASSERTION. The restriction looks for an element of ASSERTION named OBJECT (that is, it searches the level in the tree below ASSERTION for the node OBJECT). When it finds it, it verifies that the node is not empty, i.e., subsumes at least one word of the sentence.

Nodes which are not elements of the current string (the string in which the restriction is housed) can be referenced by using one or more of a set of "tree-climbing" routines. These routines correspond to the basic relations of the string grammar, such as CORE, LEFT ADJUNCT, RIGHT ADJUNCT, HOST (which goes from an adjunct to the element it adjoins), and COELEMENT (which goes from one element of a string definition to another). For example, a restriction sentence starting at ASSERTION,

THE CORE OF THE SUBJECT IS NHUMAN.

would begin by looking for the element SUBJECT. It would then invoke the routine CORE to descend to the core of the SUBJECT and test whether the sentence word corresponding to that node has the attribute NHUMAN.*

Because all grammatical restrictions test elements bearing one of only a few structural relationships to the current string (as described above), it is possible to formulate all the restrictions in terms of a small set of about 25 such "locating routines." The routines are coded in terms of the basic tree operations, such as going up, down, left, and right in the parse tree, and other operations, such as testing the name of the node. The basic tree operations are not used directly by the restrictions. The routines correspond roughly to the low-level assembly language routines in a large programming system. The routines not only simplify the task of writing the restrictions, but also permit fundamental modifications to be made to the grammar by changing the routines rather than having to individually change each of the restrictions. One example of this, the modification for conjunctions, will be described later.

The restriction language contains the full range of logical connections required for combining basic subject-predicate sentences into larger restrictions: BOTH___ AND___, EITHER___OR___, NOT___, IF___ THEN___, etc. For example, a very simple restriction for subject-verb agreement in number is

IF THE CORE OF THE VERB IS SINGULAR
THEN THE CORE OF THE SUBJECT IS NOT PLURAL.

Registers (i.e., variables) of the form *Xn*, *n* an integer, may be used to temporarily save points in the tree. For example, in

BOTH THE CORE X1 OF LNR IS NAME OR NSYMBOL
AND IN THE LEFT-ADJUNCT OF X1, NPOS IS NOT EMPTY.

*X*1 is used to save the point in the tree corresponding to "CORE OF LNR" so that it need not be recomputed for the second half of the restriction.

* Informally speaking, the noun subclass NHUMAN refers to human nouns. Grammatically it is any word which can be adjoined by a relative clause beginning with "who."

The syntax-directed compiler has the job of translating the restriction language statements into lists composed of the basic operations recognized by the restriction interpreter. For instance, the restriction sentence given above, "THE CORE OF THE SUBJECT IS NHUMAN." would be compiled into*

(EXECUTE [(STARTAT [(SUBJECT)])],
EXECUTE [(CORE)],
ATTRB [(NHUMAN)])

The EXECUTE operator is used to invoke routines. The first operation, a call on the routine STARTAT with argument SUBJECT, searches the level below ASSERTION for the node SUBJECT. This is followed by a call on the routine CORE and then by the operation ATTRB, which checks if the word associated with the node reached has the attribute NHUMAN.

The restriction language syntax (RLS) which guides the compilation process is a set of BNF productions into which have been incorporated directives for generating the list structures. Each directive is a list of code (list structure) generators which are to be executed when the production is used in the analysis of the restriction language statement.

Our first version of the RLS followed a format described by Cocke and Schwartz.[10] The generators to be invoked were simply listed between the elements of the BNF definitions; the parser called on these generators during the parsing of the statement. This arrangement is described in detail in Reference 5. It is quite efficient but has several drawbacks. First, the parser cannot back up (since it cannot undo the action of a generator) so the RLS must be written to allow analysis in a single left-to-right pass. Second, if the list structure to be generated is at all complicated, the task of creating the generator sequences is error-prone and the resulting BNF productions do not clearly indicate what code will be generated.

We have circumvented the first problem by first parsing the statement and then scanning the parse tree and executing the generators. We have attacked the second problem by allowing the user to write a list structure expression as part of each production and having the system compile this to the appropriate sequence of generator calls. In our new version of the compiler, after the restriction statement is parsed its parse tree is scanned from the bottom up. At each node, the sequence of generators in the corresponding production of the compiled RLS is executed. These generators should re-create the list structure which the user wrote as part of the (source) RLS production. This list structure is then assigned as the "value" of that node; generators on the node one level up may use this value in building a larger list structure. In this way, the small list structures at the bottom of the tree are gradually combined into larger structures. The structure assigned to the root node of the tree is the generated code for the statement; this code is written out as part of the compiled grammar of English. This procedure is similar to the compiler described by Ingermann[11] and the formalism of Lewis and Stearns.[12]

A simple example of an RLS statement is

<REGST>: : =<*REG>
→(STORE[<REG>]) | <*NULL>.

This says that the symbol REGST may be either a token of lexical type REG (a register, Xn) or the null string. If REGST matches a register, the code which will be generated and assigned to REGST is the operation STORE with an argument equal to the name of the register matched. If the null option is used in the parse tree, no code is generated.

As a more complicated example we consider two options of NSUBJ, the subject of a restriction statement:

<NSUBJ>: : =<*NODE>
→(EXECUTE[(STARTAT[(<NODE>)])]) |
CORE<REGST>OF<NSUBJ>
→<NSUBJ>:(EXECUTE[(CORE)]):<REGST> | · · · ·.

The first type of subject is simply the name of a node of the tree; the generated code is a call on the routine STARTAT with that node as argument. The second type is CORE OF followed by some other valid subject, with a register name following the word CORE if the position of the core is to be saved. The generated code here is the concatenation of three list structures (":" indicates concatenation). The first of these is the code generated to locate the NSUBJ following CORE OF; the second is a call on the routine CORE; the third is the code generated to save the present position in a register (this last structure will be empty if no register is present).

To illustrate the combined effect of these RLS statements, we present, in somewhat simplified form, the parse tree for our restriction sentence, "THE CORE OF THE SUBJECT IS NHUMAN." in Figure 2. To the left of each node appears its name; to the right, the list structure assigned to that node by the generating process.

To compile the RLS into lists of syntactic categories and generator calls in the proper internal format, we require one further application of the syntax-directed compiler. This makes for a total of three uses of the parser, two as a compiler and one as an analyzer of English. The entire process which results finally in parses of Eng-

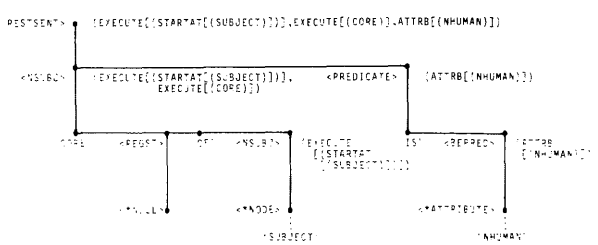


Figure 2 Parse tree for CORE OF SUBJECT IS NHUMAN

* In our list notation, square brackets enclose arguments of operators and routines.

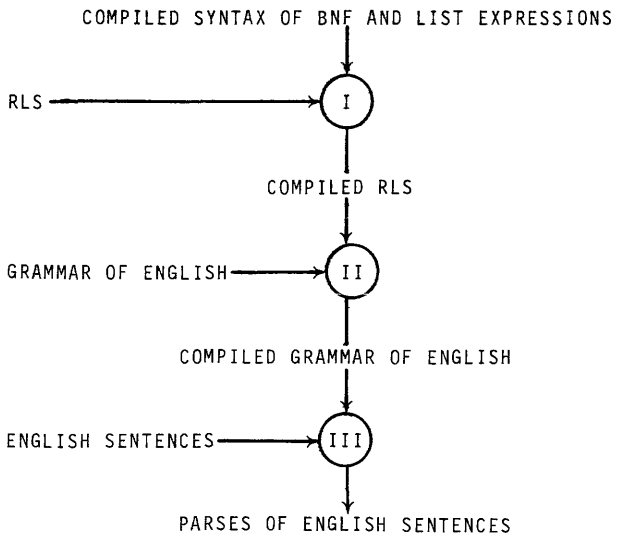


Figure 3—The three uses of the parser

lish sentences is diagrammed in Figure 3. To avoid an infinite regression, the first grammar, the compiled syntax of BNF and list expressions, must be produced by hand; fortunately, it is quite short.

CONJUNCTIONS

The problem in computing conjunctions is threefold. (1) To provide for the richness of conjunctional constructions in full-bodied English texts. This is best handled by locally conjoining the syntactic categories as they appear in the sentence (*noun* and *noun, tensed-verb* and *tensed-verb*, etc.) (2) To recover hidden or "zeroed" material from the sentence so that semantic and selectional constraints can be applied to all relevant parts of the sentence. This implies an expansion of the sentence. For example, the sentence

(a) They program and debug systems

has the expansion

(a′) They program (systems) and (they) debug systems.

(3) To meet our first two objectives without greatly complicating and enlarging the grammar. This necessitated an addition to the parser and modifications to the restriction language routines.

According to linguistic string theory an element or sequence of elements of a string may be conjoined in the following manner: If the successive syntactic elements of the string are $E_1E_2 \cdots E_i \cdots E_n$ then conjunction may occur after E_i, and the conjoined sequence consists of a conjunction followed by E_i or $E_{i-1}E_i$ or $\cdots$ or $E_1E_2 \cdots E_i$. In sentence (a), above, the syntactic categories are *pronoun tensed-verb conjunction tensed-verb noun.* In the sentence

(b) They program and we debug systems.

the elements are *pronoun tensed-verb conjunction pronoun tensed-verb noun.* In addition, if E_i is a positional variant (such as OBJECT) representing a set of alternative values for a particular position in the string, one value of E_i may be conjoined to another. In the sentence

(c) I don't like him and what he stands for.

two different values of object are conjoined, namely *pronoun* and the *N*-replacement string *what* ASSERTION.*

To include the conjunctional definitions explicitly in the BNF grammar would cause the grammar to increase enormously. Instead conjunctional strings are inserted dynamically by a special process mechanism when a conjunction is encountered in parsing the sentence. This is equivalent in effect to having all possible conjunctional definitions in the grammar before parsing begins.

The special process mechanism interrupts the parser when an element of a definition has been completely satisfied and the current sentence word to be analyzed is a conjunction. An interrupt results in the insertion in the tree of a special process node. For each special word there is a corresponding definition in the grammar. This definition consists of the conjunctional word and a string which defines the general feature of the type of special process to be computed (conjunctional, comparative). For example, <SP-AND>=AND<Q-CONJ>. The general conjunctional string Q-CONJ contains a restriction which generates a definition for Q-CONJ depending on where the interrupt takes place. If it occurs after E_i the following definition will be generated for Q-CONJ: $<E_i>|<E_{i-1}><E_i>|\cdots\cdots|<E_1><E_2>\cdots\cdots<E_i>$. Consider the sentence

(d) He and she like computers.

A tree of the analysis of the sentence is shown in Figure 4. After *he* has been matched as a pronoun an interrupt occurs and SP-AND is inserted after NVAR in LNR.

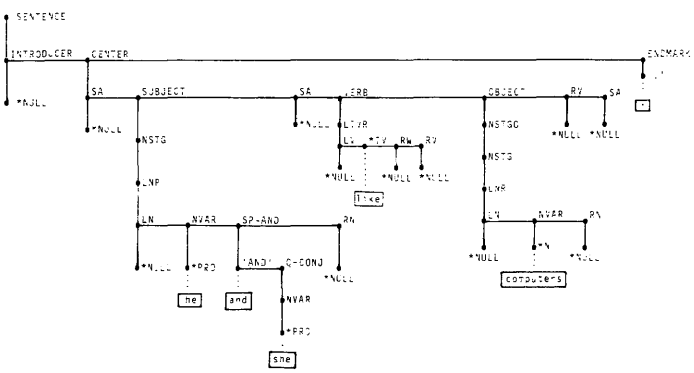


Figure 4—Parse tree for "He and she like computers"

* The ASSERTION string after words like "what," "who," etc., is expected to occur with one noun position empty, here the object of "stands for."

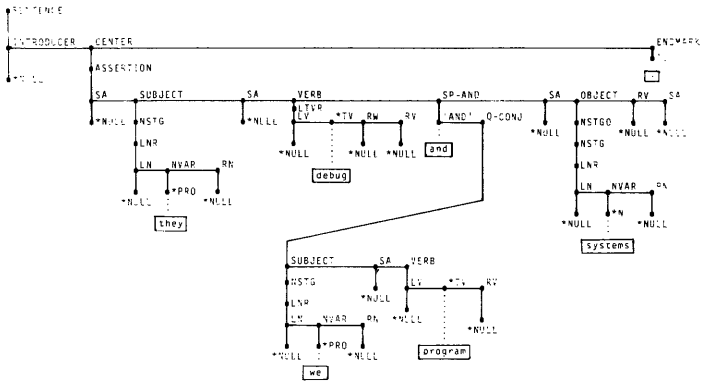


Figure 5—Parse tree for "They debug and we program systems."

Local conjoining (i.e. conjoining within a single string-element position such as SUBJECT) will fail however for

(e) They debug and we program systems.

The tree for this sentence is shown in Figure 5. Here, the conjunctional string covers two string-element positions (SUBJECT, VERB) and must be conjoined at a higher level in the analysis tree.

When local conjoining fails, the special process node is detached and normal parsing continues. But an interrupt will occur again if the conjunctional word has not alternatively been analyzed as a non-special word (such as "but" in "I will be but a minute."). A second interrupt occurs when the next element satisfied is NULL or the parse moves to a higher level of the analysis. For example, in sentence (e), "and" is still current when the VERB position of ASSERTION is completed. Therefore an interrupt will occur again at this higher level and the proper conjunctional string will be inserted. After the special process node is completed normal parsing resumes.

If the sentence contains a special process scope marker such as *both* as part of a *both . . . and* sequence an additional check is made in the restriction that defines Q-CONJ. It removes any option that contains an element whose corresponding element in the pre-conjunctional string precedes the scope marker. Thus we accept "He both debugs and programs systems," but not "He both debugs and we program systems."

Although the special process mechanism can interrupt the parser at many different points in the analysis tree, the regularity of conjunctional strings with respect to the rest of the string grammar enables us to confine the conjunctional interrupt process to just two types of definitions: to host-adjunct sequences (i.e., after each element of LXR type definitions) and to linguistic strings (i.e., after each required element in definitions of the type "linguistic string"). In this way numerous redundancies caused by conjoining at intermediate nodes are eliminated. In addition, confining the conjoining to these two types of definitions simplifies the modification to the restriction language routines which is needed for conjunctions.

Since restrictions are the key to obtaining correct parses, it is essential that they operate on conjunctional strings as well as on the ordinary strings of the grammar. However, conjunctional strings very often contain only a portion of the sequence to which a restriction applies, necessitating corrections to each restriction of the grammar to take account of conjunctional strings, or else a general solution to the problem of truncated conjunctional strings. The general solution employed by the current string parser is to modify, not each restriction, but the small number of basic routines used by all restrictions.*

Our solution is based on the following: If a restriction requires a special condition for element E_i of a string S or of a host-adjunct sequence, then that condition should also be true for E_i in the segment conjoined to S. Similarly, if a restriction requires a wellformedness condition between two elements E_i and E_j of S (or between host and adjunct), then the same condition should be true for E_i and E_j in the conjoined segment. If one of the elements E_i is not present in the conjoined segment, the restriction applies between E_j in the conjoined segment and E_i in S.

Certain of the basic restriction routines were modified in accord with the above, by introducing a "stack" operation. If in the execution of a restriction a routine is called to locate an element and this element has conjoined elements then the stack operation is executed for each of the conjoined elements. If the restriction is successful, it is resumed at the point in the restriction immediately following the routine which executed the stack operation. But when the restriction is resumed, it is looking not at the original element located by the routine but rather at its conjoined element. The restriction is successful only if it succeeds for all the conjoined elements.

Consider the operation of the selectional restriction WSEL2. This restriction states: If the core of the subject is a noun or pronoun and if the core C of the coelement verb (of the subject) has a subclassification NOTNSUBJ which consists of a list of forbidden noun or pronoun subclasses for the given verb, then the sentence word corresponding to C should not have any of those subclassifications. The verb *occurs* forbids as its subject a noun having a human subclassification. Thus the sequence *programmers occur* is not well formed whereas *problems occur* is. For WSEL2 to test the core of the subject position the routine CORE is called. In the sentence *Problems and difficulties occurred later* the CORE routine will stack *difficulties* so that the wellformedness of both *problems occurred* and *difficulties occurred* will be tested. WSEL2 will therefore fail for the sequence *Problems and programmers occurred later.* In the sentence *Difficulties and problems occurred but later disappeared* WSEL2 will be executed four times (the core value of the verb position is conjoined also) testing the wellformedness of the sequences *difficulties occurred, problems occurred, diffi-*

* In the previous implementations the restrictions were individually modified so as to operate correctly under conjunctions. This demanding task was carried out by Morris Salkoff.

culties disappeared, problems disappeared. In this way stacking has the same effect as expanding the sentence. There are some restrictions, however, such as the ones testing number agreement, which cannot use stacking and which must be changed to test specifically for conjoining. Therefore each routine that stacks has a nonstacking counterpart which these restrictions use.

Stacking is sufficient for the bulk of conjunctional occurrences, those whose form is described above. However, there are other types. In

(f) He debugged the parser and she the grammar.

there is zeroing in the middle position (i.e., in the verb position between the subject and object positions) of the assertion beginning with "she." In

(g) He heard of and liked the new system.

the noun element following *of* in the prepositional string has been zeroed. It is possible and highly desirable for the zeroed element to be filled in during the analysis. The program has a mechanism for these cases which is activated by a restriction routine. It temporarily delays the execution of all those restrictions that may refer to the zeroed element. Further on in the analysis the zeroed slot is filled in by linking it to the subtree corresponding to another part of the sentence. For example in sentence (g), above, the zeroed slot for the noun-sequence after the preposition "of" will be linked to the subtree corresponding to *the new system* so that in effect the zeroed information has been restored. The sentence thus becomes

(g′) He heard of (the new system) and he liked the new system.

After linking occurs, the delayed restrictions are executed. From that point on any restriction that refers to the zeroed element will automatically be switched to its linked subtree. While backing up, a restriction may obtain an alternative link to another subtree. In this way all analyses via different subtree links are arrived at. In "We chose and started to implement another algorithm," one analysis is "We chose (another algorithm) and started to implement another algorithm." Another analysis is "We chose (to implement another algorithm) and started to implement another algorithm."

THE USE OF THE STRING PARSER

A practical goal for the parser is to aid in the processing of scientific information. It is conceivable, for example, that answers to highly specific informational queries could be extracted from large stores of scientific literature with the aid of natural language processing techniques. Such an application requires that there be a close correlation between the computer outputs for a text and the information carried by the text.

An examination of the string output parses for texts in various fields of science[13] shows that the decomposition of a sentence into its component elementary strings constitutes a first breakdown of the sentence into its elementary informational components. The correlation would be much improved if we could (1) reduce the redundancy of grammatical forms (redundant from the point of view of the information carried by the sentence), i.e., arrive at a single grammatical form for sentences or sentence parts carrying the same information; (2) reduce ambiguity, i.e., arrive at the author's intended reading from among the syntactically valid analyses produced by the parser.

Fortunately, methods are available for attacking both problems. Transformational refinement of the grammar leads precisely to determining a single underlying sentence in semantically related forms, such as the active and passive, and numerous nominalization strings, e.g. "We should reduce ambiguity," "ambiguity should be reduced," "the reducing of ambiguity," "the reduction of ambiguity," etc.

With regard to syntactic ambiguity, the largest number of cases have their source in different possible groupings of the same string components of the sentence, the decisive criterion being which of the resulting word-associations is the correct one for the given area of discourse. For example, in "changes in cells produced by digitalis," only one of the possible groupings (that in which digitalis produces changes, not cells) is correct within a pharmacology text dealing with cellular effects of digitalis. Recently it has been shown that it is possible to incorporate into the grammar which is used to analyze texts in a given science subfield additional grammatical constraints governing the wellformedness of certain word combinations when they are used within that subfield.[14] These constraints have the force of grammatical rules for discourse within the subfield (not for English as a whole), and have a very strong semantic effect in further informationally structuring the language material in the subfield, and in pointing to the correct word associations in syntactically ambiguous sentences.

REFERENCES

1. Sager, N., "Procedure for Left-to-Right Recognition of Sentence Structure," *Transformations and Discourse Analysis Papers*, No. 27, Department of Linguistics, University of Pennsylvania, 1960.
2. Harris, Z. S., et al., *Transformations and Discourse Analysis Papers*, Nos. 15-19, University of Pennsylvania, Department of Linguistics, 1959.
3. Sager, N., Morris, J., Salkoff, M., *First Report on the String Analysis Programs*, Department of Linguistics, University of Pennsylvania, 1965. Expanded and reissued as *String Program Reports* (henceforth SPR), No. 1, Linguistic String Project, New York University, 1966.
4. Raze, C., "The FAP Program for String Decomposition of Sentences," *SPR*, No. 2, 1967.
5. Grishman, R., "Implementation of the String Parser of English," Presented at the *Eighth Courant Computer Science Symposium*, December 20, 1971. To be published in *Natural Language Processing*, Rustin, R., ed., Algorithmics Press, New York (in press).

6. Sager, N., "Syntactic Analysis of Natural Language," *Advances in Computers*, No. 8, Alt, F., and Rubinoff, M., (eds), Academic Press, New York, 1967.
7. Harris, Z. S., *String Analysis of Sentence Structure*, Mouton and Co., The Hague, 1962.
8. Joshi, A. K., Kosaraju, S. R., Yamada, H., "String Adjunct Grammars," Parts I and II, *Information and Control*, No. 21, September 2, 1972, pp. 93-116 and No. 3, October 3, 1972, pp. 235-260.
9. Sager, N., "A Two-Stage BNF Specification of Natural Language," *Information Sciences*, (in press).
10. Cocke, J., Schwartz, J., *Programming Languages and Their Compilers*, Courant Institute of Mathematical Science, New York University, 1969.
11. Ingerman, P. Z., *A Syntax-Oriented Translator*, Academic Press, New York, 1966.
12. Lewis, P. M., II, Stearns, R. E., "Syntax-Directed Transduction" *JACM*, No. 15, p. 465, 1968.
13. Bookchin, B., "Computer Outputs for Sentence Decomposition of Scientific Texts," *SPR*, No. 3, 1968.
14. Sager, N., "Syntactic Formatting of Science Information," *Proceedings of the 1972 Fall Joint Computer Conference*, pp. 791-800.

A multi-processing approach to natural language

by RONALD M. KAPLAN

Harvard University
Cambridge, Massachusetts

INTRODUCTION

Natural languages such as English are exceedingly complicated media for the communication of information, attitudes, beliefs, and feelings. Computer systems that attempt to process natural languages in more than the most trivial ways are correspondingly complex. Not only must they be capable of dealing with elaborate descriptions of how the language is put together (in the form of large dictionaries, grammars, sets of inference strategies, etc.), but they must also be able to coordinate the activities and interactions of the many different components that use these descriptions. For example, speech understanding systems of the sort that are currently being developed under ARPA sponsorship must have procedures for the reception of speech input, phonological segmentation and word recognition, dictionary consultation, and morphological, syntactic, semantic, and pragmatic analyses. The problems of coordination and control are reduced only slightly in less ambitious projects such as question answering, automatic programming, content analysis, and information retrieval. Of course, large-scale software systems in other domains might rival natural language programs in terms of the number and complexity of individual components. The central theme of the present paper, however, is that natural language control problems have a fundamentally different character from those of most other systems and require a somewhat unusual solution: the many natural language procedures should be conceptualized and implemented as a collection of *asynchronous communicating parallel processes.*

A common technique for organizing a large system is to arrange the flow of control in a hierarchy that follows the intuitive "outside-in" flow of data. In language processing, this design calls for a serial invocation of procedures, with the input routines succeeded by segmentation, word recognition, morphological analysis, and dictionary look-up. The output of these procedures would be passed to the syntactic analyzer (parser), which would build syntactic structures and send them on to the semantic and inference-making routines. The difficulty with this straightforward arrangement is that each procedure in the chain must operate in a locally uncertain environment. For example, there might not be enough information in the incoming signal to determine precisely what the string of words is (is it 'nitrite' or 'night rate'?), or dictionary consultation might produce several senses for a single, clearly identified word ('saw' as a noun, a form of the verb 'saw', or a form of the verb 'see'). Later on, the syntactic analyzer might discover several parses, or the semantic procedures might find multiple interpretations. Each level of analysis might be prepared to handle many independent possibilities, some of which are passed from earlier modules, and some of which it generates and passes on. Except for certain unusually well-behaved inputs, this linear control strategy will lead to exponential increases in the amount of computation, and the system will be hopelessly swamped in the combinatorics.

Any realistic system must have ways of selectively ignoring certain implicit possibilities, thereby reducing the effective size of the computation space. After all, most sentences spoken in everyday conversation are not ambiguous, given their total linguistic and pragmatic context. This means that if we search the computation space to exhaustion, we will find that most of the possibilities for most of the inputs lead to dead-ends, and there is only one globally consistent interpretation. The problem is to minimize the number of dead-ends encountered in arriving at this interpretation and to stop computing as soon as it is achieved. Thus if the segmentation and word recognition routines first come up with the 'nitrate' possibility, we want to feed this immediately to the syntactic and semantic routines. If a meaningful interpretation results, the extra work necessary to discover the 'night rate' sequence can be avoided, as can the additional effort that all "later" modules would have to devote to it. But if 'nitrate' is incompatible with the surrounding context, the segmentation routines must be able to resume where they left off and produce 'night rate'. In general, the various modules must be capable of communicating results and intermingling their operations in a "heterarchical"[4] fashion according to some *heuristic* strategies.

The simple hierarchical model must be abandoned, but this does not mean that module interactions can be completely unconstrained. We distinguish between *intrinsic* and *extrinsic* constraints on the order of computation. It is logically impossible for an operation that applies to a datum to be executed before that datum has come into existence. For example, the syntactic analysis of a section of the input cannot begin until at least some of the possi-

ble words in that section have been identified and at least some of their syntactic properties have been retrieved from the dictionary. This is an instance of an intrinsic ordering restriction. On the other hand, it is not logically necessary for the entire set of word or dictionary possibilities to be explicitly available prior to any syntactic operations, although "outside-in" systems impose this kind of extrinsic ordering constraint. Other control regimes might enforce different extrinsic constraints: left-to-right models, for example, require that syntactic processing be completed early in the input before segmentation is even attempted in later sections. Clearly, a model that imposes no extrinsic constraints and ensures that all intrinsic constraints are satisfied will provide the maximum degree of freedom and safety for the exercise of heuristic control strategies, and consequently, should result in the most efficient and effective natural language processors.

The multi-processing control model described below constitutes a framework in which these ideal conditions can be met. It has evolved from earlier work on a "General Syntactic Processor" (GSP), which has been discussed in some detail in another paper (Kaplan, in press[2]). The essential concepts of GSP, insofar as they are relevant to multi-processing, are presented in the following section. A later section describes the advantages of using asynchronous processes within the syntactic component of a natural language system.

AN OVERVIEW OF GSP

GSP is a relatively simple and compact syntactic processor that can emulate the operation of several other powerful algorithms, including Woods' augmented transition network (ATN) parser,[5] Kay's "powerful parser,"[2,3] and transformational generators.[1] This is possible because GSP incorporates in a general way some of the basic facilities that the other algorithms have in common. Most important for the present discussion, it gives explicit recognition to the fact that syntactic strategies are inherently non-deterministic, consisting of many alternatives that must be computed independently. Within this non-deterministic organization, GSP grammars are represented as arbitrary procedures as Winograd[5] has recently advocated. GSP also provides a small set of primitive operations that can be invoked by grammatical procedures and which seem sufficient for most ordinary syntactic strategies.

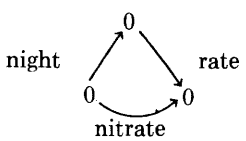
There are two distinct sources of ambiguity in syntactic processing: *grammatical alternatives* and *structural alternatives*. Grammatical alternatives occur because natural language sentences and phrases can be realized in many different ways. For example, sentences in English can be either transitive ('Mother cooked the roast.') or intransitive ('Mother cooked.'), while noun phrases might or might not begin with determiners ('the men' versus 'men'). A grammar must somehow describe the myriad patterns that the language allows, and one of the most elegant ways to express such possibilities is in the form of a (suitably augmented) finite-state transition network. (See Woods[6] for a fuller discussion of these issues). Thus GSP grammars are transition networks consisting of sets of states connected by labeled directed arcs. Paths through the grammar following arcs from state to state denote the sequences of formatives that may occur in valid phrases, while multiple arcs emanating from a single state indicate grammatical alternatives that correspond to non-deterministic choices. In actual practice, a GSP grammar is a collection of such networks, each one characterizing some type of syntactic constituent (e.g., sentence, noun phrase, verb phrase, prepositional phrase). These sub-grammars are tantamount to the multiple levels of an ATN grammar or the collection of rules interpreted by the Kay parser.

GSP also gives a formal account of the second source of ambiguity, structural alternatives. As noted above, the input to a syntactic analyzer might contain multiple possibilities, because of uncertainties at the input and dictionary look-up stages (the nitrate/night-rate example). Other alternatives not foreshadowed in the input might arise from the operation of various parts of the grammar. For example, the noun phrase sub-grammar must indicate that the string following 'saw' in sentence (1a) can be parsed as a noun phrase in at least three ways, corresponding to the interpretations (1b-d):

(1) (a) I saw the man in the park with the telescope.
    (b) The man had the telescope.
    (c) The park had the telescope.
    (d) I used the telescope to see the man.

The three noun phrase possibilities must be processed by the sentence level network to produce, in the absence of semantic constraints, two distinct parses for the entire sentence, thus signifying that it is globally ambiguous. The point is that a syntactic processor must be capable of handling structural alternatives in its input and intermediate results, as well as producing them as output.

This being the case, GSP provides a single data organization to represent constituents at all stages of processing. This data structure, called a *chart*, is also a transition network with states and arcs (which are called *vertexes* and *edges* to avoid confusion with the grammar networks). Edges correspond to words and constituents, while vertexes correspond to junctures between constituents. A sequence of edges through the chart therefore represents a single interpretation of the input, and structural alternatives are specified by multiple edges leaving a vertex, as portrayed in (2):



In this simple figure, the arrows indicate the temporal order of constituents, with an edge pointing to the set of

its possible successors. In fact, a temporal successor is just one of many properties that an edge can have. As a bare minimum, an edge that corresponds to a parent node in a linguistic tree must have a pointer to its daughter vertex, and Kaplan[2] (in press) has described the chart as a compact representation for a "family of strings of trees." An edge may have a variety of other syntactic and semantic properties, and the chart is in fact a very general, very complicated kind of syntactic graph.

The basic function of GSP is to apply a grammar network to a chart. Starting at some state and some vertex, GSP compares every arc with every edge. Unlike the arcs of an ordinary finite-state grammar, the labels on GSP arcs are interpreted as sequences of operations to be performed. Some of these operations are predicates which determine the admissibility of a transition by examining not only properties of the current edge but also any information that might have been saved away in named "registers" by previous transitions along this path. If a transition is permitted, other operations on the arc will be executed. These operations can build structures, add them to registers for use on later transitions, or insert them into the chart as structural alternatives at some vertex. Finally, the operations can cause GSP's attention to shift (non-deterministically) to a new state and a new vertex, usually one of the successors of the current edge, where the edge-arc comparison procedure is repeated. Notice that the programs on the arcs give each GSP subgrammar the power of a Turing machine, since registers can store an arbitrary amount of information which can be used to affect transitions on later arcs.

This brief discussion should convey a general feeling for how GSP is organized and what it does. In terms of multiprocessing, the most important points to remember are the following: syntactic processing is conceptualized as the interplay between two transition networks, a grammar and a chart. The grammar is stable and is interpreted as a collection of active procedures that operate on the chart. On the other hand, the chart is both dynamic and passive: it is constantly being augmented with new edges and vertexes as the grammar recognizes new structural possibilities, but it does not usually call for the execution of any actions.

MULTI-PROCESSING IN SYNTAX

Both the augmented transition network and Kay parsers guarantee that the intrinsic restrictions on the order of computation will be met, but they also impose certain extrinsic constraints. The popularity of these algorithms is due in part to the fact that their extrinsic constraints have the desirable effect of cutting down on the size of the computation space, but this is not always done in the most advantageous way. Certain benefits achieved by the ATN parser, which is essentially a top-down algorithm, are lost by the bottom-up Kay procedure, and vice versa. In this section we briefly outline these parsing strategies and point out some of their weaknesses and disadvantages. We then introduce the basic elements of the parallel processing model and show how this control regime can eliminate extrinsic constraints in syntax and lead to a mixture of bottom-up and top-down benefits.

The ATN parser begins processing at the highest syntactic level (the start of the sentence sub-grammar) and at the earliest part of the input, it being assumed that the beginning of the input corresponds to the left boundary of a well-formed sentence. Control passes from left to right through the grammar and input until an arc is reached that calls upon some other sub-grammar to compute a lower-level constituent (e.g. a noun phrase). At this point, the status of computation in the sentence network is stored away on a stack, computation is suspended, and control passes to the beginning of the lower network. If a noun phrase is identifiable at that position in the input, it is constructed and returned to the suspended sentence level, which resumes its left to right scan. Notice that the lower level computations are initiated only when their results will fit into some globally consistent pattern, so that much unnecessary computation can be avoided. For example, in the sentence 'The man left' this strategy will never look for a noun phrase beginning at 'left', since the sentence level has no need for it. Furthermore, the ATN parser can easily stop when the first complete sentence is found; if it is semantically interpretable, the additional effort to discover other parses can be eliminated.

Unfortunately, this strategy is unrealistic in several respects. Very few of the utterances in ordinary conversation are complete, well-formed sentences in which the left boundary is clearly discernible. Instead, conversation consists of a sequence of fragments and partial constituents which cannot be handled in a top-down way, even though they are perfectly understandable to human observers. A noun phrase response to a direct question should be recognized as a noun phrase even though it is not embedded in a sentential context. This is a case in which the extrinsic constraints have mistakenly ruled out some meaningful structural alternatives.

A second shortcoming of the top down strategy is that in its most naive form, it can lead to the repetition of certain computations. Consider the garden path sentence (3):

(3) The cherry blossoms in the orchard are beautiful.

If the parser first assumes that 'blossoms' is the main verb, it will expect the sentence to end after 'orchard' and be quite surprised to find the trailing verb phrase. To obtain the correct analysis, it must back up and change its hypothesis about the syntactic category of 'blossoms,' and then rescan the rest of the string. Unless special care is taken, the computation in which 'in the orchard' was recognized as a prepositional phrase will be re-executed. The top-down extrinsic constraints are not sufficient to forestall this useless effort. To avoid this, the previous structure must be retained and made available to subsequent analysis paths.

The chart provides a convenient repository for such well-formed substrings (WFSS): we can simply associate an extra vertex with the 'in' edge to record the prepositional phrases that have been identified at that position. Then the grammar operations must be modified so that they consult the chart to see if such a vertex exists before invoking the prepositional phrase sub-grammar again. Notice the control problems that arise with this solution: in general there may be several ways of realizing a particular phrase at a particular edge (e.g., 'old men and women' might or might not mean that the women are old). If all the possibilities are computed and inserted at the WFSS vertex together, then some of the work necessary to do this might turn out to be superfluous when we stop after finding the first parse. However, if the alternative phrases are constructed and inserted one at a time, there is the risk of backing up, rescanning, and encountering a WFSS vertex that is only partially complete. Thus there appears to be a trade-off between searching for the first parse and utilizing the simple WFSS mechanism to full advantage.

A top-down strategy builds a constituent only on demand, when it is needed to fulfill a higher-level phrase. The bottom-up Kay algorithm builds all types of phrases wherever they can be recognized in the chart, independent of any larger context. This is done in such a way that when needed by a higher-level computation, a phrase will have already been entered in the chart if it can be recognized at all. The chart becomes a collection of WFSS tables, one for each identified constituent at each edge, and all sub-grammars operate as though they were rescanning previously analyzed sections of the chart. In terms of the description above, the main function of the Kay parser's extrinsic ordering constraints is to preclude encounters with incomplete WFSS vertexes. The essential restriction is that examination of a vertex must be postponed until every vertex to its right has been exhaustively processed by every sub-grammar. In effect, an external controller must simulate a garden-path back up, invoking the sub-grammar at each step (see Kaplan[2] (in press) for a discussion of one way in which this constraint may be implemented).

The Kay algorithm overcomes some of the difficulties of the ATN parser. The total reliance on the well-formed substring machinery ensures that no computation will ever be repeated, so that analyses of sentences that require back up should be more efficient. In addition, the Kay strategy has no trouble identifying the fragments prevalent in natural discourse. These advantages are purchased at some cost, however. The Kay parser can handle fragments because it compulsively searches for every type of constituent in every position, but this can involve a considerable amount of wasted effort when the input is in fact well-formed. Furthermore, although it is possible to stop after the first parse has been discovered, this does not really help to reduce the amount of computation as it does for the ATN. The first parse will not emerge until the back up has reached the beginning of the chart, at which point the processing of the entire computation space is virtually complete. Thus the Kay algorithm is even less amenable to heuristic guidance than the top-down ATN parser.

For both syntactic algorithms, the extrinsic constraints are enforced to govern the circulation of information between the different sub-grammar networks, to make sure that the results of one computation are available when needed by another. The simple top-down approach is to invoke a sub-grammar each time its results are needed by another; the bottom-up approach computes results *before* they are needed and saves them in the chart, where they can be accessed on demand. Of course, the order in which alternatives are considered *within* a single sub-grammar can freely vary without serious global consequences, and this is one area where heuristic strategies can be very helpful. If processing will stop after the first parse, heuristics can direct the parser to find the most likely analysis with the minimum computation, and the effort to obtain a meaningful interpretation can be drastically reduced.

What happens if heuristics are used to change the interactions *between* the independent sub-grammars? In many cases the violation of an extrinsic constraint will have no deleterious effects (which is why it is extrinsic). Often, significant reductions in the amount of computation can be achieved. However, if the violation causes premature scanning of an incomplete vertex, some meaningful analyses may never be discovered. The multi-processing framework to which we now turn provides a general solution to the incomplete vertex problem, thereby eliminating the necessity for extrinsic ordering constraints.

Basically, we conceive of the various sub-networks of a GSP grammar as a collection of asynchronous processes. They operate on overlapping chart sections and use the chart to communicate with one another. For example, a noun phrase process can produce a structure and place it in the chart so that it will be found by any process that can make use of it (such as those corresponding to the prepositional phrase or sentence sub-grammars). Syntactic processes can cause other processes to be initiated, but once created they are, in principle, independent entities and can exert no direct control over each other's activities. Coordination between processes is entirely intrinsic, determined solely by their internal schedules of information production and consumption.

When a process is created at an edge in the chart, a vertex is established to serve as a communication *port* between the process and the global environment. Each edge has a process table in which the port and *type* (e.g., noun phrase) of the new process are recorded. A process is created only when some other entity wants to examine the information it produces. Since identical information will come through the ports of two processes of the same type at the same edge, an edge can only have one process of a given type. When a new consumer of information appears and tries to start a process, the edge's table is consulted to see if an appropriate process already exists. If so, the

previously established port is made available to the new consumer. In general, a process has no knowledge of how many or which entities are using the information it sends to its port, and a consumer has no idea of when or why the producer was created.

A process scans the chart, and whenever it recognizes and constructs a constituent, it sends it to its port. There the constituent gets incorporated as a new edge, and a port is thus a dynamic vertex. It has no edges when it is first established, but edges can appear at any time, as its process sends them back. If no constituents can be found, then its process will eventually exhaust itself without returning any results, and the port's edgeset will always be empty. This indicates to all consumers that the process' type of constituent does not exist at this position in the chart.

Notice that as long as a process is active, its port is essentially an incomplete vertex as described above. The intrinsic coordination of processes is accomplished in the following way: besides containing the results of its process, a port must indicate whether or not its process has terminated. When a port is created, it is given the initial marking "active," which persists until its process sends a termination signal. This must be the process' last action before it goes into dormancy.

When the process becomes inactive, any consumer coming along may scan the edges at the port as if it were an ordinary vertex. However, if the process is active, the consumer must take special precautions: if no edges have yet appeared at the port, the consumer must *wait* at the port, either temporarily suspending all computation or else shifting attention for the moment to some other analysis path. If some edges have already been returned, they may be scanned in the normal way, after which the consumer must again go into a wait state. A port may not be entirely abandoned until its process has become inactive. With these facilities for guaranteeing that intrinsic constraints are maintained, any heuristic strategies that seem promising can be used to guide the course of syntactic analysis. No matter what happens, implicit possibilities will never be lost.

To get a feeling for how the performance of the syntactic component can benefit from these facilities, consider how the ATN and Kay strategies could be implemented. The top-down approach still starts with a single sentence process at the beginning of the chart, but it no longer has to suspend itself when lower-level constituents are needed. To obtain a noun phrase, it spawns an asynchronous noun phrase process and then focuses on the structures being returned at the new port. Noun phrases will appear one by one, and each one can be examined as it arrives. If one of them looks particularly promising, the sentence process can continue scanning past it to see if it will fit into a valid parse. Up to this point, the only difference between this arrangement and the implementation described earlier is that the lower process can be executed in parallel (given an appropriately constructed computer). However, if there is a garden path and the sentence level must back up, rescan earlier context, and move forward again to look for a noun phrase in the same position, the port automatically behaves as a WFSS vertex. Furthermore, the intrinsic waiting constraints will keep track of all possibilities, even if the vertex is still incomplete when it is re-entered. Thus, without adding any new heuristics, the multi-processing framework allows all repetitive computations to be safely avoided within a basically top-down strategy.

This procedure still would not be able to handle fragments, since only the sentence process is initially activated. We still need a bottom-up approach to deal with ordinary discourse. For the Kay algorithm, the external controller will start up every type of process at every edge. If all processes compute to exhaustion, the effect will be the same as the previous Kay implementations. But suppose we introduce the simple heuristic that a process' allocation of real computing resources diminishes in proportion to the number of edges it has produced. If a noun phrase is found at a given position, resources will be shifted away from that process for awhile. Therefore, instead of focusing completely on one end of the chart and slowly backing up, computation will be more or less evenly distributed throughout the sentence. This makes it very likely that the first parse will be found relatively early in the analysis. If it is interpretable, the remainder of the computation space can be ignored, and we will have gained a top-down advantage in our bottom-up approach. However, if no complete parse is found, the bottom-up capability of recognizing fragments and partial constituents will be realized.

CONCLUSION

The multi-processing framework allows us to combine the advantages of the top-down and bottom-up approaches in a straightforward way. Even without intelligent heuristics, we have achieved a much improved syntactic analyzer, and as we learn more about how to make successful syntactic guesses, its efficiency and effectiveness should increase even more. This is also a convenient framework in which to test the heuristics that we might devise, and should be of great use in future investigations.

We are also currently investigating the ways in which this framework can be extended to other components of a natural language system. It appears that alternatives in the segmentation, word recognition, and dictionary look-up modules can all be efficiently represented in the chart, and that these modules can themselves be conceived of as collections of asynchronous processes. As in the syntactic component, these processes will sequence themselves automatically, according to their intrinsic constraints. In fact, it should be possible to turn all modules loose on the same chart at the same time, apply any kind of heuristics, without danger of forgetting alternatives. This would mean that natural language processors could be constructed with almost all extrinsic ordering constraints removed. We are currently investigating this possibility, as well as the possibility of treating a semantic network

like a chart and rules of inference as independent processes. We will report on these investigations in the near future.

REFERENCES

1. Friedman, J., "A computer system for transformational grammar," *Communications of the ACM 12*, pp. 341-348, 1969.
2. Kaplan, R., "A general syntactic processor," in Rustin, R., (ed.), *Natural Language Processing*. Algorithmics Press, New York 1973.
3. Kay, M., *Experiments with a powerful parser*, Santa Monica, The RAND Corporation, RM-5452-PR, 1967.
4. Minsky, M., Papert, S., *Research at the Laboratory Envision, Language and Other Problems of Intelligence*, Cambridge, Massachusetts Institute of Technology, Artificial Intelligence Memo No. 252, 1972.
5. Winograd, T., *Understanding Natural Language*, New York, Academic Press, 1972.
6. Woods, W., Transition network grammars for natural language analysis. *Communications of the ACM 13*, pp. 591-606.

Progress in natural language understanding—An application to lunar geology

by W. A. WOODS

Bolt Beranek and Newman Inc.
Cambridge, Mass.

INTRODUCTION

The advent of computer networks such as the ARPA net (see e.g., Ornstein et al.[3]) has significantly increased the opportunity for access by a single researcher to a variety of different computer facilities and data bases, thus raising expectations of a day when it will be a common occurrence rather than an exception that a scientist will casually undertake to use a computer facility located 3000 miles away and whose languages, formats, and conventions are unknown to him. In this foreseeable future, learning and remembering the number of different languages and conventions that such a scientist would have to know will require significant effort—much greater than that now required to learn the conventions of his local computing center (where other users and knowledgeable assistance is readily available). The Lunar Sciences Natural Language Information System (which we will hereafter refer to as LUNAR) is a research prototype of a system to deal with this and other man-machine communication problems by adapting the machine to the conventions of ordinary natural English rather than requiring the man to adapt to the machine.

English as a query language

There are two important reasons why one might want to use English as a mode of communication between a man and a machine. First, the man already knows his natural language and if he is to use a particular computer system or data base only occasionally or as a minor part of his work, then he may not have the time or inclination to learn and remember a formal machine language. Second, the human thinks in his native language, and if the mode of communication involves the free and immediate communication of ideas to the machine which the user is conceiving in the course of the interaction, then the additional effort required for him to translate his ideas into another language more suitable to the machine may slow down or otherwise interfere with the interaction. English is therefore an attractive medium because the human can express his ideas in the form in which they occur to him.

State of the art

Although the state of the art in "understanding" natural language by machine is still very limited, significant advances in this area have been made in recent years. Since Simmons' first survey of question answering systems,[4] our understanding of the mysterious "semantic interpretation" component has been made more clear by work such as Woods,[7,8] and the techniques for mechanically parsing natural language sentences have been advanced by the advent of transition network grammars and their parsing algorithms.[9,10,11] Recently, Terry Winograd's blocks world system[6] has demonstrated the potential of some of these techniques—especially those of procedural semantics. The field is now at the point where prototype applications to real problems can make significant contributions to our understanding of the problems of natural language communication with machines. It must be realized, that such applications are still essentially research vehicles, since the problems of mechanical understanding of natural language remain far from solution. However, by using real problems, rather than imaginary toy problems, as the vehicles for such research, one cannot only focus the effort on problems in need of solution, but may also reap the additional benefit of producing a system which will perform a task which someone really wants done.

Natural language understanding

I want to distinguish here between the objectives of this research, which I will call "natural language understanding" research, and the development of so called "English-like" languages and querying systems. There have been a number of computer question answering systems developed for special applications or for general purpose data management tasks which use English words and English syntactic constructions and call their languages "English-like." By this they mean that, although the statements in the language may look more or less like ordinary English sentences, the language makes no attempt to encompass the totality of English constructions that a user might want to use. Also, the interpretations of those sentences by the machine may differ from those which a user would assume without in-

struction. Essentially, the designer of such a system is trying to go as far as he can toward English with some set of techniques which is either easily implementable or computationally efficient, and when he encounters an English phenomenon which does not fit conveniently within those techniques or which is troublesome, he legislates it out of his language or provides for some restricted or modified form of the original phenomenon. In the kind of research I am describing, one seeks techniques for handling these phenomena.

The design of a *natural* English understanding system as a product is one that is slightly beyond the current state of the art and requires considerable linguistic research and further development of computational techniques before it will become a possibility. The construction of research prototypes such as LUNAR as testbeds and foci for the necessary research is an essential prerequisite for the eventual development of such products. As spinoffs from this research we will achieve better and better "English-like" systems as we go along.

Habitability

Most English-like systems, although they use English words and English syntactic constructions, have a syntax as limited and as formal as FORTRAN and require comparable amounts of effort to learn, remember, and use. This is not to say that these languages merely permit one to stick English words into some fixed English formats (although systems of that sort have also gone by the name of "English-like")—the language may contain a fairly complex grammar and a parsing algorithm for analyzing sentences with it. However, the "English-like" grammar is at best a restriction of what a full English grammar would be and at worst may bear little resemblance to ordinary English. The ease or comfort with which a user can learn, remember and obey the conventions of such a language has been considered by William Watt[5] and given the name "habitability." A habitable system is one in which the user will not be constantly overgeneralizing the capabilities of the system and venturing beyond its conventions, nor will he painfully adhere to a miniscule subset of the capabilities for fear of misinterpretation. It is very difficult to evaluate the habitability of a natural language or English-like question answering system without actual hands-on experience, and reported data on the subject (with the exception of the extremely inadequate statistics which will be reported here for LUNAR) is essentially non-existent. However, it has been my opinion of every English-like system which I have encountered that its habitability is very low, and although its techniques represent a significant advance in the state of the art, I am far from satisfied with the habitability of LUNAR.

THE LUNAR SYSTEM

LUNAR was originally developed with support from the NASA Manned Spacecraft Center as a research prototype for a system to enable a lunar geologist to conveniently access, compare, and evaluate the chemical analysis data on lunar rock and soil composition that was accumulating as a result of the Apollo moon missions. The objective of the research was to develop a natural language understanding facility sufficiently natural and complete that the task of selecting the wording for a complex request would require negligible effort for the geologist user.

The LUNAR system processes English requests in three successive phases:

(i) syntactic analysis using heuristic (including semantic) information to select the most "likely" parsings,
(ii) semantic interpretation to produce a formal representation of the "meaning" of the query to the system,
(iii) execution of this formal expression in the retrieval component to produce the answer to the request.

The language processor in LUNAR makes use of a general parsing algorithm for transition network grammars and a general rule-driven semantic interpretation procedure which were developed at Harvard University and at BBN over a period of years and which have been reported on in the literature.[7–11] In addition, LUNAR contains a grammar for a large subset of English, a set of semantic interpretation rules for interpreting requests for chemical analyses, ratios, etc., and a dictionary of approximately 3500 words, plus functions for setting up and interrogating a data base, computing averages and ratios, etc. The LUNAR system is described in detail in a technical report.[12]

All of the components of the system are implemented in BBN-LISP on the PDB-10 computer at BBN in Cambridge, Mass., running under the TENEX time sharing system with hardware paging and a virtual core memory for each user of up to 256K. The system is operational in two 256K tasks (called "forks")—one containing the parser, interpreter, grammar and dictionary, and the other containing the data base and retrieval functions. Although there is considerable overhead in running time for programs written in LISP and executed in a paged environment, the flexibility of this system has been a critical factor in the development of the present level of capability.

The data base

LUNAR contains two data base files provided by NASA MSC. One is a 13,000 entry table of chemical, isotope, and age analyses of the Apollo 11 samples extracted from the reports of the First Annual Lunar Science Conference, and the second is a keyphrase index to those reports. In this paper we will consider only the first. This table contains entries specifying the concentration of some constituent in some phase of some sample, together with a reference to the article in which the measurement was reported. (There are generally several entries for each combination of sample, phase, and constituent—measured by different investigators.)

The formal query language

The data base of the LUNAR system is accessed by means of a formal query language into which the input English requests are translated by the language processing component. The query language is essentially a generalization of the predicate calculus which could either be manipulated as a symbolic expression by a formal theorem prover to derive

intensional inferences or be executed directly on the data base to derive extensional inferences. Only the latter, extensional inference facility, is used in LUNAR.

The query language contains three kinds of constructions: *designators*, which name objects or classes of objects in the data base (including functionally determined objects), *propositions*, which are formed from predicates with designators for arguments, and *commands*, which take arguments and initiate actions. For example, S10046 is a designator for a particular sample, OLIV is a designator for a certain mineral (Olivine), and (CONTAIN S10046 OLIV) is a proposition formed by substituting designators as arguments to the predicate CONTAIN. TEST is a command for testing the truth value of a proposition. Thus, (TEST (CONTAIN S10046 OLIV)) will answer yes or no depending on whether sample S10046 contains Olivine. Similarly, PRINTOUT is a command which prints out a representation for a designator given as its argument.

The major power and usefulness of the query language comes from the use of a quantifier function FOR and special enumeration functions for classes of data base objects to carry out extensional quantification on the data base. The format for a quantified expression is:

(FOR QUANT X / CLASS : PX ; QX)

where QUANT is a type of quantifier (EACH, EVERY, SOME, THE, numerical quantifiers, comparative quantifiers, etc.), X is a variable of quantification, CLASS determines the class of objects over which quantification is to range, PX specifies a restriction on the range, and QX is the proposition or command being quantified. (Both PX and QX may themselves be quantified expressions.)

The specification of the CLASS over which quantification is to range is performed in the system by special enumeration functions which (in addition to whatever other parameters they might have) take a running index argument which is used as a restart pointer to keep track of the state of the enumeration. Whenever FOR calls an enumeration function for a member of the class, it gives it a restart pointer (initially NIL), and each time the enumeration function returns a value, it also returns a new restart pointer to be used to get the next member.

The use of enumeration functions for quantification frees the FOR function from explicit dependence on the structure of the data base—the values returned by the enumeration function may be searched for in tables, computed dynamically, or merely successively accessed from a precomputed list. A general purpose enumeration function SEQ can be used to enumerate any precomputed list. For example:

(FOR EVERY X1 / (SEQ TYPECS) : T ;
(PRINTOUT X1))

is an expression which will printout the sample numbers for all of the samples which are type C rocks (i.e. breccias).

The principal enumeration function for the chemical analysis data base is the function DATALINE which takes arguments designating a data file, a sample, a phase name, and a constituent. DATALINE enumerates the lines of the data file which deal with the indicated sample/phase/constituent triple. Other complex enumeration functions are NUMBER and AVERAGE which take an argument format similar to the FOR function and perform counting and averaging functions.

CAPABILITIES OF THE CURRENT SYSTEM

The current LUNAR prototype permits a scientist to easily compare the measurements of different researchers, compare the concentrations of elements or isotopes in different types of samples or in different phases of a sample, compute averages over various classes of samples, compute ratios of two constituents of a sample, etc.—all in straightforward natural English. The system removes from the user the burden of learning the detailed formats and codes of the data base tables, or learning a special programming language. For example, the system knows the various ways that a user may refer to a particular class of samples, it knows whether a given element is stored in the data base in terms of its elemental concentration or the concentration of its oxide, it knows what abbreviations of mineral names have been used in the tables, etc., and it converts the user's request into the appropriate form to agree with the data base tables. Thus, the system has made significant strides toward the goal of habitability. The following examples will illustrate some of the kinds of operations LUNAR performs.

The most typical example of a request which a geologist might make to the LUNAR system is illustrated by the protocol:

```
38**(WHAT IS THE AVERAGE CONCENTRATION OF ALUMINUM IN HIGH ALKALI ROCKS)
***
PARSING
1331 CONSES
4.987 SECONDS
INTERPRETING
2427 CONSES
11.025 SECONDS
INTERPRETATIONS:
(FOR THE X13 / (SEQL (AVERAGE X14 / (SSUNION X15 / (SEQ TYPEAS) : T ; (DATALINE (WHQFILE X15)
X15 (NPR* X16 / (QUOTE OVERALL)) (NPR* X17 / (QUOTE AL203)))) : T)) : T ; (PRINTOUT X13))

BBN LISP-10 03-09-72 . . .
EXECUTING
(8.134996 . PCT)
```

(Here, the system has typed the two asterisks, the user typed the question, beginning and ending with parentheses, and the system typed the rest. The comments 1331 CONSES and 4.987 SECONDS give a record of the memory resources and the time used during the parsing phase. A similar record is generated for the interpretation phase: The expression following the comment INTERPRETATIONS: is the formal retrieval program which is executed in the data base to produce the answer). This request illustrates a number of features of the system:

1. The user types the question exactly as he would say it in English (terminal punctuation is optional and was omitted in the example).
2. The system has translated the phrase "high alkali rocks" into the internal table form TYPEAS.
3. The system has filled in an assumed OVERALL phase for the concentration since the request does not mention any specific phase of the sample in which the concentration is to be measured.
4. The system is capable of computing answers from the data base as well as simply retrieving them (the average was not stored information).

Perhaps the simplest operation which the system will perform for the user is to collect and list selected portions (not necessarily contiguous) of the data base. For example, for a request "Give me all analyses of S10046," the system protocol would be:

```
***
37**(GIVE ME ALL ANALY0ES OF S10046)
***
PARSING
1456 CONSES
9.445 SECONDS
INTERPRETING
2112 CONSES
8.502 SECONDS
INTERPRETATIONS:
(DO (FOR EVERY X9 / (SSUNION X12 / (SEQ MAJORELTS) : T ; (DATALINE (WHQFILE (NPR* X19 / (QUOTE
S10046))) (NPR* X10 / (QUOTE S10046)) (NPR* X11 / (QUOTE OVERALL)) X12)) : T ; (PRINTOUT X9)))

BBN LISP-10 03-09-72 . . .
EXECUTING

I HAVE 15 HITS
DO YOU WANT TO SEE THEM? YES
3956    S10046    OVERALL    S102    44.06752    PCT    D70-235
3967                         T102    8.3405
3968                                 6.50559             D70-254
3865                         AL203   11.7149             D70-235
3900                         FEO     16.9818
3901                                 15.438              D70-254
3928                         MNO     .20659              D70-235
3929                                 .22725              D70-254
3927                         MGO     9.11845             D70-235
3875                         CAO     13.71216
3917                         K20     .20478
3918                                 .19515              D70-242
3919                                 .14455              D70-254
3933                         NA20    .4718               D70-235
3934                                 .50146              D70-254
```

This example illustrates some additional features of the system. Again, since no phase was mentioned, the system assumed the OVERALL phase (i.e. the rock as a whole). If the user had wanted to see all the phases, he could have said explicitly "for all phases." Similarly, since no specific elements or isotopes were mentioned, the system assumed a standard list of major elements was intended (Our geologist informant assures us that this is what a geologist would mean by such a question). Again, if the user really wanted to see all chemical element analyses, he could say so explicitly.

The comment I HAVE 15 HITS DO YOU WANT TO SEE THEM? illustrates another feature of the system. If the result of a request is more than 5 "hits" (effectively 5 lines of output), the system types this comment and gives the user the option of listing them offline.

In addition to averaging and listing, the system can also compute ratios, count, and interpret some anaphoric references and comparatives as indicated in the following examples:

```
31**(HOW MANY BRECCIAS CONTAIN OLIVINE)
***
PARSING
815 CONSES
4.633 SECONDS
INTERPRETING
1514 CONSES
7.29 SECONDS
INTERPRETATIONS:
(FOR THE X12 / (SEQL (NUMBER X12 / (SEQ TYPECS) : (CONTAIN X12 (NPR* X14 / (QUOTE OLIV)) (QUOTE
NIL)))) : T ; (PRINOUT X12))

BBN LISP-10 03-09-72 . . .
EXECUTING
(5)

-----
***
32**(WHAT ARE THEY)
***
PARSING
487 CONSES
2.755 SECONDS
INTERPRETING
1158 CONSES
4.053 SECONDS
INTERPRETATIONS:
(FOR EVERY X12 / (SEQ TYPECS) : (CONTAIN X12 (NPR* X14 / (QUOTE OLIV)) (QUOTE NIL)) ; (PRINTOUT
X12))

BBN LISP-10 03-09-72 . . .
EXECUTING
S10019
S10059
S10065
S10067
S10073
-----
***
34**(DO ANY SAMPLES HAVE GREATER THAN 13 PERCENT ALUMINUM)
***
PARSING
981 CONSES
4.614 SECONDS
INTERPRETING
902 CONSES
3.566 SECONDS
INTERPRETATIONS:
(TEST (FOR SOME X16 / (SEQ SAMPLES) : T ; (CONTAIN' X16 (NPR* X17 / (QUOTE AL203)) (GREATERTHAN
13 PCT))))

BBN LISP-10 03-09-72
EXECUTING
YES.
T
-----
```

```
***
35**(LIST K / RB RATIOS FOR BRECCIAS)
***
PARSING
662 CONSES
3.366 SECONDS
INTERPRETING
1642 CONSES
6.537 SECONDS
INTERPRETATIONS:
(DO (FOR GEN X9 / (SSUNION X10 / (SEQ TYPECS) : T ; (RATIO (QUOTE KO2) (QUOTE RB) X10 (NPR* X11 /
(QUOTE OVERALL)))) : T ; (PRINTOUT X9)))

BBN LISP-10 03-09-72
EXECUTING
I HAVE 17 HITS
DO YOU WANT TO SEE THEM? YES
(472.2222  S10018  D70-205)
(473.5884  S10018  D70-242)
(518.2477  S10019  D70-218)
(345.4411  S10019  D70-256)
(463.3003  S10021  D70-242)
(568.8333  S30046  D70-235)
(462.4408  S10046  D70-242)
(408-2933  S10048  D70-220)
(566.1499  S10056  D70-235)
(480.1913  S10059  D70-253)
(481.85    S10060  D70-235)
(457.9177  S10060  D70-242)
(487.5714  S10060  D70-248)
(489.1304  S10061  D70-205)
(458-9973  S10065  D70-236)
(473.1551  S10065  D70-258)
(500.173   S10073  D70-215)
---
```

The system also understands restrictive relative clauses and certain adjective modifiers (some of which cause restrictions on the range of quantification of the noun phrase and some of which change the interpretation of the noun they modify). Some other modifiers (such as "lunar" modifying "samples") are known to be redundant and are deliberately ignored. The following example contains all three:

```
46**(LIST MODAL PLAG ANALYSES FOR LUNAR SAMPLES THAT CONTAIN OLIV)
***
PARSING
1099 CONSES
4.346 SECONDS
INTERPRETING
2774 CONSES
12.33 SECONDS
INTERPRETATIONS:
(DO (FOR GEN X20 / (SSUNION X1 / (SEQ SAMPLES) : (CONTAIN X1 (NPR* X3 / (QUOTE OLIV)) (QUOTE
NIL)) ; (DATALINE (WHQFILE X1) X1 OVERALL (NPR* X4 / (QUOTE PLAG)))) : T ; (PRINTOUT X20)))

BBN LISP-10 03-09-72
EXECUTING
I HAVE 13 HITS
DO YOU WANT TO SEE THEM? YES
1679   S10020   OVERALL   PLAG   30.7   ***   D70-159   0
1680                             21.4         D70-173   31
1681                             28.5                   40
```

1682		24.6	D70-305	0
2141	S10022	15.6	D70-179	
3109	S10044	33.1	D70-154	41
3110		34.1		42
4440	S10047	37.8	D70-159	0
5796	S10058	37.1	D70-155	
8582	S10072	20.4	D70-173	
8583		18.5	D70-179	
9311	S10084	22.0	D70-186	
9312		15.0	D70-304	

The structure of the formal query language for accessing the data base and the techniques for semantic interpretation enable the user to make very explicit requests with a wide range of diversity within a natural framework. As a natural consequence of the arrangement, it is possible for the user to combine the basic predicates and functions of the retrieval component in ways that were not specifically anticipated, to ask questions about the system itself. For example, one can make requests such as "List the phases.", "What are the major elements?", "How many minerals are there?", etc. Although these questions are not likely to be sufficiently useful to merit special effort to handle them, they fall out of the mechanism for semantic interpretation in a natural way with no additional effort. If the system knows how to enumerate the possible phases for one purpose, it can do so for other purposes as well. Furthermore, anything that the system can enumerate, it can count. Thus, the fragmentation of the retrieval operations into basic units of quantifications, predicates, and functions provides a very flexible and powerful facility for expressing requests.

EVALUATION

As I said in my introduction, the construction and evaluation of research prototypes such as LUNAR is an essential prerequisite to the development of natural English understanding systems. It is natural at this point to ask, "How close to that goal are we?" "How natural is the communication with LUNAR?" In this section, we will consider some of the data both statistical and anecdotal which we have available. It is admittedly grossly inadequate to the task but may help to give some impression of what can be achieved with the techniques used in LUNAR and also some of the problems that still remain with this and other natural language understanding systems.

Demonstration of the prototype

At the Second Annual Lunar Science Conference, in Houston, Texas, January 11-13, 1971, a version of LUNAR system was run as a demonstration twice a day for three days. During this time the lunar geologists attending the conference were invited to ask questions of the system. Approximately 110 requests were processed, many of which were questions whose answers would contribute to the work of the requestor and not merely "toy" questions to see what the system would do. These requests were limited to questions which in fact dealt with the data base of the system (many people asked their questions before they could be told what the data base contained) and were restricted to not contain comparatives (which we did not handle at the time). The requests were freely expressed, however, without any prior instructions as to phrasing and were typed into the system exactly as they were asked.

Of 111 requests entered into the system during the three days, 10 percent of them failed to perform satisfactorily because of parsing or semantic interpretation problems. Another 12 percent failed due to trivial clerical errors such as dictionary coding errors which were easily corrected during or immediately after the demonstration. The remaining 78 percent of the requests were handled to our complete satisfaction, and with the correction of trivial errors, 90 percent of the questions expressed fell within the range of English understood by the system. This performance indicates that our grammar and semantic interpretation rules, which were based on the information of a single geologist informant, did indeed capture the essential details of the way that geologists would refer to the objects and concepts contained in our data base. Examples of the requests which were processed are:

(GIVE ME THE AVERAGE SM ANALYSIS OF TYPE A ROCKS)
(WHAT IS THE AVERAGE MODAL CONCENTRATION OF ILMENITE IN TYPE A ROCKS?)
(GIVE ME EU DETERMINATIONS IN SAMPLES WHICH CONTAIN ILM)
(GIVE ME ALL K / RB RATIOS FOR BRECCIAS).
(WHAT BE ANALYSES ARE THERE?)
(GIVE ME OXYGEN ANALYSES IN S10084)
(WHAT SAMPLES CONTAIN CHROMITE?)
(WHAT SAMPLES CONTAIN P205?)
(GIVE ME THE MODAL ANALYSES OF P205 IN THOSE SAMPLES)
(GIVE ME THE MODAL ANALYSES OF THOSE SAMPLES FOR ALL PHASES)

(DOES S10046 CONTAIN SPINEL?)
(WHAT PHASES DOES S10046 HAVE?)
(WHAT IS THE AVERAGE CONCENTRATION OF IRON IN ILMENITE)
(GIVE ME REFERENCES ON SECTOR ZONING)
(GIVE ME REFERENCES ON ABYSSAL BASALTS)
(GIVE ME ALL IRON / MAGNESIUM RATIOS IN BRECCIAS)
(GIVE ME ALL SC46 ANALYSES)
(WHAT SOILS CONTAIN OLIV)
(GIVE ME ALL OLIV ANALYSES OF S10085)
(WHAT ARE ALL TUNGSTEN ANALYSES?)
(GIVE ME IRON ANALYSES FOR PLAGIOCLASE IN S10022)
(GIVE ME ALL ZIRCONIUM CONCENTRATIONS IN ILMENITES)

Anecdotal data

The above statistics have to be evaluated with several grains of salt. However, they do give some impression of the habitability of the system to lunar geologists for which it was tailored. The results must be balanced by the experience of non-geologists who have tried to use the system. In one anecdotal example, which is perhaps typical of the failures of LUNAR and other attempted natural language understanding systems, the first question asked of the system by a graduate student in psychology (given only the instructions "ask it a question about the moon rocks") was "What is the average weight of all your samples?" This sentence failed even to parse due to the fact that the grammar of the system did not know that some determiners in English (such as "all") can be used as a "predeterminer" in addition to another determiner in the same noun phrase. Both "What is the average weight of all samples?" and "What is the average weight of your samples?" would have parsed. Assuming that the request had parsed (the grammar has now been expanded to the point where it would), the system would still not have understood it for several reasons. First, the system contains no semantic rules for interpreting ownership or possession (no one had ever attributed ownership of the samples to the system before). Secondly, LUNAR's data base does not contain information about the weights of samples and consequently there are no semantic rules for interpreting "weight of." This same student had to ask 5 successive questions before he found one that the system could understand and answer. This of course is not surprising for questions that are not even constrained to be about the contents of the data base. On another occasion, a class of graduate students in information retrieval given an appropriate introduction spent an hour and a half asking it questions and found only two that it failed to handle correctly.

The difference in performance for geologist users and non-geologists is not just that the geologists use technical jargon with which the layman is not familiar, but also their familiarity with the material leads to certain habits in language which do not cover the full spectrum of possible English constructions. Thus, in tailoring a system to the geologist, we have concentrated our effort (although not to the exclusion of all else) on those linguistic phenomena and problems which the geologists actually use. The non-geologist strays outside of this habitable region more easily than the geologist and this is the source of most of the problems in the above example.

There are other limitations of the system—even for a geologist user—which do not show up in the statistics of the demonstration. For example, although the parsing system contains experimental versions of some of the most sophisticated conjunction handling of any mechanical grammar to date[11] the handling of conjunctions in the system is still far from adequate. Although none of the questions during the demonstration involved complicated conjunction constructions, it is also true that no single person asked more than a few questions during the conference. I am confident that if a geologist really sat down to use the system to do some research he would very quickly find himself wanting to say things which are beyond the ability of the current system.

Linquistic fluency and completeness

There are two scales which can be used to measure the performance of a system such as LUNAR. We can call them *completeness* and *fluency*. A system is *complete* if there is a way to express any request which it is logically possible to answer from the data base. The scale of *fluency* measures the degree to which virtually any way of expressing a given request is acceptable. The two scales of completeness and fluency are independent in that it is possible to have a fluent system which will accept virtually any variations on the requests which it accepts, but which is nevertheless incomplete. Likewise, a system may be logically complete but very restricted in its syntax. A natural language system which is incomplete cannot answer certain questions, while a system that is not fluent is difficult to use.

Fluency of LUNAR

The LUNAR prototype is quite fluent in a few specific constructions. It will recognize a large number of variations on the request "give me all analyses of constituent x in phase y of sample z." It knows many variations of "give me" and many different variations on "analysis." However, there are other requests which (due to limitations in the current grammar) must be stated in a specific way in order for the grammar to parse them and there are others which are only understood by the semantic interpreter when they are stated in certain ways.

In the area of syntax, the LUNAR system is very competent. Most questions that fail to be understood have nevertheless parsed correctly, and questions having nothing at all to do with lunar geology parse equally well. The grammar knows about such things as embedded complement constructions, tense and modality, and other linguistic phenomena that probably do not occur in the lunar geology application.

Most of the limitations of fluency in the current system are due to the fact that the necessary semantic interpretation rules have not been put into the system. Continued development of the grammar and semantic rules would result in continued improvements in fluency, and there is no visible ceiling other than an economic one to the fluency which can be achieved. The following list of sentences gives a representative sample of the kinds of questions which the system can understand and the degree of variability permitted.

1. (List samples with Silicon)
   (Give me all lunar samples with Magnetite)
   (In which samples has Apatite been identified)
   (How many samples contain Titanium)
   (Which rocks contain Chromite and Ulvospinel)
   (Which rocks do not contain Chromite and Ulvospinel)
2. (What analyses of Olivine are there)
   (Analyses of Strontium in Plagioclase)
   (What are the Plag analyses for breccias)
   (Rare earth analyses for S10005)
   (I need all chemical analyses of lunar soil)
   (What is the composition of Ilmenite in rock 10017)
   (List the analyses of Aluminum in vugs)
   (Nickel content of opaques)
3. (Which samples are breccias)
   (What are the igneous rocks)
   (What types of sample are there)
   (What is the number of phases in each sample)
4. (Give me the K / Rb ratios for all lunar samples)
   (What is the specific activity of A126 in soil)
   (Give me all references on fayalitic Olivine)
   (Which rock is the oldest)
   (Which is the oldest rock)
5. (What is the average analysis of Ir in rock S10055)
   (What is the average age of the basalts)
   (What is the average Potassium / Rubidium ratio in basalts)
   (In which breccias is the average concentration of Titanium greater than 6 percent)
   (What is the average concentration of Tin in breccias)
   (What is the average concentration of Tin in each breccia)
   (What is the mean analysis of Iridium in type B rocks)
   (I want the average composition for glasses in dust)
6. (Modal Plag analyses for S10058)
   (Modal Olivine analyses)
   (Give me the modal Olivine analyses for S10022)
   (Give me all modal analyses of Plag in lunar fines)
   (List the modes for all low Rb rocks)
7. (Which samples have greater than 20 percent modal Plagioclase)
   (Which samples are more than 20 percent Plag)
   (How many rocks have greater than 50 ppm Nickel)
   (Which samples contain more than 15 ppm Barium in Plag)
   (How much Titanium does S10017 contain)
   (How much Nickel is in rock 10046)
   (How old is S10047)

Completeness of LUNAR

The criteria for logical completeness is a level of achievement that is not generally met by current data management systems with artificial request languages, much less by a system that recognizes natural language. The formal query language used for the retrieval component of LUNAR fares better than most data management systems in this respect since it is fundamentally an extension of the predicate calculus, but there are still some extensions which the language requires in order to fully achieve logical completeness.

More stringent than the incompleteness of the formal request language, there are limitations in the completeness of the subset of English handled by the system. This arises largely from the difficulties of parsing conjunction constructions, but there are also problems such as the ambiguity of the scopes of quantifiers. With some further work on conjunctions, the subset of English currently handled by the grammar should become a very convenient language to use. However, semantic interpretation techniques for this subset are far from complete and much further work is needed here.

CONCLUSION

What we have accomplished

The current LUNAR prototype represents a significant step in the direction of the goals of natural language understanding. Within the range of its data base, the system permits a scientist to ask questions and request computations in his own natural English in much the same form as they arise to him (or at least in much the same form that he would use to communicate them to another person). This is borne out by the performance of the system during the demonstration at the Second Lunar Science Conference. The system answered most of the questions dealing with its data base which were asked by the investigators during the demonstration. The effort required to cast the request into a form suitable for execution in the data base is assumed by the English processor, which translates the requests into compact "disposable" programs which are then executed in the data base. The English processor therefore functions as an automatic programmer which will convert the user's request into a tailor-made program for computing the answer. The English processor knows the ways in which geologists habitually refer to the elements, minerals, and measurements contained in its data base; it knows the specific details of the data base table layouts; and it knows the correspondence between the two. Thus, for example, the user need not know that the

mineral Olivine is abbreviated OLIV in the data base, that the concentrations of Titanium are recorded in terms of the percentage of TiO2, that the class of rocks referred to variously as "type A," "high alkali," or "fine grained crystalline" are encoded as "TYPEAS" in the data base. These facts are "known" by the natural English processor, and the user's request is automatically translated from the form in which he may ask it into the proper form for the data base.

Where we stand

Although our current system does indeed exhibit many of the qualities that we have outlined as our goals, we are still far from achieving the goal as stated. The knowledge that the current system contains about the use of English and the corresponding meanings of words and phrases concerns those English constructions which pertain to the system's data base of chemical analysis data, which has a very limited and simple structure. Indeed this data base was chosen as an initial data base because its structure was simple and straightforward. In order to incorporate additional data bases into the system, it will be necessary to provide the system with information about the ways that the users will refer to those data bases in English, the vocabulary they will use, the ways they will use that vocabulary, and the "meanings" of the words and constructions in terms of the data base tables. For some tables (those whose structure is as simple and direct as the chemical analysis table), this process may be a direct extension of the current facility and may require only the addition of new semantic rules for interpreting the new words and constructions. For other applications, however, this will require much greater sophistication in both the linguistic processing and the underlying semantic representations and inference mechanisms. One type of data which will require considerable advancement is the representation and use of data which describes surface and structural features of the samples. This data does not fit conveniently into a table or a paradigm, and the techniques for storing it, indexing it, and providing access to it for retrieval and inference remain to be developed. Indeed, it is in the handling of non-tabular information of this sort that natural language querying may hold its greatest promise, but such potential is as yet undeveloped.

REFERENCES

1. Bobrow, D. G., Murphy, D. P., Teitelman, W., *The BBN-LISP System*, BBN Report 1677, Bolt Beranek and Newman Inc., Cambridge, Mass., April 1968.
2. Myer, T. R. and Barnaby, J. R., *TENEX Executive Language*, Bolt Beranek and Newman Inc., Cambridge, Mass., January 1971.
3. Ornstein, S. M., Heart, F. E., Crowther, W. R., Rising, H. K., Russell, S. B., Michel, A., "The Terminal IMP for the ARPA Computer Network," *AFIPS Conf. Proc.* Vol. 40, pp. 243-254, 1972.
4. Simmons, R. F., "Answering English Questions by Computer: A Survey," *Communications of the ACM*, vol. 8, No. 1, pp. 53-72, January 1965.
5. Watt, W. C., "Habitability," *American Documentation*, vol. 19, No. 3, July 1968.
6. Winograd, T., *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*, MAC TR-84, MIT Project MAC, February, 1971.
7. Woods, W. A., *Semantics for a Question-Answering System*, Ph.D. Thesis, Harvard University, Cambridge, Mass., August 1967.
8. Woods, W. A., "Procedural Semantics for a Question-Answering Machine," *AFIPS Conference Proceedings*, vol. 33, 1968.
9. Woods, W. A., *Augmented Transition Networks for Natural Language Analysis*, Harvard Computation Laboratory Report No. CS-1, Harvard University, Cambridge, Mass., December 1969.
10. Woods, W. A., "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, vol. 13, pp. 591-602. October 1970.
11. Woods, W. A., *An Experimental Parsing System for Transition Network Grammars*, BBN Report 2362, Bolt Beranek and Newman Inc., Cambridge, Mass., May 1972.
12. Woods, W. A., Kaplan, R. M., and Nash-Webber, B., *The Lunar Sciences Natural Language Information System: Final Report*, BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, Mass., June 1972.

Experiments in sophisticated content analysis

by GARY R. MARTIN

University of California
Los Angeles, California

ABSTRACT

In recent years, natural language data processing research and development has turned away from its earlier goals of fully-automatic high-quality translation, and similarly unmanageable tasks, toward the construction of computational tools for a variety of more practical applications. At the Center for Computer-Based Behavioral Studies at UCLA, such tools are being used in the analysis of message sets originating in experimental gaming situations, studies of social interaction, group simulations, as well as for the analysis of outside documents of interest.

A central tool in these studies is the so-called Stanford Inquirer, a more sophisticated version of the earlier General Inquirer. The use of the Stanford Inquirer has heretofore involved considerable costs in time and money for the manual pre-coding of the object texts—an essential step in the identification of the "themes" of interest in the text. Besides being expensive, this manual text encoding has been subject to inter-code inconsistency and bias. Through automated interactive theme encoding, the Center is working to overcome these obstacles to effective, large-scale text analysis. In addition, these text processing tools are being shaped for other applications, such as automated document classification and routing.

Two approaches are called sequential coding and the other based on the Woods-Kaplan transition network analyzer will be discussed.

Modelling English conversations

by R. F. SIMMONS

University of Texas
Austin, Texas

ABSTRACT

A notable if limited degree of success has been obtained in natural language question answering systems as seen in such examples as Woods' Lunar Data Base Model and Winograd's natural language controlled hand. Most of the conversational capabilities of these systems take the form of answering questions or accepting data. Colby's recent simulation of a paranoid personality offers a freer form of conversation although the model's understanding of English is very limited. Computer based teaching systems often include a tree of conversational responses to student questions and answers but these are highly specific to exactly predictable contexts.

Our current research is concerned with modelling general conversations in English. The models may be text-based, based on a structured model of information, or a combination of these. A basic babbler accepts an input statement in restricted English, analyzes it to find matching text or data in its model, and responds with something relevant to the input. The babbler adds information to its model when input sentences are given it. The question of what is relevant as a response is the interesting line of the research. The babbler is given purposes—such as to present some given set of information. It is given some capability to model the speaker, to remember what has been said, and an ability to evaluate connotative aspects of the meanings of words. These features allow it to select a conversational response as a function of its purposes, its model of the speaker, and the denotative and connotative meanings of the input sentence. The main emphasis of the research is to discover how to present lesson material in a computer controlled conversational system.

The efficiency of algorithms and machines—A survey of the complexity theoretic approach

by JOHN E. SAVAGE

Brown University
Providence, Rhode Island

ABSTRACT

The credibility of computer science as a science depends to a large extent on the complexity theoretic results which are now emerging. In this survey the efficiency of algorithms and machines for finite tasks, i.e., tasks representable by functions with finite domain and range, will be examined.

The complexity of functions can be measured in several ways. Two useful measures to be discussed in this survey are the shortest length program for a function on a universal Turing machine and the smallest number of logical operations to compute a function.

Two storage-time tradeoff inequalities for the computation of functions on random-access general purpose computers will be stated. These imply that efficient use of these machines is possible only for algorithms using small storage and large time or the reverse. Intermediate amounts of storage and time generally imply inefficient operation.

Hypergeometric group testing algorithms

by F. K. HWANG and S. LIN

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

ABSTRACT

Given a finite population P consisting of g good and d defective objects, the problems of hypergeometric group testing (HGT) is concerned with the identification of the d defectives by means of the following test procedure. Any subset $X \subset P$ can be tested with two possible results: (1) either all elements of X are good, or (2) at least one element of X is defective. In the latter case, we have no knowledge as to which ones or how many are defective. In recent years, various algorithms have been proposed to solve this problem with the aim of minimizing the maximum number of tests required. However, no optimal algorithm is known at present for general g and $d>1$.

We define an algorithm s to solve the HGT problem to be an l-set algorithm if throughout the entire process of implementing s, we only need to partition the still unclassified elements of P into at most l sets ($P_1, \cdots P_l$) where objects in each P_i need not be differentiated from one another. Restricting s to be an l-set algorithm limits the range of useful tests we can make as the amount of information we can keep after each successive tests depends on l. We show that all previously known *HGT* algorithms are either 1 or 2-set algorithms. By increasing l, we are able to construct some new classes of *HGT* algorithms which are more efficient than all previously known algorithms. Finally, we are able to construct optimal *HGT* algorithms for $l \leq 3$.

The file transmission problem

by PETER WEINER

Yale University
New Haven, Connecticut

ABSTRACT

The *file transmission problem* is to determine the best way to send a file A (assumed to be a linear string over a finite alphabet) from one computer to another via a transmission line, assuming that the receiving computer has access to another file B called the base file. In addition to sending the characters of A directly, we allow the transmission of a copy command which directs the receiving computer to append a specified, but variable length, substring of characters taken from the base file to the end of the file under construction. The cost of transmission is taken as the sum of the number of characters directly sent and K times the number of copy commands. An *optimal derivation* of A is a minimum-cost sequence of characters and copy commands which allow the receiving computer to construct the file A. We present an algorithm for obtaining an optimal derivation. This algorithm is itself optimal in that both its run time and storage requirements are linear functions of the lengths of A and B.

Min-max relations and combinatorial algorithms

by W. PULLEYBLANK

University of Waterloo
and
I.B.M. Canada Ltd.
Waterloo, Ontario, Canada

ABSTRACT

Min-max relations are an important tool in the development of combinatorial algorithms, for they provide a means of determining when an optimal solution has been obtained and a means of demonstrating the optimality of the solution. Many combinatorial problems can be expressed as integer programming problems. When a set of linear inequalities sufficient to define the convex hull of the feasible solutions is known, linear programming duality immediately yields a min-max theorem.

We discuss these ideas with respect to the weighted matching problem and describe several min-max theorems which can be obtained in this fashion.

Analysis of sorting algorithms

by C. L. LIU

University of Illinois
Urbana, Illinois

ABSTRACT

By analyzing an algorithm, we mean to study the performance of an algorithm including the assertion of its correctness and a determination of the cost of its execution. Although a given algorithm is often analyzed in a particular way that is most suitable for such an algorithm, we are more interested in general procedures and techniques that can be used to study the performance of classes of algorithms. To be able to talk about general analysis techniques will not only add to our understanding of the behavior of a class of algorithms but will also, in many cases, lead to useful synthesis procedures. A good example illustrating these points is the various techniques that can be used to analyze a class of sorting algorithms which can be modelled as networks made up of comparator modules. In this paper, we discuss several approaches to such an analysis problem. Moreover, synthesis procedures suggested by these analysis techniques will also be presented.

In search of the fastest algorithm

by IAN MUNRO

University of Waterloo
Waterloo, Ontario, Canada

ABSTRACT

Problems from many disciplines are frequently reduced to graph theoretic questions. This leaves the challenge of finding a solution to the graph problem as efficiently (cheaply) as possible. We will discuss techniques by which some of these questions may be answered in what is more or less a single scan of the input. Our attention then turns to problems that seem a little harder; that is, to problems for which there are trivially algorithms taking time polynomial in the amount of input, but no known algorithms which take time linear in the amount of data. This class includes, among many other problems, maximum matching in bipartite graphs and digraph transitive closure. We will discuss surprisingly fast algorithms for certain of these problems and lament our inability either to do better or to prove we cannot do better.

Introduction to the theory of medical consulting and diagnosis*

by EDWARD A. PATRICK,** LEON Y. L. SHEN and FRANK P. STELMACK

Purdue University and Regenstrief Institute for Health Care
Lafayette, Indiana

INTRODUCTION

Medical consulting and diagnosis is not just a matter of storage and retrieval of information or of computing the a posteriori probability of disease. A physician must interact with numerous components of the health care team such as a nurse who may communicate his orders for tests or drugs, the physician consultant who advises him, and a textbook or literature which provides him with information about diseases. The creation of a physician through the medical school process causes the build-up of thought processes in the physician orientated both to consulting and decision making. This thought process is not an accidental one but it is dictated by the interrelationship of the components of the health care team. It is during the building of this thought process, in medical school, that computer assistance to medical consulting and diagnosis has a good chance of becoming part of the physician's "bag of tricks". In addition, computer assistance in medical consulting and diagnosis can be an aid to education during medical school.

For the past three years we have been working to develop a theory of medical consulting and diagnosis by using: the tools of pattern recognition and computers; the experience of actually going to medical school, and the requirements of physicians in practice. On one hand there is the challenge to develop a system with good performance which will be used by new physicians and physicians already in practice; while on the other hand there is the challenge to anticipate a new health care delivery system more efficient than the present one. The theory introduced in this paper is basic to both of these approaches.

The theory requires that we define *classes, measurements,* and *features* of a *system.* Then provision is made for *subsystems* of the system where a *subset of classes* and a *subset of measurements or features* are defined for each subsystem. The number of classes (diseases) and measurements (signs, symptoms, and lab tests) is very large in medicine; the use of subsystems helps to resolve that problem. Subsystems can be defined as organ systems such as renal, cardiovascular, and respiratory, or as disease classifications such as bacterial, viral, myocardial, endocardial, etc. The distinction between a feature and measurement will become clearer later in the paper, but essentially a feature is a function of measurements.

The measurement vector is denoted $\mathbf{x}=[x_1, x_2, \ldots x_L]$ and the classes $\omega_1,\omega_2, \ldots, \omega_M$ for the medical system.[1] There exists a *class conditional probability density* $p(\mathbf{x} \mid \omega_i)$, for each class (disease), $_i=1, 2, \ldots, M$ which generally is unknown.

A *measurement or feature is significant* for a class ω_i if and only if the a posteriori probability of class ω_i is affected by the value of that measurement or feature. This is very important because although the number of measurements L is very large (thousands), the number of significant measurements or features for a particular class is relatively small. This leads to the definition of a *class-measurement relationship* for each class defined by items:

(1) a list of significant measurements (l_i of them) for the class ω_i.
(2) possible values of these measurements for class ω_i.
(3) any information about $p(\mathbf{x} \mid \omega_i)$.

If a class-measurement relationship is stored, then by addressing that class all items ((1), (2), and (3)) can be retrieved. By addressing a measurement, the class having that measurement as a significant measurement can be retrieved (this is an association operation); should more than one class have that measurement as a significant measurement, then all such classes are retrieved.

Retrieval of a list of significant measurements for a specified class or all classes having a specified significant measurement or set of significant measurements is very basic to the *consulting* part of our theory. This has application for providing advice to physicians about *drug interactions, diseases to consider in the differential diagnosis, measurements to take next, abnormal measurement values for a particular class,* and *information about* $p(\mathbf{x} \mid \omega)$ *for class* ω.

Theoretically, given the differential diagnosis (all diseases under consideration), all necessary significant measurement values, and the $p(\mathbf{x} \mid \omega)$ for those diseases, then a minimum probability of error decision can be made. The physician usually makes decisions, however, without $p(\mathbf{x} \mid \omega)$ for the diseases in his differential diagnosis. For this reason $p(\mathbf{x} \mid \omega)$ is not currently known for most diseases although there is

* This work supported by the National Science Foundation, Grant GJ-1099 and the Regenstrief Institute for Health Care.
** Also at School of Medicine, Indiana University.

much a priori knowledge available. The physician makes a decision using rules or criteria which approximate the minimum probability of error method. Sometimes the physician uses a decision boundary in the measurement space for separating one disease from others. This brings up the fact that it may be necessary to store criteria or rules or decision boundaries for a particular disease in differential diagnosis if the system is to interact with today's physicians—at least until they get used to working with $p(\mathbf{x} \mid \omega)$.

The *a priori class probabilities* $P_i = p(\omega_i)$, $i = 1, 2, \ldots, M$ must also be stored and associated with the respective classes. Provision must be made for dimensionality reduction or feature extraction by defining a *feature as a function of certain measurements*. Also provision must be made for allowing *features themselves to be classes* or *classes themselves to be features*.

Past research in applying theory to consulting and decision making has not considered the total problem. A review of the literature will show that the following kinds of difficulties are present in the approaches taken:

(a) The measurements $x_1, x_2, \ldots x_L$ are considered to be statistically independent features when in fact they are measurements which are statistically dependent.

The often made assumption that $p(\mathbf{x}|\omega_i) = \prod_{j=1}^{L} p(x_j|\omega_i)$

is not correct.

(b) There are failures to recognize insignificant features for a particular class.

(c) Lack of ability to proceed when some of the significant measurements for a class are not available.

(d) Lack of ability for interaction and the recognition of the importance of the consulting function.

(e) Lack of ability to introduce a priori knowledge about correlation among measurements or features.

In many ways the consulting part of the system is like making available a computer stored textbook of medicine. An advantage of the consulting part of the system over a conventional textbook of medicine is that, because of the class-feature relationship format, association can be made quickly which are not convenient when using a conventional textbook. Also, the class-feature relationship format provides for an organized collection of training samples so that class conditional probability densities $p(\mathbf{x} \mid \omega_i)$ can be computed.

The class-feature relationship format is basic to minimum probability of error (special case of minimum risk) decision making. A user of the system may choose to make the decision, utilizing interactively the computer's computation of the a posteriori class probabilities for the classes in the differential diagnosis.

COMPLEX FEATURES AND FEATURE EXTRACTION

Features or complex features

In the preceding section a measurement vector $\mathbf{x} = [x_1, x_2, \ldots x_L]$ was defined and a feature was said to be a function of measurements. Now a feature vector $\mathbf{y} = [y_1, y_2, \ldots yl]$ is defined; let

$$y_j = f_j(\mathbf{x}^j, \mathbf{p}^j) \tag{2.1}$$

where

y_j is the jth feature (or complex feature),
f_j is the functional relationship,
$\mathbf{x}_j = [x_1{}^j, x_2{}^j, \ldots, x_{lj}{}^j]$ is the vector of measurements (or features) used to form y_j, and $\mathbf{p}^j$ is a user supplied vector of parameters.

Sometimes Eq. 2.1 will be applied to a set of measurements $\mathbf{x}^j$ to form a feature y_j, for different values of j. Then the resulting features y_j are operated on a second time by a function with form (2.1) to form a new set of features which we call *complex features*. Thus measurements $x_1, x_2, \ldots, x_L$ are the first attributes taken but they are not the properties which characterize a disease or differentiate diseases.

The choice of y_j, $\mathbf{x}^j$, f_j, and $\mathbf{p}^j$ depend heavily upon the nature of the problem to be solved. Depending upon what is useful, y_j and $x_k{}^j$ may be binary, multiply quantized discrete, or continuous variables. The function f_j may be linear or non-linear as required. Deterministic examples of f_1 are:

$$f_1 = 1/3(x_1{}^1 + x_2{}^1 + x_3{}^1)\ , \tag{2.2a}$$

$$f_1 = x_1{}^j + \ln(x_2{}^j) + \exp(x_3{}^j)\ , \text{ and} \tag{2.2b}$$

$$f_1 = \begin{cases} 1 \text{ if } (x_1{}^1 > p_1{}^1 \cap x_2{}^1 < p_2{}^1) \cup x_3{}^1 = p_3{}^1 \\ 0 \text{ otherwise} \end{cases} \tag{2.2c}$$

A feature f_1 can be an estimated probability such as

$$f_1 = \text{prôb}\ (x_2{}^1 > p_1{}^1, \cap\ x_3{}^1 > p_2{}^1) \tag{2.2d}$$

where prôb is an estimate obtained using training samples.

It is clear that the y_j's formed in the above manner may be further combined to obtain a complex feature y_{l+1} illustrated in Figure 1.

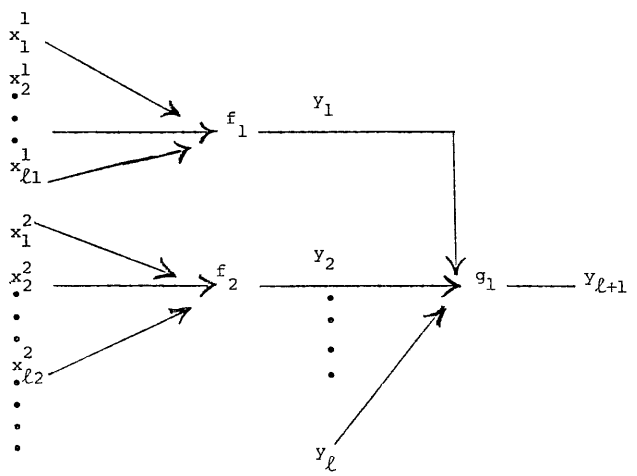


Figure 1—Obtaining features from measurements and a complex feature from features

Example of a complex feature

The complex feature y_{l+1} may be the binary variable, "does the patient have a paracardial friction rub?" Binary features $y_1, y_2, \ldots, y_l$ may be "are causes of a paracardial friction rub present?" or "are effects of a paracardial friction rub present?" For example, the feature y_1 may be the binary feature "is systemic infection present?" which is determined by measurements x_1^1 = sedimentation rate, x_2^1 = temperature, x_3^1 = headache, x_4^1 = ASO titer, etc.

If a researcher tried to apply Bayes theorem directly to the measurements $x_1, x_2, \ldots x_L$ assuming that they are statistically independent (as some researchers have), he would be making a grave error. Considerable a priori medical knowledge goes into obtaining the complex feature y_{l+1} from the measurements. In a differential diagnosis which includes rheumatic fever, for example, the physician wants to know if the complex feature y_{l+1} = paracardial friction rub is present. Several measurements including chest sounds are taken, but the physician is interested in the complex feature y_{l+1}.

Have you ever heard the argument that a physician uses intuition and he can't even explain how he makes a diagnosis? One explanation of this is that he is conscious of complex features like the above paracardial friction rub and he doesn't care to recall the training which showed him how to obtain the features.

The conclusion is that attention should be given to obtaining $p(\mathbf{y} \mid \omega_i)$ using training samples after the a priori medical knowledge is used to form a vector of complex features $\mathbf{y}$ from the measurements, rather than trying to obtain $p(\mathbf{x} \mid \omega_i)$ using training samples.

R features vs. 1 feature with R values

A problem in feature definition for medical diagnosis related to the procedures in previous vectors is illustrated as follows: suppose we are interested in the differential diagnosis of thyroid disease and tentatively list as binary measurements,

x_1 = "is the patient's skin dry?"

x_2 = "is the patient's skin wet?"

Should a single feature y_1 be formed with three values, normal skin, dry skin, and wet skin?

Theoretically if the class conditional densities of $\mathbf{x}$ are known (with $\mathbf{x}$ including x_1 and x_2) optimum performance can be achieved. A problem arises, however, when there is a relatively small sample size to estimate the class conditional densities.

To formalize the problem, let there be R binary measurements $x_1, x_2, \ldots, x_R$ which are mutually exclusive. Define a feature y_1 as

$$y_1 = j \text{ if } x_j = 1 \text{ and } x_k = 0 \text{ for } k \neq j \qquad (2.3)$$

We conclude without further proof that it is desirable for the purpose of density estimation to combine R binary measurements into a single R valued feature if the measurements are mutually exclusive and can be ordered.

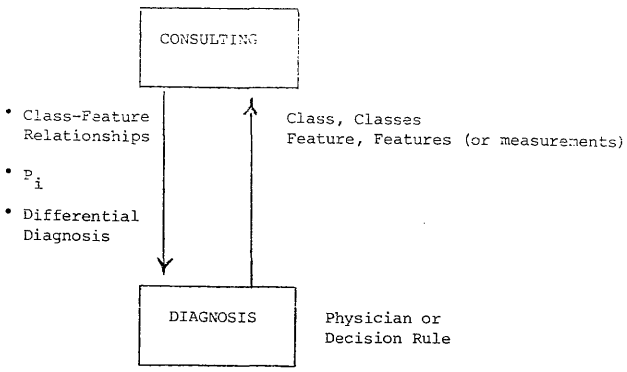


Figure 2—Simplified model of consulting and diagnosis

When is a feature a class?

Is hypertension a feature or a class (disease)? It can be both, depending on the problem. First note that y_1 = "hypertension?" is a feature determined by measurements x_1 = weight, x_2 = age, x_3 = systolic blood pressure, x_4 = diastolic blood pressure, and x_5 = "supine or standing?" The feature y_1 = "hypertension?" may be part of the class-feature relationship of classes (diseases) such as ω_1 = adrenal cortical overactivity, ω_2 = adrenal medullary overactivity, ω_3 = pregnancy, ω_4 = coarctation of the aorta, or ω_5 = renal causes. On the other hand hypertension may itself be a class.

If hypertension is to be a class then one must be able to bring it into the *differential diagnosis*. According to current medical practice, the differential diagnosis including hypertension may be simple with only two classes: hypertension present, hypertension not present. A more complex differential diagnosis may include classes corresponding to degrees of hypertension. These degrees of hypertension can be defined using a complex function such as (2.1).

CONSULTING AND DIAGNOSIS

Introduction

In subsequent subsections consulting and diagnosis will be defined. For now, consider the model shown in Figure 2, where a physician or decision rule is aware of a class or classes in the differential diagnosis, or is aware of an abnormal feature or features. These classes or features are given to the consulting mode which, through *associations*, provides the physician or decision rule with the class-feature relationships for the classes, other classes in a differential diagnosis, and a priori class probabilities.

This model closely approximates the process used by a physician because we believe that this is the best way to get him to use the resulting system. We do not mean best for political or emotional reasons, but best because he is an essential captain of health care delivery and is more likely to use a system which he understands. This does not imply that a better system could not be devised if he were willing to be educated differently. The constraints giving birth to the

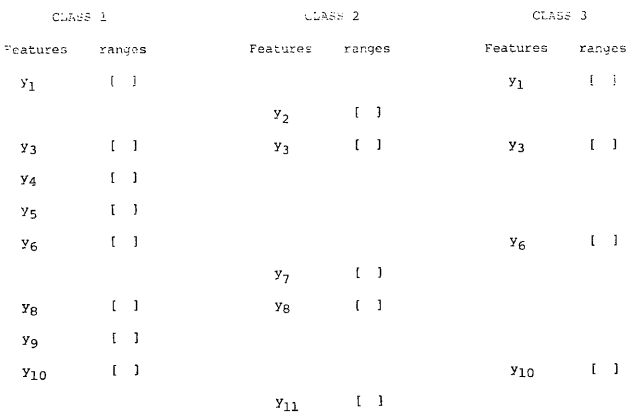


Figure 3(a)—Class-feature relationship structure for a three class example

system are integral in having a health care captain and physician-patient interaction.

There are several approaches which have been taken to computer assisted diagnosis. One approach, or part of an approach, is Bayes theorem stated as

$$p(i \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid i)P_i}{p(\mathbf{y})} \tag{3.1}$$

This reflects two things a physician does. One, he increases the a posteriori probability of disease if $p(\mathbf{y} \mid i)$ is significant or for large P_i. Second, and very important, is information about $p(\mathbf{y} \mid i)$ through consulting usually is in the form* of $p(y_1 \mid i), p(y_2 \mid i), \ldots p(y_l \mid i)$. It is for these two reasons that Bayes theorem is so integral and so important to diagnosis. Usually, the physician does not calculate $p(i \mid \mathbf{y})$; rather he takes the results $p(y \mid i)$ from the respective features and, using a logical decision tree, ranks the respective a posteriori probabilities for the classes in his differential diagnosis.

Bayes theorem requires $p(\mathbf{y} \mid i)$, the construction of which is fundamental to Bayes framework. Construction of $p(\mathbf{y} \mid i)$ has been attempted using various models[1] such as a multinomial model, a Gaussian model, and distribution free models. Although theoretically one of the main objectives in Bayes framework is the insertion of a priori knowledge into $p(\mathbf{y} \mid i)$, and this is optimal, implementation has not been achieved to date.[2] A suboptimum solution is the previously described approach used by the physician. The physician's approach being suboptimal mathematically, is less complex and this may be why he can implement it.

Physician's approach

Here, the physician's approach to diagnosis will be described as closely as possible with mathematics. First, a physician after examining a patient goes into the consulting phase retrieving classes to be part of the differential diagnosis.

* Theoretically he should have the joint class conditional density $p(\mathbf{y}\ i)$.

Associated with each class are features* for that class as shown in Figure 3a, for a three class example. The features and their ranges for a class are part of the class-feature relationship for that class. Listing a feature for a class means that it is a *significant feature*$_i$ if a feature is not listed for a class, then it can have any value insofar as that class is concerned.

Ideally, the class-feature relationship for class i is the class conditional probability distribution $p(\mathbf{y} \mid i)$, which is a multivariate distribution. The physician, of course, does not have the ability to know $p(\mathbf{y} \mid i)$ in general; he approximates it.

One way he approximates $p(\mathbf{y} \mid i)$ is to seek $p(y_1 \mid i)$, $p(y_2 \mid i), \ldots$, for the respective features while in the *consulting mode.* For example, if class i is acute rheumatic fever, he recalls that if y_j=ASO titer, then $p(y_j$ is elevated $\mid$ acute rheumatic fever$) = a$. In Figure 3b, the approximated class conditional probability densities (assuming independent features) are shown for the three class example.

If a physician does not remember information in Figure 3b, he obtains it through discussion with other physicians or from medical literature by going to the *consulting mode.* Already he obtained the names of the three classes by going to the consulting mode.

With knowledge of the information in Figure 3a and Figure 3b, the physician has a differential diagnosis, but he may not have all the features required for this differential diagnosis (there remain lab tests to be ordered). By examining information in Figure 3a and 3b he selects additional features to be extracted (usually from tests). To do this he does not utilize "optimal theory" from sequential feature selection. Rather, he selects any feature y which has a p(y|i) significantly different from that for other classes.

Unfortunately when going to the diagnosis mode, the physician does not have the information in Figure 3b. This

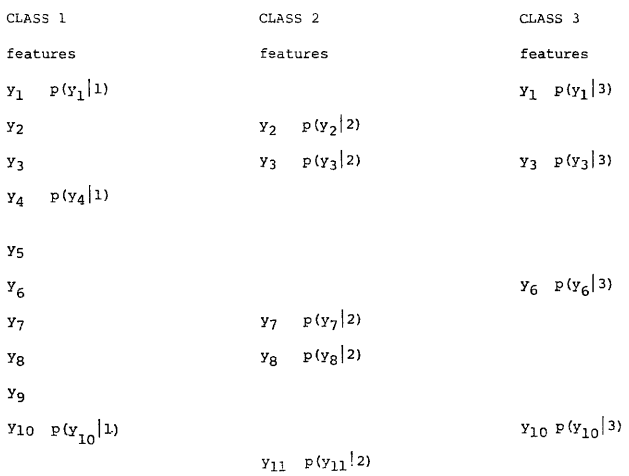


Figure 3(b)—Approximate class-conditional probability densitires as part of class-feature relationships

* We will assume that features have been formed and will not refer to measurements.

is because until an organized approach to patient data collection is instituted, the class conditional probability densities will not be available. What the physician could do at this point is to compute (assuming $y_1, y_2, \ldots, y_L$ statistically independent)

$$p(i|\mathbf{y}) = \frac{\prod_{s=1}^{11} p(y_s|i)}{\sum_{i=1}^{3} \prod_{s=1}^{11} p(y_s|i)} \tag{3.2}$$

for each class i, using the patient's feature vector **y**. Note that only three features are significant for class 1. Therefore, the following simplified computation of $p(1|\mathbf{y})$ can be made:

$$p(1|\mathbf{y}) = \frac{p(y_1|1)\, p(y_4|1)\, p(y_{10}|1)}{p(y_1|1)p(y_4|1)p(y_{10}|1) + p(y_1|2)p(y_4|2)p(y_{10}|2) + p(y_1|3)p(y_4|3)p(y_{10}|3)} \tag{3.3}$$

The physician only approximates eq. (3.3) using logical rules such as counting the number of "positive features" (signs, symptoms, and lab tests) for each class. An example of a logical rule is Eq. (2.1) or more specifically Eq. (2.2d). For example, in place of computing (3.3), the physician computes a complex feature as in (2.2d) and this complex feature indicates the probability the class concerned is active.

The use of a complex feature like (2.2d) may be superior to (3.2) or (3.3) because it includes correlation information among the features. Theoretically $p(i|\mathbf{y})$ should be calculated according to (3.1) rather than (3.2); but use of a complex feature may be a better approximation to (3.1) than eq. (3.2).

Now then, if this system of consultation plus diagnosis is implemented, would a physician use it? He would use the consulting part if he is practicing in a medical center because there he sees the serious diseases. The problem with using the diagnosis part is that we may have failed to include all the features in the class-feature relationships. For example, ASO titer is a feature used to indicate an acute strep infection, important to the diagnosis of acute rheumatic fever. One measurement of ASO may be insufficient; rather the feature should indicate if ASO is high and increasing and how high. It is for this reason, *failure to incorporate features*, that a physician will not use the system. Therefore, feature definition may be a more serious problem than is the problem of how to use more and more powerful techniques to obtain $p(\mathbf{y}|i)$ as a multivariate class conditional probability density.

EXAMPLE—Rheumatic Fever Diagnosis

Introduction

In this section we use the problem of diagnosing rheumatic fever to illustrate many of the properties shown in the previous sections. We will reformulate the way physicians usually diagnose rheumatic fever utilizing these properties.

Rheumatic fever is a serious disease with peak onset in early childhood which can be difficult to diagnose. An important approach to diagnosis of rheumatic fever utilizes the "Jones criteria."[3] First we will show the measurements and features involved in the Jones criteria.

Features and measurements for rheumatic fever

The Jones criteria has "major", "minor", and "supporting evidence" for rheumatic fever. According to the previous sections, the majors are features and the minors and supporting evidence are measurements. First we list the evidence as in the Jones criteria as measurements, keeping in mind that some of these measurements are features:

Major Evidence (features)
y_1 Carditis
y_2 Polyarthritis
y_3 Chorea
y_4 Erythema marginatum
y_5 Subcutaneous nodules

Minor Evidence (measurements)
x_1 Fever
x_2 Arthralgia
x_3 Previous rheumatic fever or rheumatic heart disease
x_4 Increased erythrocyte sedimentation rate or C-reactive protein
Leukocytosis
x_5 Prolonged P-R interval
x_6 Recent scarlet fever
x_7 Throat culture positive group A streptococci
x_8 ↑ ASO titer

Supporting Evidence (measurements)
x_9 History recent sore throat
x_{10} Family history rheumatic fever
x_{11} Abdominal pain
x_{12} Epistaxis
x_{13} Tachycardia
x_{14} Rheumatic pneumonia
x_{15} Pallor and anemia
x_{16} Precordial pain
x_{17} Weight loss
x_{18} Malaise

The Jones criteria is that rheumatic fever has significant a posteriori probability if there are two majors or one major and two minors, while supporting evidence increases the probability. Thus, the Jones criteria is in fact just a complex feature like (2.1), or more specifically (2.2d).

When considering rheumatic fever, the differential diagnosis includes the following classes: myocarditis (toxic, viral, parasitic and fungal), rheumatic carditis, endocrine

disorders, blood diseases (anemias), heart tumor, acute endocarditis, and subacute bacterial endocarditis.

Basically, it appears that only the majors, $y_1, \ldots, y_5$, are features and the other evidence constitute measurements used to determine features. Examples of the measurements used to determine the features are as follows:

y_1 Carditis $= f(x_1, x_3, x_4, x_5)$
y_2 Polyarthritis $= f(x_2, x_{13}, x_{15}, x_{16})$

We would add one additional major, evidence of a recent strep infection, denoted y_6, and there are many measurements associated with this feature:

y_6 Evidence recent strep infection $= f(x_4, x_6, x_7, x_8, x_9)$. Other measurements serve to affect the a priori probability of rheumatic fever as follows:

Increase a priori probability of rheumatic fever $= f(x_3, x_{10})$

Summary

In summary, using a priori medical knowledge to reduce dimensionality results in the following features:

y_1 = Carditis
y_2 = Polyarthritis
y_3 = Chorea
y_4 = Erythema marginatum
y_5 = Subcutaneous nodules
y_6 = Evidence of recent strep infection

Furthermore, the a priori probability of rheumatic fever is increased by $f(x_3, x_{10})$.

We thus have reduced dimensionality from 23 measurements or features to 6 features and have introduced the a priori class probability as a function of measurements. Dimensionality reduction has been achieved by recognizing that features are functions of measurements. The quality of this dimensionality reduction needs to be investigated both experimentally and theoretically.

Decision making

Computation of $p(i|\mathbf{y})$ by $p(i|\mathbf{y}) = \frac{p(\mathbf{y}|i)Pi}{p(\mathbf{y})}$ requires $p(\mathbf{y})$ which is not available unless $p(\mathbf{y}|i)$ is available for all diseases i in the differential diagnosis, because $p(\mathbf{y}) = \sum_{i=1}^{M} p(\mathbf{y}|i)Pi$ where M is the number of diseases. The Jones criteria avoids this problem because rather than calculating $p(i|\mathbf{y})$, it *establishes a decision boundary* such that on one side are all samples of rheumatic fever.

Remaining work

The next step is to form the joint probability distribution of $\mathbf{y} = [y_1, y_2, y_3, y_4, y_5, y_6]$ using training samples for rheumatic fever and training samples for other diseases in the differential diagnosis. A study of overlap in these two distributions is important, as are the estimated distributions $p(\mathbf{y}|i)$ for rheumatic fever.

CONSULTING—STORAGE, ASSOCIATION AND RETRIEVAL

The consulting mode is concerned with providing information upon request. Information to be retrieved is as follows:

(1) Features of a class (signs, symptoms, lab tests)
(2) Classes for which a feature is significant
(3) Treatments for a class
(4) Differential diagnosis given a class
(5) Differential diagnosis given a set of features
(6) A priori class probabilities P_i
(7) Class-conditional probability densities

In a previous section the class-feature relationship involved the class, the features of the class ((1) above), and the class-conditional probability density ((7) above). Two ways to store and retrieve the above information are the hierarchical approach[4,5] and the networking approach.[6,7,8] The networking approach appears to offer the advantage of economical storage, easier retrieval and is a more general structure.

For the purpose of updating, the concept of the class-feature relationship must be preserved. The networking approach will preserve these relationships while making available associative retrieval as illustrated in Figure 4. In this illustration

rheumatic fever→ASO titer

means the feature ASO titer is associated with the class rheumatic fever, while

ASO titer→rheumatic fever

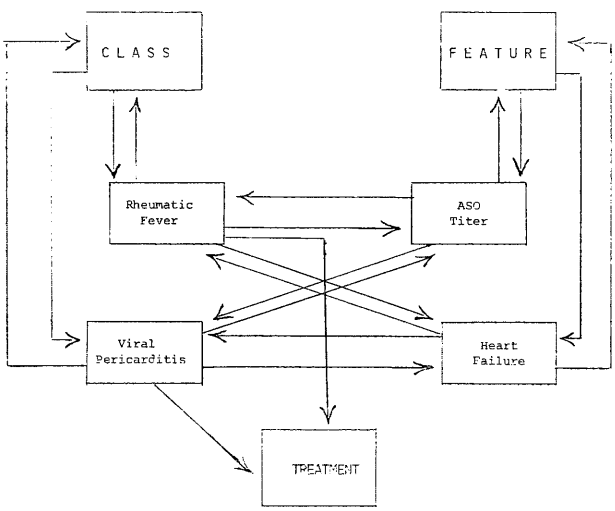


Figure 4 Illustration of networking

means that the feature ASO titer is associated with the class rheumatic fever.

CONCLUSION

The approach presented has emphasized the need to incorporate a priori knowledge into the decision process. Some ways of doing this are the use of class significant complex features; the use of density estimation techniques which utilize "people samples" in the measurement space, feature space, and the relationship space; and the introduction of the consulting mode.

REFERENCES

1. Patrick, E. A., *Fundamentals of Pattern Recognition*, Prentice-Hall, Inc., Englewood Cliffs, 1972.
2. Patrick, E. A., et al., *Computer and Pattern Recognition For Large Systems*, Purdue University, Dept. of Electrical Engineering Report TR-EE 72-27, Sept. 1972.
3. Nelson, W. E., Vaughon, V. C., Mc Kay, R. J., *Textbook of Pediatrics*, W. B. Saunders Co., Philadelphia, 1969, pp. 538-539.
4. Gorry, G. A., "Strategies For Computer-Aided Diagnosis," *Mathematical Biosciences*, Vol. 2, 1968, pp. 293-318.
5. Bleich, H. L., "The Computer As A Consultant," *The New England Journal of Medicine*, Vol. 284, No. 3, Jan. 21, 1971, pp. 141-147.
6. Patrick, E. A., et al., *Interactive Computing And Pattern Recognition With Application to Health Care Delivery*, Purdue University, Dept. of Electrical Engineering Report TR-EE 72-25, Sept. 1972, pp. 23-32.
7. Su, S. Y. W., "Managing Semantic Data In An Associative Net," *Proc. Symp. on Information Storage and Retrieval*, J. Minsky and S. Rosenfeld (eds.), Association for Computer Machinery, New York, 1971, pp. 105-116.
8. Isner, D. W., "An Inferential Processor For Interacting With Biomedical Data Using Restricted Natural Language," *Proc. Spring Joint Computer Conference*, 1972, pp. 1107-1124.

Pattern recognition with interactive computing for a half dozen clinical applications of health care delivery

by ROBERT S. LEDLEY

National Biomedical Research Foundation
Georgetown University Medical Center
Washington, D. C.

THE NBR CLINICAL-IMAGE PATTERN-RECOGNITION LABORATORY

The clinical applications to be described make use of the NBR clinical-image pattern-recognition laboratory, which consists of (see Figure 1)

(1) FIDAC (Film Input to Digital Automatic Computer), a high-resolution, high-speed, on-line flying-spot scanner;
(2) MACDAC (MAn Machine Communication with Digital Automatic Computer), one of the first inexpensive interactive display instruments based on the storage-tube principle;
(3) the VIDIAC (Vidicon Input to Automatic Computer), a unique vidicon scanning instrument that utilizes a modern image-converter tube for enabling inputting video data into the computer;
(4) SPIDAC (SPecimen Input to Digital Automatic Computer), an automatic microscope with instantaneous automatic focusing, and with a computer-controlled x-y stage that can scan a 1 cm $\times$ 1 cm area in 2.5μ steps in about five minutes; and
(5) DRIDAC (DRum Input to Digital Automatic Computer), a drum scanner that utilizes two read heads simultaneously to cut the scan time in half.

Accessory instrumentation includes

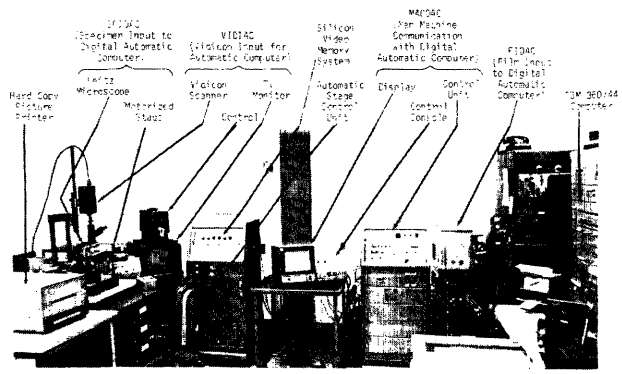
(6) the silicon-video memory (that utilizes the image-converter tube) and
(7) the computer-interface control unit.

All of these instruments are controlled by the Foundation's IBM 360/44 computer, in a unique, integrated, pattern-recognition hardware system with capabilities, to our knowledge, duplicated nowhere else.

Intimately associated with this hardware capability is an extensive software capability. This software capability includes the following systems:

(1) FIDACSYS (The FIDAC System), which inputs the images into the computer from the scanning devices, and accomplishes overall picture manipulation and enhancement;
(2) SYNTAXSYS (SYNTAX System), which is a pattern-recognition language that uses an approach based on our original research on picture grammars;
(3) BUGSYS, which is a picture-processing and measuring language originated by us that utilizes conceptual pointers for picture analysis and manipulation;
(4) MACDACSYS (The MACDAC System), which implements the interactive capabilities of the MACDAC unit, including computer-interrupt features for accepting information from and displaying pictorial and alphanumeric information on the MACDAC unit;
(5) SPIDACSYS (The SPIDAC System), which automatically directs the motion of the SPIDAC microscope stage, detects good chromosome spreads or other features on the glass microscope slide, records the location of such features (to the nearest 1.25μ), and directs the vidicon scan into the computer;
(6) DOCSYS (Display Of Chromosome Statistics), which is a programming system that enables on-line computer-console interaction with the disk memories of the computer for the statistical evaluation and display of large masses of data in a file; and
(7) REMOTE, a programming system that enables a remote user to be serviced by the computer, especially for the investigation of large files stored on the computer's disk systems.

The application of these hardware and software capabilities to interactive computing for health-care delivery falls into four main categories. First, there are aids to the interpretation of very important noninvasive diagnostic techniques, such as thermography and echocardiography; second there is the analysis of medical time-motion studies of body systems used in diagnosis, such as cineangiographs and fluorescence retinographs; third there is the evaluation of images formed by the differential radiation absorption or distribution properties of body structures, such as in the evaluation of roentgenographs or scintillation scans; and finally there is the investigation of images formed by the optical microscope, such as occur in the clinical bacteriology or hematology



National Biomedical Research Foundation Pattern Recognition Laboratory Showing SPIDAC, FIDAC, MACDAC, FIDAC and the IBM 360/44 (DRIDAC not shown)

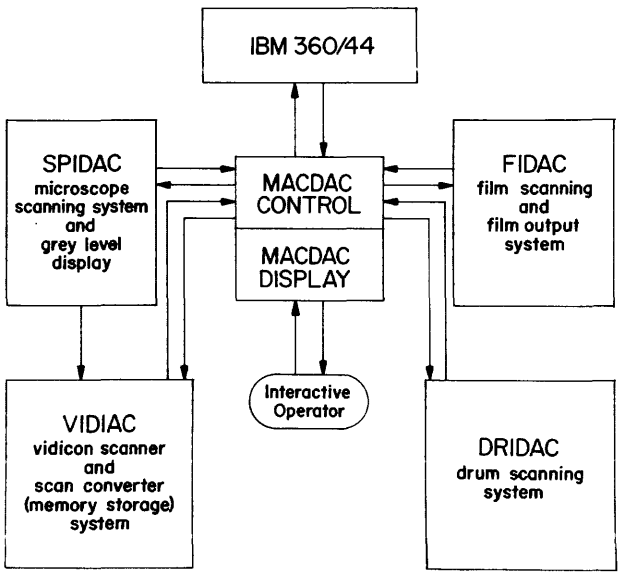


Figure 1—The National Biomedical Research Foundation Clinical-Image Pattern-Recognition Laboratory. *a.* The instrumentation layout. *b.* Block diagram of instrumentation organization

laboratories. In this paper we shall consider six such applications, namely in the fields of thermography, echocardiography, cineangiography, fluorescence retinal cinematography, radiology of diffuse lung lesions, and the analysis of optical images of bacteria.

THERMOGRAPHY

Thermography is the thermal mapping of areas of the human body. Infrared sensing devices perceive and collect the skin's emitted energy, relate it to a reference black body, and transform it into an electrical signal. The sensitive detectors used in this work are an outgrowth of military reconnaissance systems. When these systems were declassified in 1956, they opened a new medical diagnostic field.

The first medical application of thermography was to diseases of the breast.[1] Following this, a number of investigators began to explore the diagnostic possibilities of infrared sensing devices. The early work of Wood[2] correlated thermographic findings with disease of the carotid complex. The ensuing years have seen a number of reports on the application of thermography to the fields of cerebrovascular disease,[3] oncology,[4] orthopedics,[5] obstetrics,[6] the diagnosis and management of the burn patient,[7] and cardiovascular disease,[8] to name but a few.

Figure 2 shows a normal and an abnormal thermograph. In the abnormal thermograph, the forehead, which is supplied by a branch of the internal carotid artery, is "cool" while the cheek, supplied by the external carotid, is "hot," owing to some blockage of the internal carotid thereby indicating the predisposition of the patient to a possible stroke. In Figure 3 we show the result of scanning the thermographs and having the computer make a contour plot of the grey levels. The computer is programmed to determine the average temperature at various locations of the face, thereby quantitating the difference in heat distribution. Quantitative comparisons can be made on thermographs from the same patient at different times to detect progress in the disease state. Such quantitation should assist in making more accurate correlations between the thermograph's appearance and the disease state of the patient. Since the infrared detector itself puts out electronic signals, eventually it should be possible to perform the computer analysis directly from these signals and not from a picture. However, this can only be accomplished after specific results and experience have been obtained in the computer analysis of the pictures.

The first step in our automated analysis is to calibrate the thermograph. In Figure 4, on the left, are imaged grey-level squares that are obtained from a standard temperature box, each square being fixed at a particular known temperature. The FIDAC scans each of these squares and relates the corresponding grey-level value with the proper temperature. Figure 4 is a black-and-white representation of the contours formed by every other grey level.

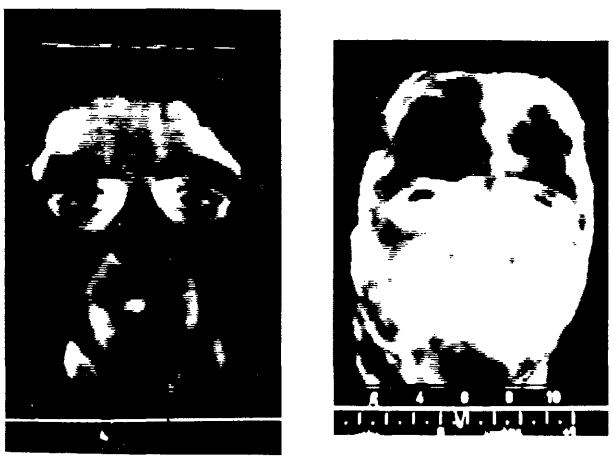


Figure 2—Illustration of a normal (left) and abnormal (right) thermograph

For the application of thermography to detecting diseases of the carotid complex, we must compare the average temperature of a region of the forehead with that of a region of the cheek. In order to locate these areas, we identify on the thermograph three points, namely points on the right and left temple and a point on the bottom of the chin. Small pieces of aluminum foil are pasted on these points on the patient so that they can be seen in the thermograph. Using the MACDAC interactive-graphics console, the computer operator points to these standard locations on the display of the thermograph. The computer then automatically generates a "standard face" through these points. The standard face is a drawing of correct anatomical proportions that has previously been placed into the computer's memory. For a particular individual, it is automatically adjusted so that it will match with the three fiducial points; a projective geometric transformation is made to adjust the standard face to that of the particular patient (see Figure 5).

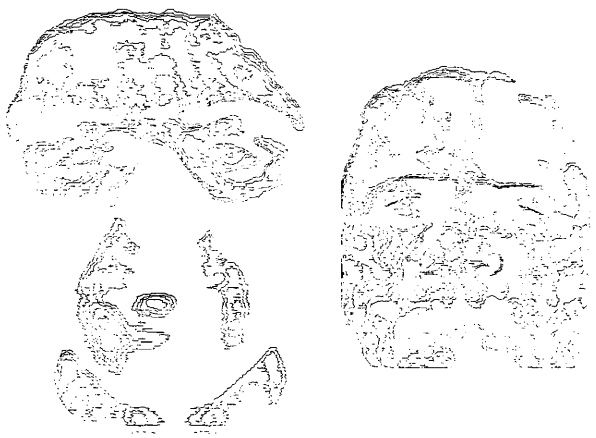


Figure 3—Contour plots of the normal and abnormal thermograms of field 2 as produced by the computer

The computer now selects the proper area of the thermograph for analysis, as shown in Figure 5. The following areas are selected: right forehead, left forehead, total forehead, right cheek, left cheek, lips, and the chin, making seven areas in all. For each of these areas the average temperature is determined. The analysis consists in examining the ratios of the average temperatures of different areas. For one particular normal patient, the results are shown in Figure 5. If the ratio of the average absolute temperatures of area 1 to area 4 or area 2 to area 5 is less than .98, then the patient is considered to be abnormal.

Another important application of thermography is to the detection of breast cancer. A thermogram with some of the calibration squares is shown in Figure 6a. Figure 6b illustrates the standard picture of the breast when the fiducial points are taken as the sternoclavicular notch and the nipples. Figure 6c shows the average temperatures for four quadrants of each breast and the total average temperature of each breast. Temperature asymmetries and other such features are correlated with the early detection of breast cancer.

Figure 4—Illustration of a thermogram with calibration squares on the left

ECHOCARDIOGRAPHY

Ultrasound provides a noninvasive method of recording movement of such intracardiac structures as the mitral valve, left ventricular endocardium and epicardium, left and right ventricular septum, and left atrium.[9] Improved methods for

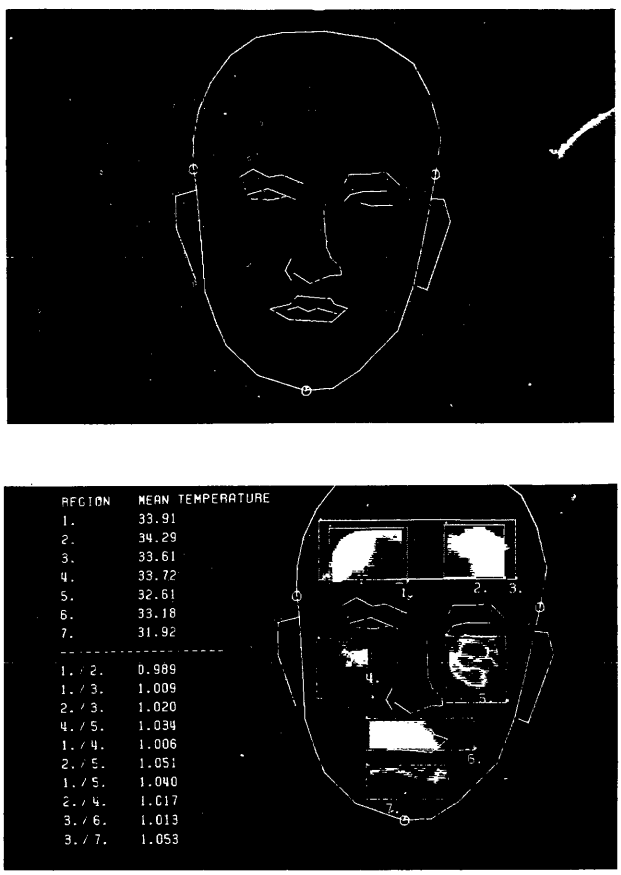
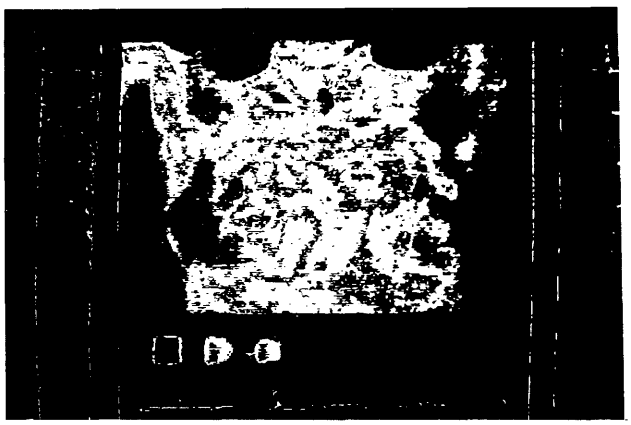
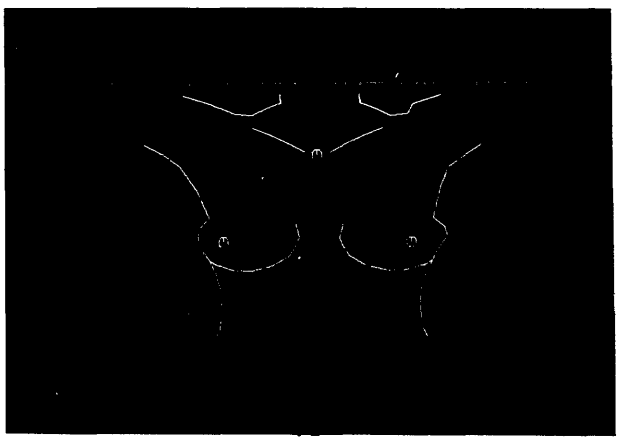


Figure 5—*a*. Standard drawing of face showing the three fiducial points. *b*. Computer analysis of thermogram showing the final results





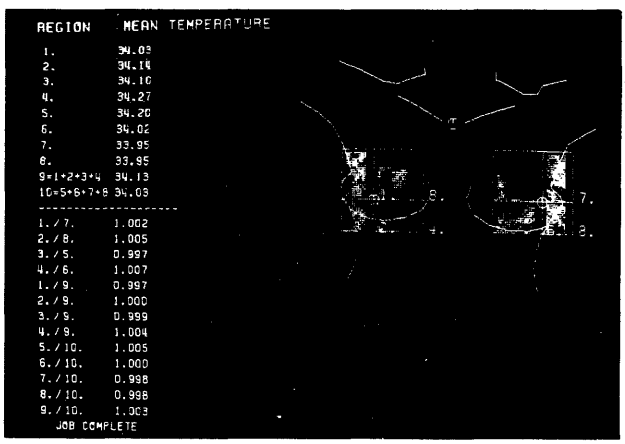


Figure 6—*a.* Computer display of thermogram of the breast. *b.* Standard drawing showing fiducial points. *c.* Results of computing average temperatures for each of the four quadrants of each breast

recognition of these structures and more precise calculation of the amplitude and rate of their movements offer a potentially important means for diagnosing heart disease as well as for monitoring ventricular function, particularly changes in function induced by drugs, physiological maneuvers, or unstable disease. In the discussion given below, application to mitral-valve reflections is first described. Clinical applications of ultrasound to the mitral valve have shown that distinct relationships exist between the ultrasoundcardiograph (echocardiograph) and the pathology of the mitral valve. Our long-term goal is to design special-purpose circuitry that can analyze the echocardiograph on-line, since the original signal is in electronic, not pictorial, form. When this is accomplished, the echocardiogram can be used to assist in monitoring the patient in intensive care units.

In making an ultrasoundcardiograph of the mitral valve, an ultrasound transmitter, which transmits one microsecond bursts of a 2.25 MHz tone 1,000 times per second, is directed at the anterior leaf of the mitral valve through the third or fourth interspace.[10] This beam of sound penetrates the body, and a portion of this beam is reflected whenever it crosses an interface between different structures or tissues. These reflections are gathered and shown on the face of an oscilloscope. Successive bursts of sound are recorded on the oscilloscope, resulting in a graph of mitral valve motion, with time as the vertical axis and distance from the ultrasound transmitter as the horizontal axis. (This type of display is often referred to as the "B mode.") For manual analysis, photographs are taken of the oscilloscope face, and the data is then obtained from these pictures (see Figure 7a).

The resulting echocardiograph contains the mitral-valve curve which traces the movement of the anterior leaf of the mitral valve. The distinctive curve of a normal heart contrasts sharply with the tracing of a heart with mitral stenosis.

Standard labels are used to designate different portions of the curve. Point A represents the open position of the mitral valve due to atrial contraction. Point B occurs during the rapid closure of the valve to its closed position, at point C. The interval from point C to point D represents the closed portion of the cycle. At point D a rapid opening of the valve occurs, leading to the position of maximum aperture, at point E, during early diastole. Point F represents the final closure of the mitral valve before atrial contraction and point A.

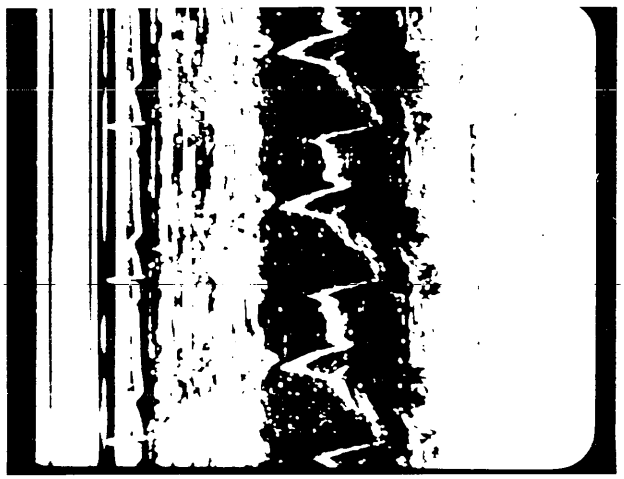
Clinical studies have been able to correlate specific characteristics of the mitral-valve curve with malfunctions of the mitral valve. The most significant of these are as follows:

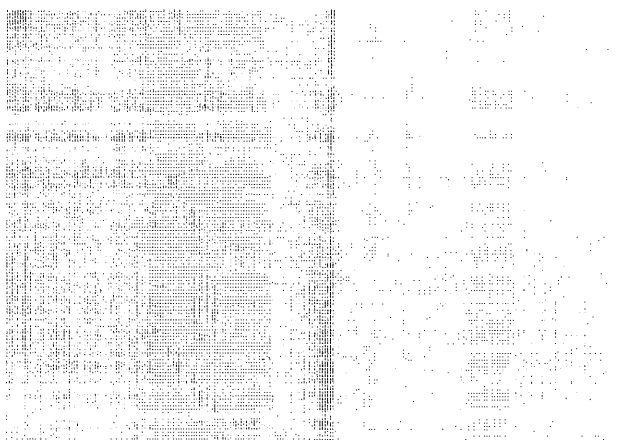
(1) The slope of the mitral-valve curve between points E and F is related to the total area of the mitral valve (or the mitral orifice). This slope has also been correlated with mitral regurgitation.
(2) The total amplitude of the mitral-valve curve (i.e. the distance between points C and E) is related to the degree of calcification and rigidity of the anterior leaf of the mitral valve.
(3) The slope of the curve between points D and E is related to the degree of calcification.
(4) The area of the mitral valve was also related to the elapsed time between points D and E.

These studies also suggest that a statistical diagnosis can be made on the basis of the above-mentioned data.

Several problems arise, however, in obtaining this data. The primary problem in ultrasoundcardiography is that it is often difficult to obtain a clear echocardiograph, owing to such factors as obesity, severe emphysema, previous surgery,

and cardiac displacement. In addition, the mitral-valve curve can often be obscured or confused with echoes from other structures which appear on the echocardiograph. A "dropout" phenomenon, in which the mitral-valve leaf drops temporarily outside the ultrasound beam and thus off of the echocardiograph, is also common. Finally, once an acceptable echocardiograph is obtained, the making of hand measurements from the rough echocardiograph introduces further error into any statistical diagnosis.





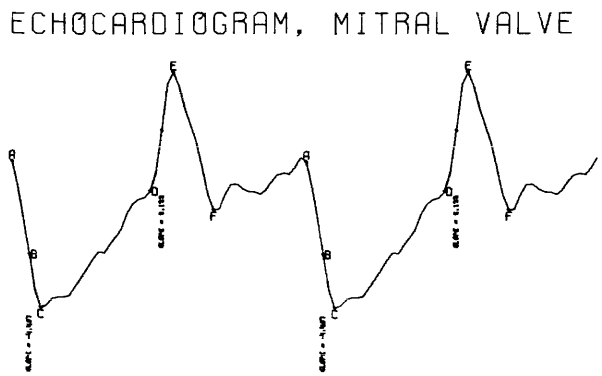


Figure 7—*a*. Original echocardiograph for displaying mitral valve motion. *b*. Binary computer printout showing a partially cleaned results. *c*. Final results of computer analysis

The use of a digital computer for the analysis of the echocardiograph eliminates, or at least minimizes, many of these potential problems. Before the computer can proceed with the analysis of the echocardiograph, now in the form of a photograph, it must be digitized and fed into the computer's memory by the FIDAC. Since the picture is basically black and white, it can be read-in in binary.

The first step of the computer analysis is to clear the echocardiograph of all excess "noise" caused by reflections from objects other than the mitral valve, leaving only the mitral-valve curve. This is the most difficult part of the program, for the computer is unable to "view" the entire picture and isolate the mitral-valve curve, but must instead rely on mathematical features of the curve that set it apart from the noise on the echocardiograph (see Figure 7b).

The program proceeds first to establish more continuity on the often discontinuous curve. To accomplish this, it "looks" at groups of nine spots in a 3×3 array. If any one of the nine spots indicates a reflection, the group is assigned the value of a reflection. What this accomplishes is a shrinking of the echocardiograph from a picture with 664×446 spots to one with 222×149 spots, at the same time smoothing the curve and eliminating discontinuities.

The program then proceeds to go through each horizontal line of the echocardiograph (in the "distance" direction), counting the length (in number of spots) of each reflection segment. If the length of a segment is greater than a certain input parameter, the segment is discarded. This mathematically contrived step also has clinical validity, for the thickness of the anterior leaf of the mitral valve, represented by the thickness of the mitral-valve curve, is small compared to the thickness of the thorax wall, which is found on the echocardiograph as noise.

If the length of the segment is less than the designated parameter, the computer finds the midpoint of the segment and discards all the other points of that segment. This step succeeds in clearing most of the noise from the echocardiograph and results in a curve with a thickness of only one spot.

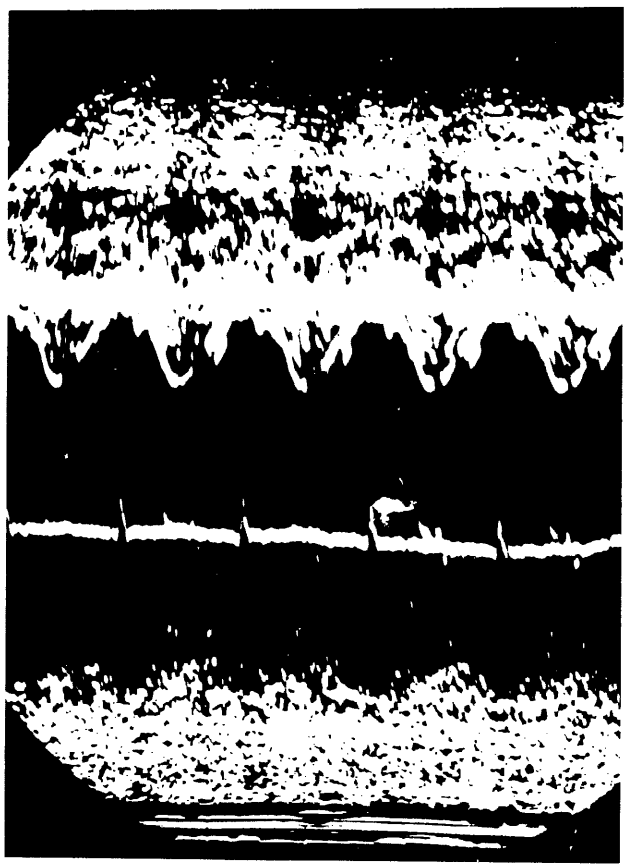
One additional step is required, however, before the computer can completely isolate the mitral-valve curve. In this final step, the computer counts the length of each remaining segment in the time (vertical) direction. This segment need not be completely continuous, the computer program accounting for a certain amount of discontinuity. The computer discards all curves shorter than a second input parameter and keeps only the longest segments, for the mitral-valve curve (while possibly discontinuous) will stretch the length of the echocardiograph. This final step completely isolates the mitral-valve curve and makes possible a statistical analysis of the echocardiograph. Figure 7c shows a computer plot of these results.

The analysis of the echocardiograph is based on the six lettered points described above. While the location of these points is often easily determined by eye, the computer must once more rely on mathematical techniques. Since there are often several cycles of valve motion on each slide, the

computer starts by finding the period of the curve. This it does iteratively by assuming a series of periods for the curve, averaging the points of the curve for each period, and taking the sum of the squares of the deviations of the points from the average curve for each period. The least sum-of-squares deviation gives the best period for the curve. The computer then averages the several cycles of the curve and obtains the graph of an average curve of valve motion.

The six points are then relatively easy to find (see Figure 7c). Point E is defined as the absolute maximum of the averaged curve. Point C is defined as its absolute minimum. Point A is defined as the point of greatest deviation from a straight line drawn between points E and C. Point F is defined as the lowest minimum between points E and A, closest to E. Point D is defined as the point on the curve of greatest deviation from a straight line drawn between points C and E. And finally, point B is defined as the point between points A and C at which the curve has the greatest slope. (The slope is found using the numerical derivative of a sixth-degree polynomial approximation designated by seven points on the curve.)

Once these six points have been located, it is a simple matter to make the measurements described above. In addition, other measurements can be made, such as the area



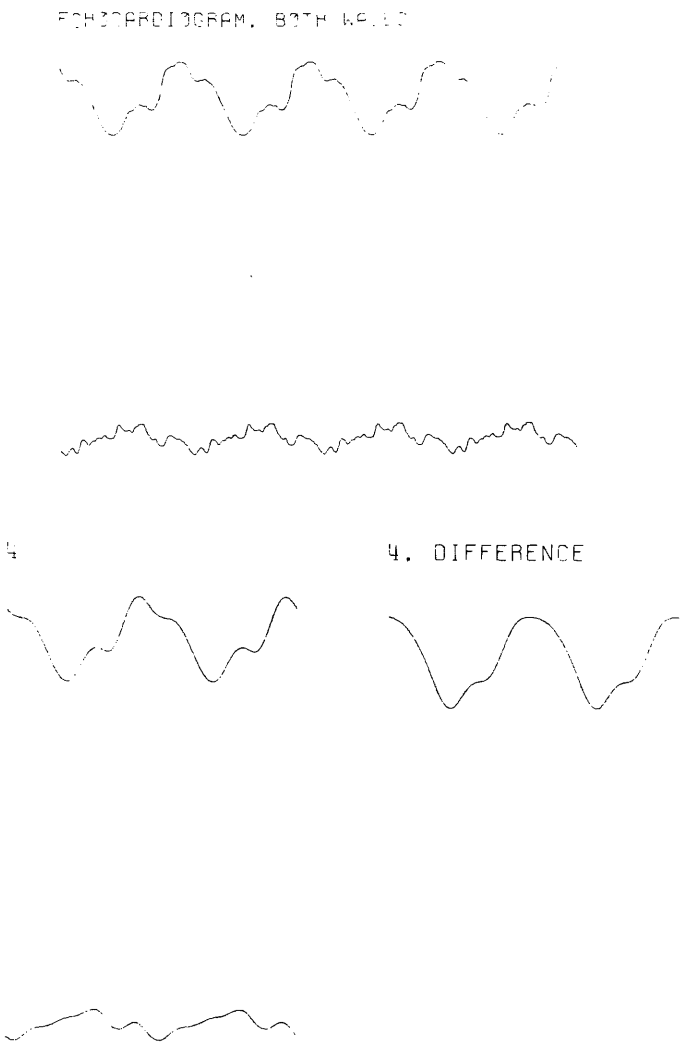


Figure 8—*a*. Original echocardiograph for anterior-posterior dimension of left ventricle. *b*. Computer printout of motion of anterior-posterior walls of left field after cleaning. *c*. Smoothed curve of anterior-posterior wall movement and curve of the difference

under the curve and the standard deviation of the curve, data very difficult to obtain by hand.

Another important parameter that can be obtained from an echocardiograph is the anterior-posterior dimension of the left ventricle. Figure 8a shows the echocardiograph made for determining this dimension with the movement of the anterior and posterior walls of the left ventricle seen in the upper and lower waves. A computerized cleaning process identifies these waves, as shown in 8b. A fourier analysis is made of each wave and a "smoothed" final wave is constructed using the first four coefficients, thereby leaving out higher frequencies which contain noise. The difference between the curves indicates the anterior-posterior dimension as a function of time (see Figure 8c).

CINEANGIOGRAMS

Increasing use is being made of angiocardiographic techniques in the evaluation of myocardial function in man. Although biplane serialographic studies have been successfully employed for the measurement of ventricular end-diastolic and end-systolic volumes, the analysis of four or six films exposed per second makes it difficult to derive much information concerning the rate of change of ventricular and atrial volumes. The work of Dodge et al.[11] and of Bunnell and associates[12] makes it clear that chamber volumes can also be determined with accuracy by a single-plane method. Such approaches would make it feasible to analyze cineangiograms exposed at 60 frames/sec., and in this manner sufficient data would be available to determine the velocity and acceleration of atrial and ventricular filling and emptying, data which are essential for the calculation of ventricular compliance and the velocity of myocardial-fiber shortening. Determination of these variables would ultimately make it feasible to characterize myocardial function in patients with various forms of valvular heart disease and myocardial failure, and to determine the effects of various interventions, such as cardioactive drugs, muscular exercise, etc., on myocardial dynamics.

The time required for the analyses, measurements, and computations of cineangiograms exposed at 60 frames/second would be staggering, unless such determinations are automated. The densitometric scanner-computer system outlined below facilitates such analyses and allows almost instantaneous calculation of atrial and ventricular volumes and rates of change of volumes, and when combined with the appropriate simultaneous measurements of intraventricular pressure and rate of change of intraventricular pressure, permits estimation of ventricular compliance, the velocity of myocardial ejection, the velocity of myocardial-fiber shortening, and the velocity of contractile-element shortening.

Three steps are involved, namely recording the cineangiograms, transferring the pictures into the memory of the computer, and analyzing the pictures on the computer. If the medical instrumentation used allows an image of the pressure curve to be superimposed on each cineangiogram picture, the magnitude and variation of the pressure occurring during the time of exposure of the frame is obtained. Alternatively, this information could be obtained from simultaneous magnetic-tape recording. The cineradiographs are taken at the rate of 60 frames per second. The result produces the projected area of the left ventricle on the film. It has been shown that such one-plane cineangiograms are sufficient to calculate the volume of the chamber.[13,14] The volume is given by

$$V = \frac{D_1^2 \times D_2 \times 0.85 \times \pi}{c\ 6}$$

In order to proceed with the automatic computer analysis of cineangiograms, each frame of the movie film must be successively recorded in the memory of the digital computer. This is accomplished by the FIDAC,[15] which digitizes each frame of the film within 0.3 sec. Figure 9 illustrates a cineangiogram sequence of film frames.

Figure 9—Segment of cineangiogram film strip

A roll of film is processed as follows: It is placed in the film-transport unit of the FIDAC instrument, and, after setting the "frame count" to 1, the computer signals the FIDAC to scan the frame, and within 0.3 sec the picture is in the computer's memory. Next a spectrum for the picture is computed, from which it can be determined whether or not the picture is blank—that is, either all black or all white (or at least 98% black or white). If the picture is blank, the program signals FIDAC to move to the next frame. In this way blank frames, usually leader frames, can be skipped automatically.

Two main "objects" are to be processed in each frame, the projected ventricular area and the instantaneous-pressure curve, if available. A scan is first made to locate the ventricular area on the frame. Various aspects of this area are recognized and processed by the FIDACSYS part of the programming system. Another internally programmed scan is made to locate the pressure curve. This pressure curve can then be processed by the BUGSYS part of the programming system.

One of the most difficult aspects of the automatic analysis of the cineangiogram has been the problem of identifying the contour or "boundary" of the radio-opaque area as seen in the film. A number of techniques have been proposed, but these all require extensive computation time.[16]

The computer analysis technique which we employ is illustrated in Figure 10a. The length of the heart as seen in the cineangiogram does not vary appreciably and runs from the upper left to the lower right of each frame. However, as the heart beats, the width as measured from the upper right to the lower left of the frame will vary. If we make a grey-level contour plot along such a diagonal, we can easily see the region in which the boundary of the heart should be selected. The grey-level profile has two troughs and a central plateau. For the location of the boundary we select that grey level that is 20% of the distance from the corresponding trough to the plateau, as illustrated. The distance between these points along the diagonal gives us the minor axes (the major axes being fixed). The volume can then be computed by the formula. For illustrative purposes, we draw a "standard" heart cineangiogram shape using the projective transformations mentioned above, based on the four points illustrated in Figure 10b. In Figure 10c we have superimposed such drawings of four different frames (although they overlap quite a bit).

Alternative methods of computing the volume are based on the utilization of the area of the projection rather than just

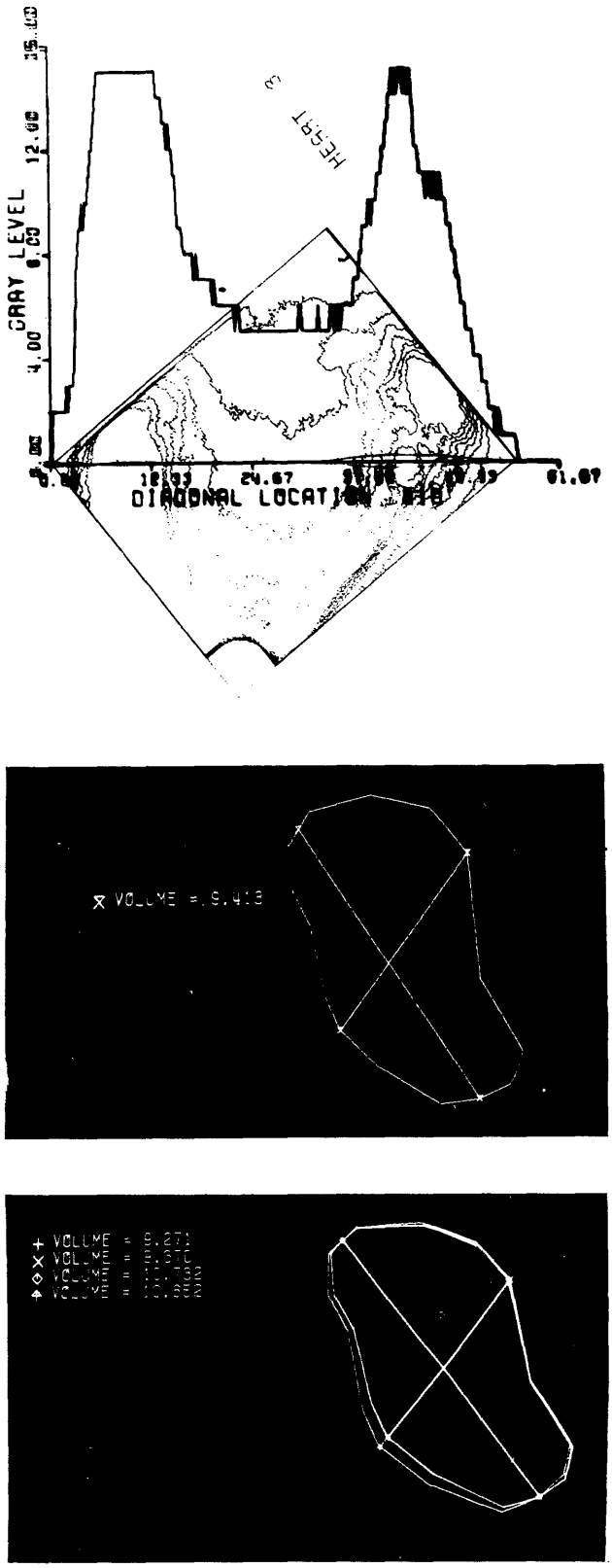


Figure 10—*a*. Method of analysis. *b*. Results from the analysis of one frame. *c*. Superimposition of the results of the analysis from several frames

the two diameters. The area can be determined either by the transformed standard contour as drawn through the four points; or else the boundaries of a cineangiogram can be determined completely by taking many diagonals parallel to the single diagonal mentioned above. The grey-level profile along each of the parallel diagonals will have the same general appearance as that shown in Figure 10a, and for each such, diagonal the boundary points can be determined. In this way an even more accurate estimate of the cineangiogram projection of the heart can be made.

RETINAL-FLUORESCENCE CINEMATOGRAPHY

Retinal-fluorescence cinematography can produce thousands of frames of film for quantitative analysis (see Figure 11). The technique can contribute greatly to the understanding of normal blood flow in the eye and can be used to diagnose ongoing and imminent disease states.[17–20] However, the manual work required for the film analysis is excessively great and hence limits the application of this important method. With the use of automatic picture pattern recognition, a computer can be programmed to extract quantitative information from such movie films rapidly and to correlate the results with the disease process. In our discussion, we shall first describe the equipment used, which takes pictures of the retina at the rate of 20 frames per second; next we shall relate the results to eye disease; and finally we shall discuss briefly the computer pattern-recognition approach.

Our method utilizes a Zeiss fundus camera (modified by Dyonics, Inc.) with a 35-mm Arriflex cinecamera and a constantly cooled special strobe tube.[18] A fully automatic twin injector (Dyonics model 2020) greatly facilitates and standardizes the injection of 5 ml of 10 percent fluorescein through a butterfly needle. Under a pressure of 50 p.s.i. the dye is injected within 0.5 seconds, immediately followed by a 15-50 ml normal saline flush. Synchronously with the injection, an electronic timer is activated. After three to five seconds of preset delay, camera and synchronized strobe start automatically, currently with 19-20 frames per second. A running stop-watch is filmed before or after the procedure to double-check the time factor. Each run lasts for 20 seconds. All these events are set off by one foot switch.

Normal flow patterns and velocities in the retinal vascular system in different age groups are presently being studied in Georgetown University's Ophthamology Laboratory.[21] On the basis of such normal ranges, it will be possible not only to

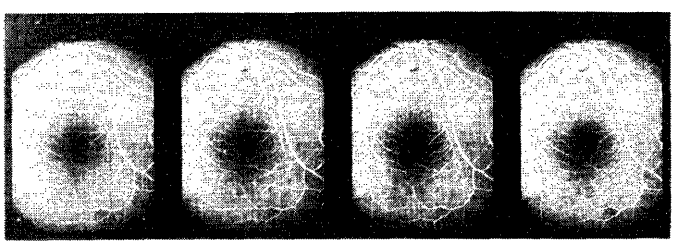
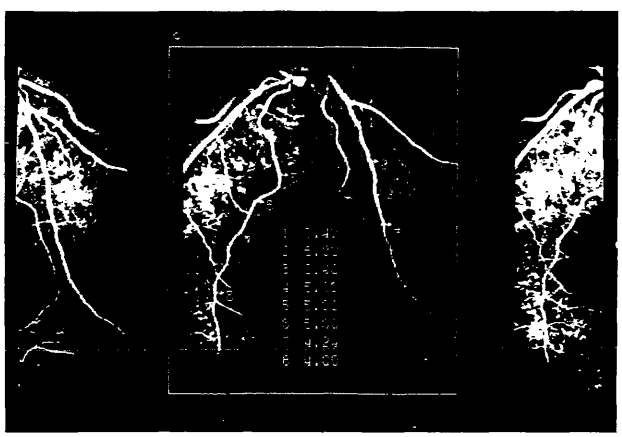


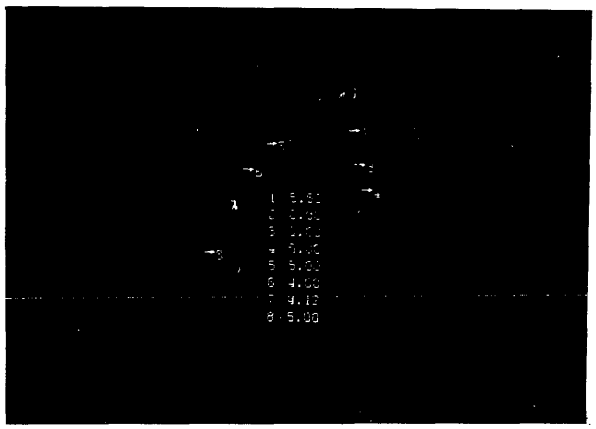
Figure 11—Segments of film strip of retinal fluorescence cinematography

discover and objectively document flow alterations in conditions such as thromboses and pre-thromboses, dysproteinemias and paraproteinemias, diabetes, and anemias, but also to check directly on the efficacy or failure of treatments by anticoagulation, plasmapheresis, etc.

It can be safely said that fundus-fluorescence cineangiography offers three major advantages: first, better chronological detail than rapid-sequence still photography; second, objective and quantitative analysis of flow differences in individual vessels in health and disease; third, its inherent educational value. The method has already taught us to abandon the term and concept of a singular "retinal circulation time." Similarly, the concept of the already almost traditional five "filling phases" as applied to the entire retina will probably also soon have to become more restricted, revised, and qualified.

For automatic pattern recognition of the film, the FIDAC scans successive film frames at the rate of 0.3 sec. per frame. The computer analysis consists in locating the fluorescence vessels, measuring the extent of flow of the fluorescein, measuring the diameters of the vessels, and so forth. This is accomplished on successive frames of the films, and velocities and accelerations of the parameters are calculated. The interactive-graphics unit is utilized to indicate the location of those points at which the diameter of a retinal vessel is to be measured. This interaction need only be done on the first frame of a film sequence. The computer will remember from frame to frame the location of these points and will make any small adjustments that may be required due to minute movements of the patient. The computer is programmed to measure the diameter of a vessel in a direction perpendicular to its length. In Figure 12a we show one frame in which eight different locations have been measured. In Figure 12b we show only the eight different points, for clarity, since they are somewhat obliterated by the retinal vessel image in Figure 12a. We also make a plot of the diameter of each point as a function of the time (or frame number). Figure 12c shows points at which three measurements were made, and Figure 12d shows the plots of the measurements made and these three plots on successive frames.





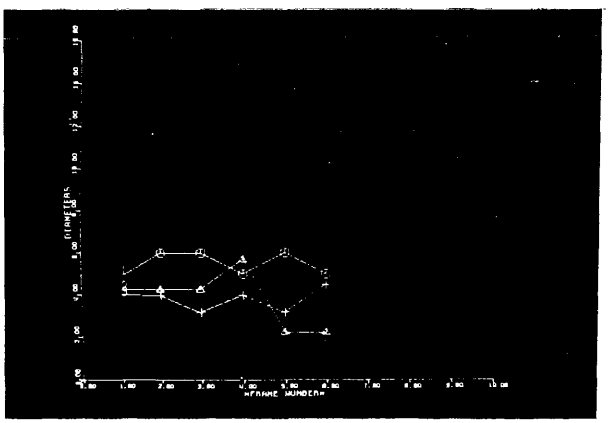


Figure 12—*a*. Illustration of the complete analysis of the diameters of eight points on the retinal vessels. *b*. Illustration showing only the points selected where the retinal image has been superimposed so as not to interfere with the point locations. *c*. Computer analysis of three points. *d*. Plot of diameter results from the three points from several frames

CHEST X RAYS

Chest X rays can be used to aid in the diagnosis of pneumoconioses, tumors, pneumonias, and occurrences of pleural fluid. Previous work of others has included the evaluation

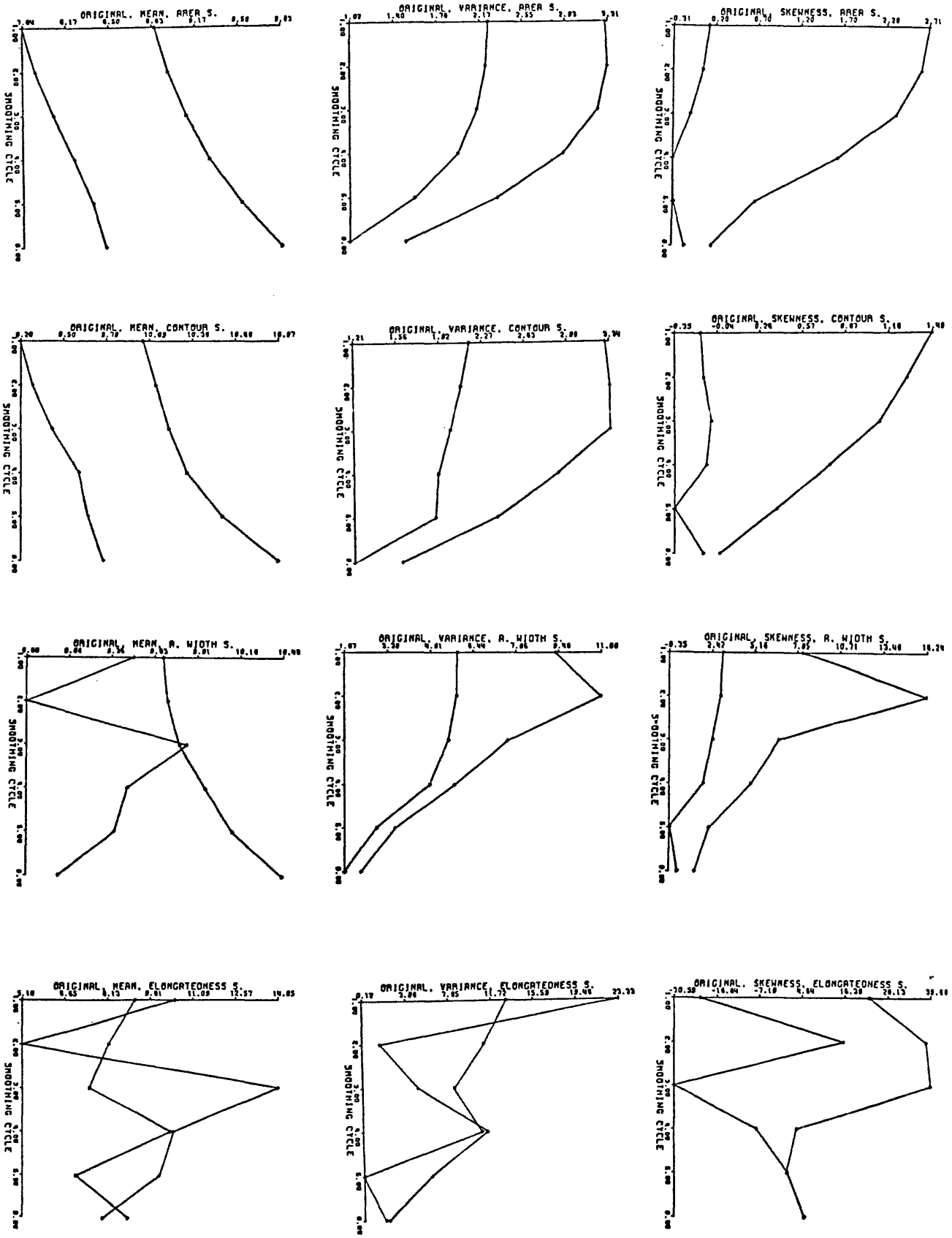


Figure 13—Illustrations of the 25 parameter curves for each of two X rays taken on patients with different stages of pneumonoconiosis. The array curves correspond to the posterior anterior matrices respectively.

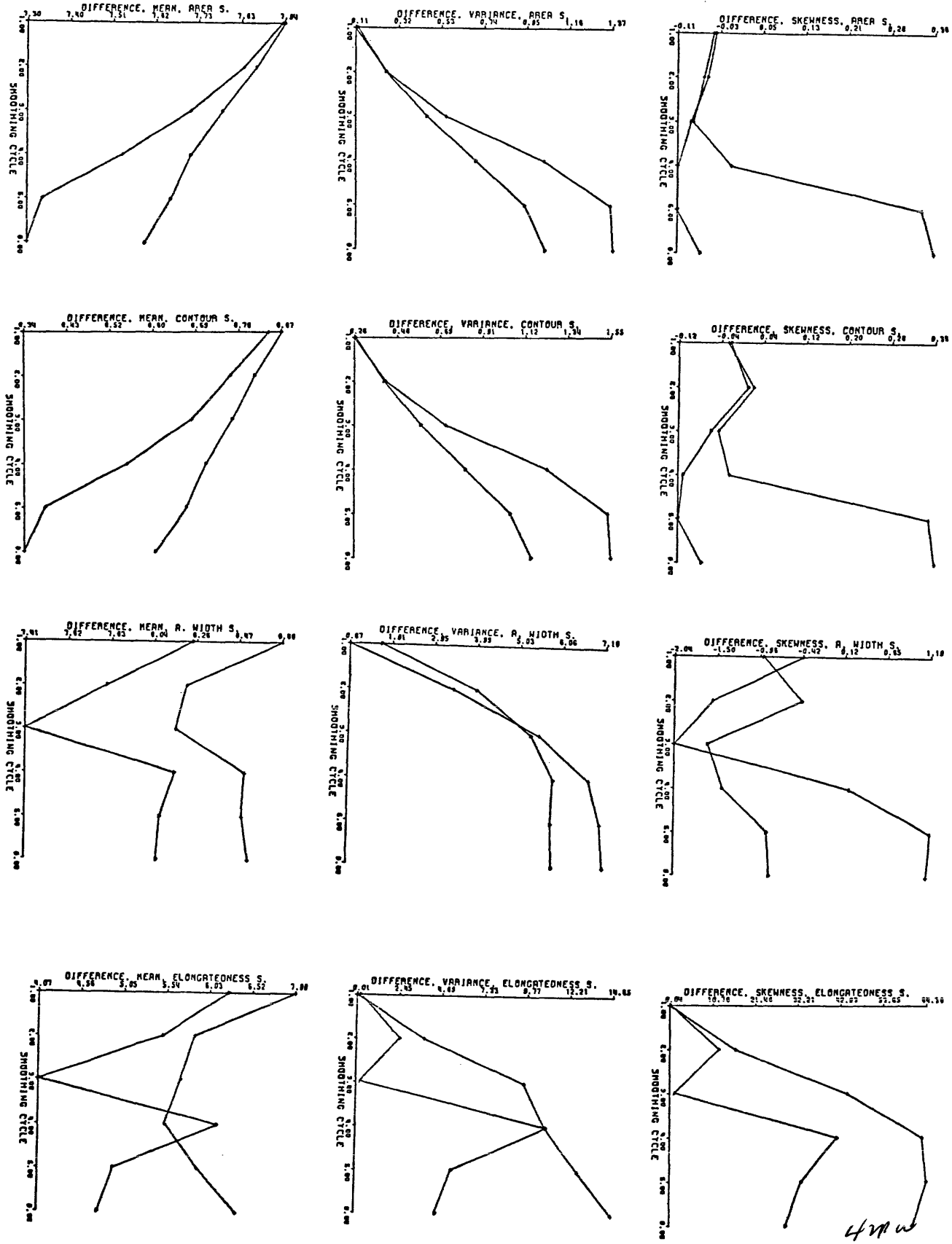


Figure 13—Continued

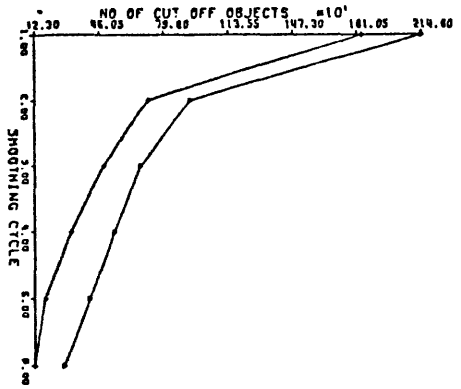


Figure 13—Continued

from chest X-rays of cardiomegalies and specific heart-chamber enlargements.[22–24] Here our goal is to classify a chest X-ray film automatically according to the UICC/ Cincinnati standard films. This is a set of X-rays showing specific examples as "standards" for the classification of the severity of pneumoconiosis, or "black lung" disease, contracted by coal miners. Our approach to the automation conceptually follows that taken by the radiologist in his evaluation of chest films. Thus we recognize the various anatomic features, regions, and parts of the lung and rib cage (and heart), and then characterize the type and extent of "opacities" in each of the regions of importance by means of a texture analysis. The characterization of the anatomic features is accomplished by interactive aids in which the apices of the lungs are pointed out to the computer and standard lung feature curves are drawn projectively through the points, as in other applications described above. For the texture analysis, the method is first to develop 25 texture-parameter functions for each of the standard X-ray plates. Then, for a particular patient, the 25 parameter functions are compared with those of the standards, and the diagnosis corresponds to that standard which most closely matches the 25-parameter vector-function of the patient.

The 25 parameter functions of two X-rays of different severities of pneumoconiosis are shown in Figure 13.

The texture parameters are developed as follows.[25] We attempt to characterize a function $g(x, y)$ by means of attributes that can be derived from the grey-level partition of $g(x, y)$, i.e.

$$g(x, y) = \{g(x, y, 1), g(x, y, 2), \cdots g(x, y, n)\}$$

where $g(x, y, i)$ is the function $g(x, y)$ defined only for those points (x, y) for which $g(x, y) = i$. Each such partition can be characterized by its area, boundary length, "width," and "proportion." The area is simply the number of spots (x, y) for which $g(x, y) = i$. The boundary is the length of the contour lines separating $g(x, y, i)$ from $g(x, y, i\text{-}1)$ and $g(x, y, i+1)$. The "width" is defined as the area divided by the boundary length. This is analogous to the width of an annular ring between two circles of radius r_1 and r_2, for then

$$r_2 - r_1 = \frac{\pi(r_2^2 - r_1^2)}{\pi(r_2 + r_1)} = \frac{\text{area of annular ring}}{\pi \times \text{boundary length}}$$

Finally the "proportion" is defined as the width divided by one half the boundary length.

Thus for each grey-level value, each of the four attributes can be computed, producing a "spectrum" for each attribute. That is, the attribute value will be a function of the grey-level, e.g.,

$$A_i = A_i[g(x, y, i)], \; i = 1, 2, \ldots, n$$

for n grey-levels. However, what is really desired is a set of parameters that describe the behavior of the attributes as functions of the grey levels; i.e., we wish to characterize A_i as $A_i = A(i)$. For this we simply choose the first, second, and third moments (i.e. the mean, variance, and skewness, respectively). Summarizing, for each of the four attributes, spectra are generated; and for each spectra, three moments are computed, making $4 \times 3 = 12$ parameters.

However, for texture analysis the important thing is the variation of the parameters when computed for a sequence of "smoothing pictures" $\bar{g}_i(x)$ for neighborhoods i of increasing sizes. Here

$$\bar{g}_i(x) = \frac{1}{a_i} \sum_{x \epsilon n_i} g(x)$$

where $x = (x, y)$ and n_i is the set of points in the neighborhood of size i around x, and a_i is the number of points of n_i. Thus for each smoothing picture we obtain the 12 parameters; or in other words, the 12 parameters are actually functions of the smoothing cycle. Hence we have an array of parameter functions:

$$Pij(s) = \begin{bmatrix} P_{11}(s) & P_{12}(s) & P_{13}(s) \\ P_{21}(s) & P_{22}(s) & P_{23}(s) \\ P_{31}(s) & P_{32}(s) & P_{33}(s) \\ P_{41}(s) & P_{42}(s) & P_{43}(s) \end{bmatrix}$$

where i indexes the attributes (i.e., area, boundary length, width, and proportion), j indexes the moments of the spectrum (i.e., mean, variance, and skewness), and s represents the smoothing cycle (i.e., neighborhood size).

Often it is also important to work with the so-called difference picture, namely

$$d_s(x, y) = g(x, y) - \bar{g}_s(x, y)$$

We can once again compute an array of parameter functions based on the difference pictures. For each smoothing cycle s, we have a difference picture and hence the 12 parameters. Thus we can form an array of parameter functions:

$$Dij(s) = \begin{bmatrix} D_{11}(s) & D_{12}(s) & D_{13}(s) \\ D_{21}(s) & D_{22}(s) & D_{23}(s) \\ D_{31}(s) & D_{32}(s) & D_{33}(s) \\ D_{41}(s) & D_{42}(s) & D_{43}(s) \end{bmatrix}$$

where, as above, i indexes the attributes, j indexes the moments of the spectrum, s represents the smoothing cycle on which the difference picture is based.

Finally, we come to the parameter that is the count per unit area of the number of local maxima. This is accomplished in an efficient and convenient, although approximate, manner by utilizing the difference picture. The mean grey level is used as a grey-level "cutoff" and the number of "objects" in the picture above this cutoff level are counted. This is done for each of the difference pictures, resulting in a parameter function $C(s)$.

Summarizing, twenty-five parameter functions are used to characterize the texture of a picture, twelve associated with the original picture and its smoothing cycles, twelve associated with the difference pictures, and one additional parameter function associated with the number of local maxima on the smoothing pictures. These, then, are the texture parameters used to characterize an X ray and compare it with the analogous parameters of the standards.

BACTERIOLOGY SLIDE SCANNING

Automation of clinical microscopy is eminently suited to the applications of pattern recognition. At the present time, clinical microbiologists spend considerable time at the microscope scanning slides in order to determine Gram reaction, size, shape, and general morphology of bacteria isolated from clinical specimens. An area of application for the automatic microscope is precisely this onerous task of routine microscopy. A variety of tissues and specimens (urine, sputum, etc.) can be screened quickly and accurately for the presence of bacteria. Several stains are routinely used, for example, the Gram stain, the acid-fast stain, and the simple stain (methylene blue), as well as others. The appropriate parameters can be established and the computer programmed to interpret observations from the microscope. The general techniques of staining etc. are rather concisely presented in the *Manual of Methods in Clinical Microbiology* published by the American Society for Microbiology (for example, see page 58). If one were able to provide a reasonably priced automatic scanning system using a modification of the routine light microscope, then considerable savings in hospital bacteriology-laboratory costs as well as an improved accuracy of diagnosis can be achieved.

Automation in the bacteriology laboratory has heretofore been mainly applied to bacterial-culture analysis, such as the automated reader for large assay plates, automatic growth recorder for microbial cultures, automatic plate streaker, etc.[26–28] Such apparatus examines or handles the bacteria on

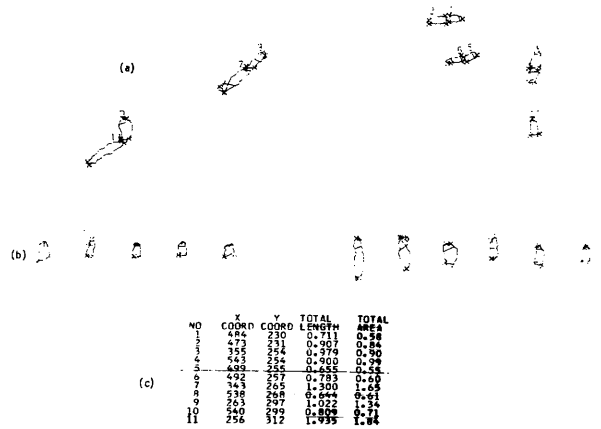


Figure 14—*a*. E. Coli analyzed by the computer. *b*. Computer alignment of the E. Coli bacteria. *c*. Table of values computed for each of the bacteria

a macroscopic level, in terms of colonies or turbidity measures. Little if any automation has been approached at a microscopic scale. However, even the initial procedure in the manual identification of bacteria is the examination of the Gram-stained bacteria at a microscopic level, for whether an organism is a Gram-positive or Gram-negative coccus or rod now determines the methods used for its cultivation and identification on the macroscopic level.

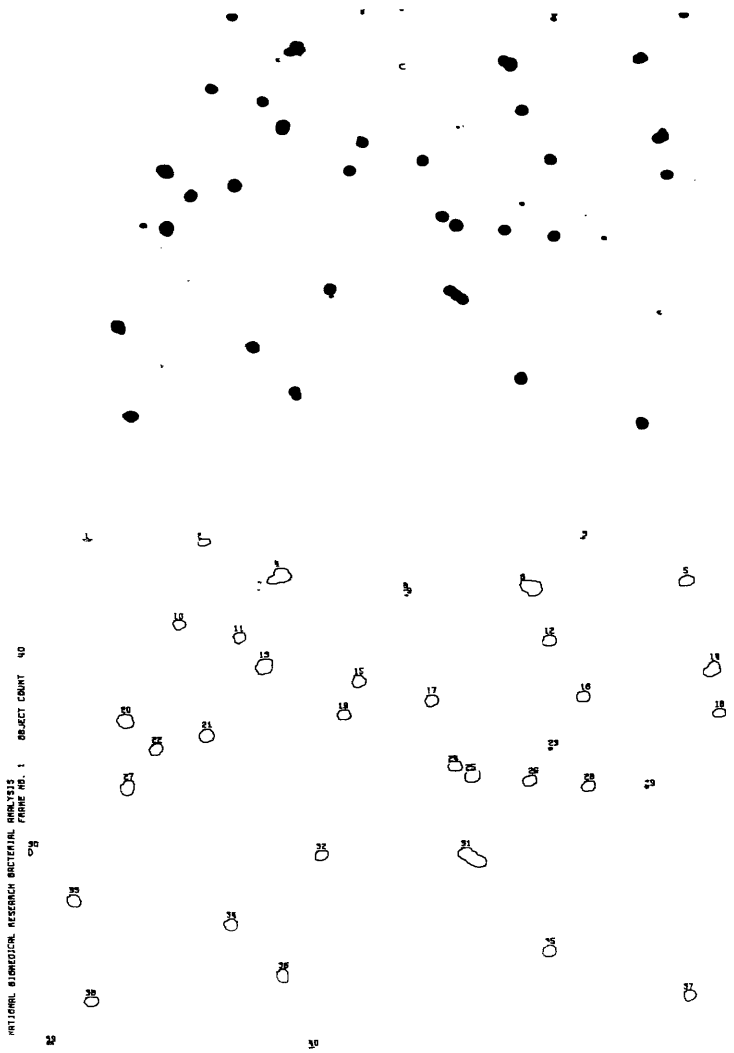
At present, microscopic examination of the bacteria is made at only two points in the clinical bacteriological examination. The first is on fresh fluid or tissues from the patient, which are examined microscopically using wet mounts with phase contrast or vital stains or using dried smears with the Ziehl-Neelsen stain as well as the Gram stain. A microscopic examination can be again made from the bacterial cultures on the streaked and incubated plates. These examinations are usually only cursory in nature, however, because of the tedium and difficulties involved in counting or measuring the bacteria. That is, the information obtained from the microscopic examination of bacteria in routine clinical examination is limited at present not so much by the information available in the microscopic field as by the inability of manual methods to conveniently and effectively extract from the microscopic field all the information that exists there. Thus the full potential of the microscopic examination of bacteria has not been reached.

The automation of the microscopic examination by the SPIDAC automatic microscope holds promise not only of eliminating the tedium of present methods but also of opening the way for the development of entirely new techniques in clinical bacteriology based on such direct microscopic examination. For instance, very sparce populations of bacteria can be found by means of the relentlessly systematic complete scanning of a slide that is accomplished by automatic means. The more precise automatic measurement of morphological characteristics can increase the specificity with which the bacteria can be recognized at an initial microscopic examination. Utilizing the SPIDAC, it now becomes feasible to develop new reactions or stains that can aid in differentiating

between different types of bacteria on a microscopic, individual-organism level.

The advantages of such developments are great. First, the time required to identify bacteria is substantially reduced in most cases. This time reduction occurs because of the increased information obtainable from the examination of the initial specimen, and because only very tiny colonies are necessary for examination of bacteria after streaking. Second, automatic microscopic analysis reduces the space requirements for large incubators and the number of personnel needed for handling and manipulating the many bacteriological plates. Finally, such direct automatic analysis can enable the development of the most accurate methods for determining the antibiotic sensitivity of bacteria.

Figure 14a shows E. coli bacteria scattered in a field, where the ends of each bacteria have been determined and each has been given a number corresponding to the order in which they were recognized by the computer. In Figure 14b these bacteria have been aligned above their corresponding accession number; in Figure 14c a table of total length and total area for



OBJECT NO	BOUNDARY	AREA	MAJOR AXIS	MINOR AXIS	RATIO	INDEX
1	20.5	25.0	9.000	5.000	0.556	0.06
2	28.5	67.0	10.440	8.485	0.813	0.08
3	10.8	7.0	5.000	1.414	0.283	0.06
4	56.3	199.0	20.616	13.416	0.651	0.06
5	36.1	100.0	13.000	9.220	0.709	0.08
6	53.5	192.0	18.682	13.038	0.698	0.07
8	10.2	7.0	4.472	2.000	0.447	0.07
9	6.8	6.0	3.000	1.414	0.471	0.13
10	31.6	78.0	11.045	8.544	0.774	0.08
11	33.0	85.0	11.045	9.487	0.859	0.08
12	35.0	98.0	12.369	9.849	0.796	0.08
13	45.0	158.0	15.000	13.454	0.897	0.08
14	44.6	140.0	15.000	11.402	0.760	0.07
15	35.6	101.0	11.705	11.402	0.974	0.08
16	34.1	92.0	11.402	9.487	0.832	0.08
17	35.0	101.0	11.402	10.440	0.916	0.08
18	30.1	74.0	10.440	8.602	0.824	0.08
19	34.1	90.0	11.180	9.220	0.825	0.08
20	43.2	143.0	14.866	11.705	0.787	0.08
21	37.8	117.0	12.369	11.402	0.922	0.08
22	33.6	94.0	11.402	9.849	0.864	0.08
23	11.1	12.0	4.123	3.162	0.767	0.10
24	34.1	97.0	12.369	10.000	0.808	0.08
25	38.1	121.0	13.038	11.662	0.894	0.08
26	34.7	94.0	11.402	10.817	0.949	0.08
27	39.6	128.0	13.038	12.207	0.936	0.08
28	33.6	94.0	12.083	9.434	0.781	0.08
29	9.7	11.0	4.000	2.828	0.707	0.12
30	15.1	19.0	6.083	4.243	0.697	0.08
31	64.2	235.0	25.495	19.235	0.754	0.06
32	32.7	88.0	11.180	9.899	0.885	0.08
33	35.6	100.0	12.530	9.899	0.790	0.08
34	35.0	100.0	12.083	10.198	0.844	0.08
35	35.0	103.0	11.705	10.440	0.892	0.09
36	37.2	101.0	12.207	9.849	0.807	0.07
37	33.3	86.0	10.817	10.630	0.983	0.08
38	33.6	92.0	12.166	10.000	0.822	0.09
39	14.8	18.0	6.083	5.385	0.885	0.08
40	9.4	9.0	4.123	3.162	0.767	0.10
AVERAGE	31.0	87.0	10.949	8.797	0.803	0.08

Figure 15—*a*. Photomicrograph of a field of candida albicans. *b*. Organisms as determined by the computer. *c*. Table of values computed for each object in the field

each of the bacteria is given. In Figure 15a we show some Candida albicans, a gram positive yeast organism. Figure 15b shows the organisms as determined by the computer, and Figure 15c gives the area, the length of the major axes, the length of the minor axes, the ratio of the minor to major axes, and the value of an index which is the area divided by the square of the contour length. From such data, average values for individual organisms can be determined, even though some of the boundaries encompass more than one touching or adjacent organism.

ACKNOWLEDGMENTS

This work was supported by grants HD-05361, RR-05681, GM-15192, and GM-10797 from the National Institutes of Health, Public Health Service, Department of Health, Education, and Welfare, to the National Biomedical Research Foundation.

The electrical instrumentation was built under the direction of Thomas Golab, Yeshwant Kulkarni, and Louis S. Rotolo, and the software was developed by Han K. Huang, Fred Ledley, Gerard Pence, Menfai Shiu, James Ungerleider, and James B. Wilson of the National Biomedical Research Foundation. The medical applications were under the direction of Dr. Margaret Abernathy, Dr. William E. Battle, Dr. Jay N. Cohn, Dr. Peter Y. Evans, Dr. Vincent F. Garagusi, and Dr. Homer L. Twigg of the Georgetown University Medical School; and Dr. Rita R. Colwell of the University of Maryland.

REFERENCES

1. Lawson, R. N., "Implications of Surface Temperature in the Diagnosis of Breast Cancer," *Can. Med. Assn Journal*, Vol. 75, p. 309, 1956.
2. Wood, E. H., "Thermography in the Diagnosis of Cerebrovascular Disease," *Radiology*, Vol. 83, pp. 540-542, 1964.
3. Abernathy, M., O'Doherty, D. S., "The Diagnosis of Extracranial Carotid Artery Insufficiency by Infrared Thermography," *American Heart Journal*, Vol. 82, pp. 731-741, December 1971.
4. Isard, H. J., Bernard, J. O., Shilo, R., "Thermography in Breast Cancer," *Gyn. and Ob.*, Vol. 128, pp. 1289-1293, 1969.
5. Aarts, N. J. M., "Thermography in Malignant and Inflammatory Diseases of the Bones, Medical Thermography, *Proceedings of a Boerhaave Course for Post Graduate Medical Education*, Leiden, 1968.
6. Reynolds, W. A., Ayers, M. A., Parker, G. M., "Thermoplacentography," *Radiology*, Vol. 80, pp. 825–827, 1967.
7. Mladick, R. N., Georgiade, N., Thorne, F., "A Clinical Evaluation of the Use of Thermography in Determining Degree of Burn Injury," *Plastic and Reconstructive Surgery*, Vol. 38, pp. 512–518, 1966.
8. Abernathy, M., Ronan, J., Winsor, D., "Diagnosis of Coarctation of the Aorta by Infrared Thermography," *Am. Heart Journal*, Vol. 82, pp. 731-741, December 1971.
9. Segal, B. L., "Echocardiography," *Modern Concepts of Cardiovascular Disease*, Vol. 38, No. 11, pp. 63-67, November 1969.
10. Elder, I. E., "Ultrasoundcardiography in Mitral Valve Stenosis," *The American Journal of Cardiology*, Vol. 19, pp. 18-31, January 1967.
11. Dodge, H. T., Hay, R. E., Sandler, H., "Pressure-Volume Characteristics of the Diastolic Left Ventricle of Man with Heart Disease," *American Heart Journal*, Vol. 64, p. 503, October 1962.
12. Bunnell, L. L., Grant, C., Jain, S. C., Carlisle, R., "One-Plane Cineangiographic Volume Estimates of the Left Ventricle in Man," *Federation Proceedings*, Vol. 25, No. 2, Part I, p. 279, March/April, 1965.
13. Sandler, H., Hawley, R. R., Dodge, H. T., Baxley, W. A., "Calculation of Left Ventricular Volume from Plane (A-P) Angiocardiogram," *Proceedings of American Society for Clinical Investigation*, p. 78, 1965.
14. Chapman, C. B., Baker, O., Reynolds, J., Bonte, F. J., "Use of Biplane Cinefluorography of Measurement of Ventricular Volume," *Circulation*, Vol. 18, pp. 1105-1117, December 1958.
15. Ledley, R. S., "High Speed Automatic Analysis of Biomedical Pictures," *Science*, Vol. 146, No. 3641, pp. 216-223, October 9, 1964.
16. Chow, C. K., Kaneko, T., "Automatic Boundary Detection of the Left Ventricle from Cineangiograms" to appear in *Computers and Biomedical Research*, Vol. 5, No. 4, August 1972.
17. Dollery, C. T., "Dynamic Aspects of the Retinal Microcirculation," *Arch. Ophthal.* 79, p. 536, 1968.
18. Evans, P. Y., "Retinal Circulation Times," *1st Intern. Symp. Fluor. Angiogr.*, Albi, France, 1969.
19. Evans, P. Y., Wruck, J., "Macular Circulation Times," *21st Intern. Cong. Ophthamology*, Mexico, 1970.
20. Shimizu, K., "Basic Patterns of Inflow of Fluorescein into the Retinal Arterioles," *2nd Intern. Symp. Fluor. Angiogr.*, Miami, 1970.
21. Evans, P. Y., Wruck, J., "Fluorescein Cinephotography in Macular Disease," *Photography in Ophthamology 2nd Intern. Symp. Fluor. Angiogr.*, Miami, 1970.
22. Kruger, R. P., Hall, E. L., Dwyer, S. J., Lodwick, G. S., "Digital Techniques for Image Enhancement of Radiographs," *Int. J. Biomed. Comp.*, Vol. 2, pp. 215-238, July, 1971.
23. Sutton, R. N., Hall, E. L., "Texture Measures for Automatic Classification of Pulmonary Disease," *IEEE Trans. Comp.*, Vol. C-21, No. 7, pp. 667-676, July 1972.
24. Toriwaki, J., Fukumura, T., "Computer Program System for Processing of Complex Photographs with Application to Automatic Interpretation of Chest X-Ray Films," *Pattern Recognition*.
25. Ledley, R. S., "Texture Problems in Biomedical Pattern Recognition," *Proceedings of the 1972 IEEE Conference on Decision and Control and 11th Symposium on Adaptive Processes*, pp. 590-595, December 13-15, 1973, New Orleans, Louisiana.
26. Baillie, A., Gilbert, R. J., (eds.), *Automation Mechanization and Data Handling in Microbiology*, Academic Press, New York, 1970.
27. Spanberry, M. N., "Computer Processing of Microbiology Data-Part of a Total Laboratory System," *Am. J. Med. Tech.*, Vol. 35, pp. 77-92, February 1969.
28. Trotman, R. E., "The Application of Automatic Methods to Diagnostic Bacteriology," *Biomed. Eng.*, Vol. 7, No. 3, pp. 122-131, April 1972.

Interactive pattern recognition—A designer's tool

by EDWARD J. SIMMONS, JR.

Rome Air Development Center
Griffiss AFB, New York

INTRODUCTION

The marriage of interactive processing techniques with the technology of pattern recognition is particularly significant to the designers of equipment requiring the automatic recognition of objects or events. This new tool allows the system designers to develop better recognition logic more quickly than in the past. But, perhaps most importantly, they can develop this logic themselves, so all design alternative and resulting systems effects can be analyzed in detail.

This paper will show how easy it is to use an interactive pattern recognition system to solve a variety of problems. It is hoped that other similar systems will be developed and used to improve the quality of human life.

INTERACTIVE PATTERN RECOGNITION

The Air Force's Rome Air Development Center has produced a unique tool for solving automatic recognition problems, called the **O**n-Line **P**attern **A**nalysis and **R**ecognition **S**ystem (OLPARS). This interactive pattern recognition tool has been used quite successfully by system design engineers and research workers, who are familiar with pattern recognition concepts, but who would hardly qualify as pattern recognition "experts."

Before discussing these applications a few words on the organization of the OLPARS is in order. The system consists of a computer with a graphics console. The console has function keys, a keyboard, a lightpen, and a cursor controlling track ball. There are four main analysis modules forming the OLPARS: data structure analysis, measurement evaluation, data transformations, and recognition logic design. In discussing the various applications, the purpose of each of these modules will be explained. Each module contains several algorithms which may be called as needed to aid in the analysis and logic development. Underlying these modules is the OLPARS executive which provides the data filing system, passes the proper information to each called algorithm, and formats various graphics data displays to the user. Extensive details of the OLPARS organization may be found in References 1 and 2.

Table I contains a listing of the applications on which the OLPARS has been used along with the principal investigator. Note the wide variety of users, ranging from Dr. Sammon, a noted authority in pattern recognition, to Mr. Dragg, an engineer having no formal training in pattern recognition.

One of the early applications of the OLPARS was in the traditional pattern recognition problem of spoken word recognition, in this case confined to the ten digits zero through nine. Each of seven speakers spoke the digit five times, producing 35 samples for each digit. These samples were digitized and 24 features, based on zero crossing information, were extracted.

Dr. Foley[3] has developed an engineering rule of thumb regarding the minimum sample size of the design set. This rule states that if the ratio of the number of samples per class to the number of features is less than three, then the performance of the recognition logic is unreliable, i.e., results on the independent test set will be quite different then the design set results. Clearly this rule is violated for the speech data, so it would be foolish to attempt to design recognition logic. However, it is reasonable to perform the first step in a pattern recognition problem, namely, data structure analysis.

There are two reasons to perform data structure analysis. The first is to search for separable clusters or modes. If these are found then the data must be partitioned into subclasses so that each subclass has only one mode. Figure 1, Multi-Mode Class, demonstrates this point. Suppose the nearest mean vector logic is to be used. The mean vectors of class A and Class B fall almost on top of each other. If class A is broken into subclasses then the mean vectors of each of three clusters will separate quite nicely. The second reason for structure analysis is to detect any "wild shots," caused by an error in the data collection, which would produce inaccurate recognition logic.

Performing structure analysis on the speech data enabled the user to make an interesting discovery about his data collection. Figure 2, Spoken Sixes, shows the spoken sixes as plotted on a plane defined by the two covariance eigenvectors whose eigenvalues are largest. This plane, then, is the plane in our 24 dimensional hyperspace which contains the maximum variance in our data. Note that

TABLE I—OLPARS Applications

APPLICATION	PRINCIPAL INVESTIGATOR
1. Spoken Digit Recognition	Mr. Chapin RADC In-House
2. Ground Sensors	Mr. Proctor RADC In-House
3. Earth Resources	Mr. Dragg NASA Manned Spacecraft Center
4. Shock Trauma Diagnosis	Dr. Sacco Army—Edgewood Arsenal
5. Genetic Disorder Recognition	Dr. Stowens Utica State Hospital Mr. Proctor RADC In-House
6. Aerospace Object Shape Identification	Mr. Webb RADC In-House
7. Radar Emitter Identification	Mr. Elefante RADC In-House
8. Handprinted Characters	Dr. Sammon Pattern Analysis and Recognition Corporation (RADC Contract)
9. Map Features	Capt. Fick RADC In-House
10. Photo-Interpretation	Dr. Sacco Army—Edgewood Arsenal
11. Helicopter Mission Profiles	Mr. Merritt Army—Edgewood Arsenal
12. Photometric Analysis Techniques	Mr. Roberts/RADC In-House & Lt. Harkleroad/AF Space & Missile Systems Organization

there are two clusters. A similar result also occurs for sevens, while each of the remaining classes have only one cluster, as expected. Being on-line with the computer made understanding this phenomenon very simple, the operator simply outputs to the line printer the vector indices of each point in each cluster. This simplifies going back to the original waveform data producing that vector to learn the cause. In this case, it was that the "s" sound is of such low energy content that the device which initialized the feature extractor was not triggered by the "s" for some speakers. When this threshold was readjusted, the sixes and sevens became the expected single cluster classes. Thus, performing the pattern recognition function interactively enabled the quick identification and correction of data collection errors before much time was wasted.

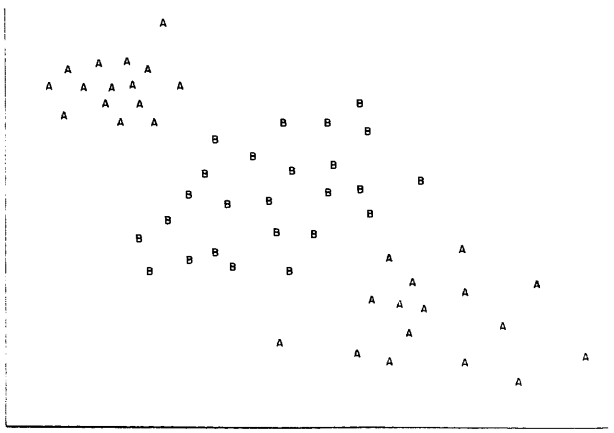


Figure 1—Multi-mode class

Another interesting application of OLPARS is passive identification of ground vehicles using unattended sensors. Here we are dealing with secondary phenomena, which is somewhat unusual in pattern recognition. Printed characters were designed to convey information, so is spoken language. These then are primary phenomena which must be recognized. However, the purpose of a truck is not to make noise or thump the ground, but rather these outputs are incidental. Thus, identifying light and heavy trucks, tracked vehicles, men walking, and aircraft is no easy matter. Add root noise caused by

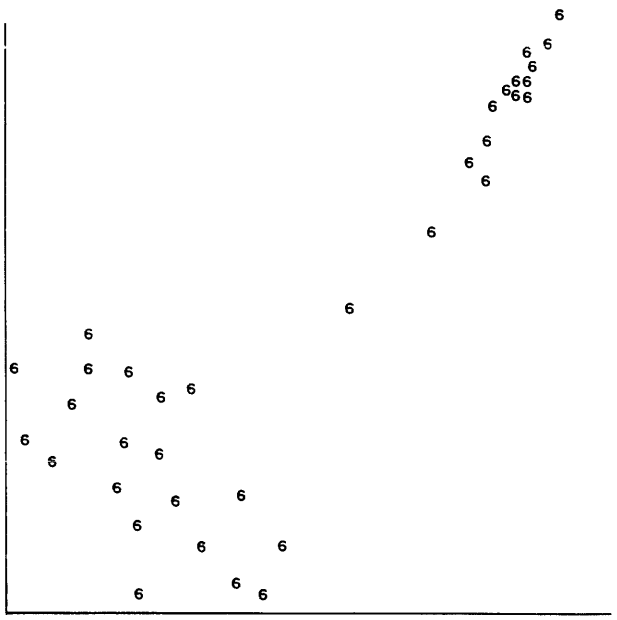


Figure 2—Spoken sixes

swaying trees, stray animals, weather factors (wind, rain, thunder, etc.) and the possibility of two or more targets occurring together, and the problem becomes extremely complex.

Current solutions are based on the single class aspect. In this problem, 48 features were extracted from each target sample, which consists of five seconds of seismic signature data. They include total energy in each five hertz bin of the FFT (Fast Fourier Transform) of the signal, spacing between the harmonics, ratio of maximum deviation to average deviation, and number of frequencies for which the energy exceeds 25, 50, and 75 percent of the maximum energy value. However, these sensors are to be air dropped, impact implanted, and never recovered. This puts severe limitations upon the target classifier complexity that can be afforded, 25 features appearing to be an upper bound.

This data was presented to the OLPARS and a successful solution was found in about one month of analysis. First the measurement evaluation module was used to determine the discriminating power of each of the 48 features. This is done using one of two algorithms, probability of confusion or discriminant measure, which analyzes each feature independently. The former essentially measures the overlap of the class conditional probabilities while the latter measures the significance of a feature in discriminating between classes on a pairwise basis. The OLPARS operator can then get a ranking of each measurement, based on either algorithm, over all classes, for a specified class versus all other classes, or for a specified pair of classes. In this manner the operator can decide on the best five, ten, fifteen, twenty, etc. features, light gun those features he wishes to use and use the data transformations module of the OLPARS to transform his data into two data sets, the original and a new set consisting only of those features selected. In the sensor case, the operator constructed new sets made up of 12, 16, 19, 22, 33, and 44 features. This allowed the design of several target identifiers so that the trade-off between complexity and classification error could be ascertained.

The second step was to use the structure analysis module to analyze the data. Since no "wild shots" were found and the data did have just one mode per class, the operator quickly passed to the recognition logic design module. The algorithm chosen here was Fisher's Linear Discriminant, which works by discriminating between pairs of classes. A direction is found in the hyperspace defined by the features which best satisfy the mutual criteria of maximizing the between class scatter while minimizing the within class scatter. The distance between the two classes is then bisected and this value taken as the threshold. An unknown vector is projected onto this direction and called class 1 if its value is less than the threshold and class 2 if greater. This is done for all pairs of classes and that class getting the most votes is the class to which the unknown vector is assigned. If there is no space between the classes, i.e. they overlap, the threshold is the value bisecting the space between the maximum value of the density functions of each class when plotted on the Fisher direction.

TABLE II—Confusion Matrices for Fisher Thresholds

UNADJUSTED

	H	A	T	M	C
H	121	7	1	2	1
A	1	166	3	1	0
T	4	2	131	0	2
M	4	1	0	127	1
C	3	0	0	6	131
TOTAL SAMPLES	133	176	135	136	135

a.

ADJUSTED

	H	A	T	M	C
H	120	3	1	2	1
A	2	170	4	1	0
T	4	2	130	0	2
M	4	1	0	129	1
C	3	0	0	4	131
TOTAL SAMPLES	133	176	135	136	135

b.

LEGEND: H—Helicopter
A—APC
T—Heavy Truck
M—Human
C—C 131

In the sensor problem, as in most real world problems, the classes overlap so that perfect discrimination is impossible. Table IIa shows the confusion matrix which results from applying the Fisher technique to the 16 feature sensor data. This resulted in 39 identification errors for 715 targets, or a correct identification rate of 94.5 percent. However, OLPARS allows the operator to go one step further in refining his logic. He may look at the data projected on each of the Fisher directions and adjust the thresholds to better fit the shape of the density functions. When this was done, the resultant confusion matrix is shown in Table IIb, note there now are only 34 errors or a 95.2 percent correct identification rate. Since each class has 130 or more samples, Dr. Foley's rule of thumb is satisfied. In fact, an independent test set of 605 targets confirmed these results. This 16 feature design was chosen because it represents the best compromise between complexity and correct target recognition rate. This design is now undergoing testing under field conditions. Thus, the interactive pattern recognition tool—OLPARS——made solving this highly complex problem a task one man could perform in a month.

NASA's Earth Observations Program provided another opportunity for OLPARS to show its usefulness. This data set consisted of the digital output from a 12 channel multi-spectral scanner that had been flown over the Tippecanoe River Valley of Indiana during various parts of the growing season. The purpose was to automatically detect what type of field was being overflown (oats, rye, clover, wheat, corn, soybeans, or alfalfa). NASA personnel were trained in operating the OLPARS and designed recognition logic, which had a correct recognition rate of 98.4 percent in one work week.

During the data structure analysis work something very interesting occurred in this data, the eigenvector projection of the data associated with the two largest eigenvalues gave the barest suggestion that the oats class might be bimodal. Since the user was on-line with the computer, he merely asked for the data projected against the eigenvectors associated with the second and third largest eigenvalues. This produced Figure 3, Oats Projection, where the two modes are easily seen. Thus, in less than five minutes the user went from hunch to verification. Try that on any batch system!

The Army's shock trauma work represents a considerably different use of OLPARS. The system was not asked

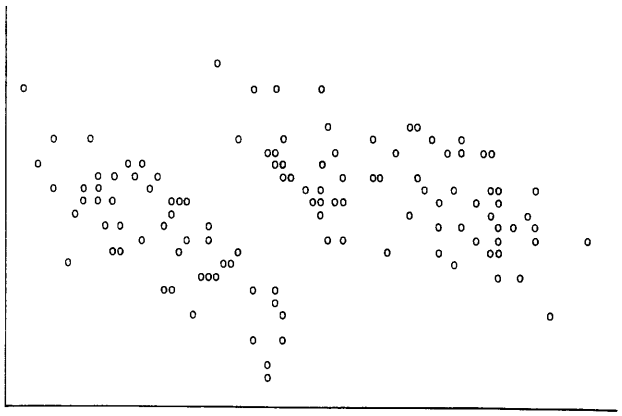


Figure 3—Oats projection

to design logic to discriminate between people who would recover from the trauma, but rather to find the optimum plane in the data space so that the movement of the patient in that plane, as physiological parameters change during the course of treatment, would indicate the effectiveness of the treatment and assist the doctors in determining what therapy to apply or if the patient has passed the point of recovery. The data was provided to the Army by the University of Maryland Center for the Study of Trauma and consisted of 12 measurements: systolic blood pressure, diatolic blood pressure, hemoglobin, hematocrit, serum fibrinogen, serum sodium, serum potassium, serum chloride, serum osmalality, blood urea nitrogen, glucose, and serum creatine. This data was taken at the beginning of therapy and at the end for 70 patients who lived and 70 patients who died.

The OLPARS data structure analysis module contains several algorithms in addition to the eigenvectors discussed previously. Among them are Non-Linear Mapping (NLM), which fits the n-dimensional data onto a plane so that the error in the inter-point distances is minimized, and Discriminant Plane Mapping, which maps two operator selected classes onto a plane defined by the Fisher direction (the same as in Fisher's Linear Discriminant algorithm for classification) and the best orthogonal direction which satisfies the same criteria. All three of these techniques were used in this application; however, NLM did not provide a plane that was useful.

The data was divided into four classes: final measurements on surviving patients (Class A), final measurements on dying patients (Class B), initial measurements on surviving patients (Class L), and initial measurements on dying patients (Class D). The measurement evaluation module was used to select the best six measurements and

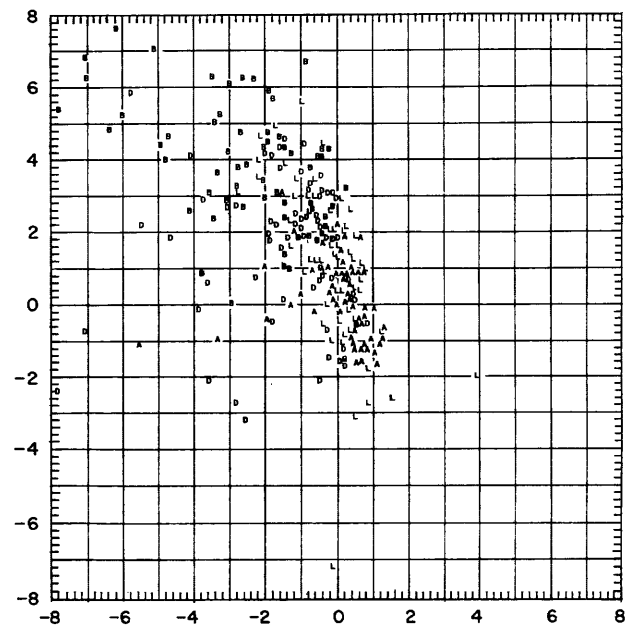


Figure 4 Eigenvector plane

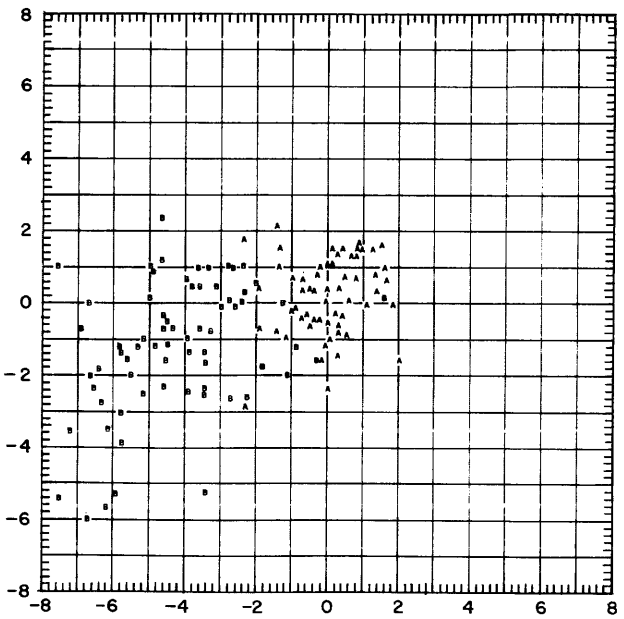


Figure 5—Discriminant plane

the resulting discriminant plane for classes A and B is shown in Figure 5, Discriminant Plane. While some overlap of the classes does occur, there is very definitely some lethal combinations of the measurements and some indicating survival. Figure 4, Eigenvector Plane, shows the eigenvector projection of all the data. Clearly the center of the plane is a region of good prognosis and the left region an area of poor prognosis. Further work is still being conducted on this application.

The last application to be discussed is that of genetic disorder recognition. Dr. Stowens[4] believes that since they are laid down in the womb, they never change except by mutilation, and are unique to each individual, that prints contain a map of the genetic code of the individual. Therefore, certain patterns in these prints should indicate genetic related disorders. To data, OLPARS has been

used in two aspects of this work, recognition of parents with high probability of bearing children afflicted with Down's syndrome and recognition of adult women with tendencies toward schizophrenia.

The features used in these studies consisted of 23 measurements made on each hand. The loops, whorls, and arches of each finger, the four main lines of each palm, the palmer creases, the four inter-digital area patterns, the axial triradii, and the hypothenar pattern were the areas of interest.

In the schizophrenia study data was collected from 82 white women who, in the opinion of *all* staff members of the Utica, N.Y., State Hospital, met *all* diagnostic criteria for this disease. A control population comprised of 295 white women who appeared well and from whom no history of familial or genetic diseases could be elicited was used for comparison. This data was analyzed using the discriminant plane technique discussed earlier. The resulting plane is shown in Figure 6, Schizophrenics vs. Normals. If the Fisher Linear Discriminant drawn in this figure is used, 245 of the 295 control samples lie to the right of this discriminant and 59 of the 82 schizophrenic samples lie to the left. Due to the promise of this technique further data is now being collected.

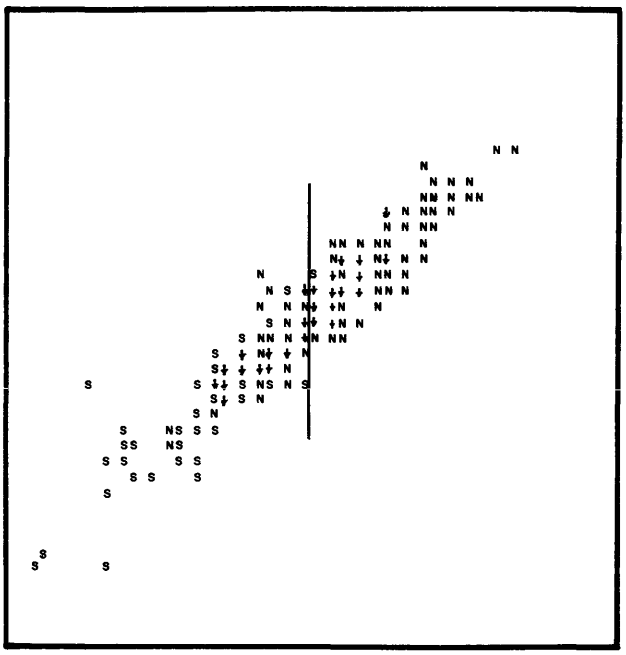


Figure 6—Schizophrenics vs normals

CONCLUSION

Five of the twelve applications of the OLPARS have been discussed with the intention of showing how a system designer or a research worker can use interactive pattern recognition systems as tools to solve automatic target identification problems and as statistical analysis packages. The OLPARS is a versatile tool limited only by the imagination of the user and availability of data. It is hoped that more tools of this nature will be built and used to make human life a bit easier.

ACKNOWLEDGMENTS

The author wishes to thank all the users of the OLPARS for taking advantage of this tool and the management of the Rome Air Development Center for having the courage to build the OLPARS. Special appreciation goes to Mr. Donald Roberts and Mr. Albert Proctor who worked with users, teaching them the operation of the system, and used the system themselves on many of the RADC in-house applications.

REFERENCES

1. Sammon, J. W., Jr., Opitz, C. B., "Programs for On-Line Pattern Analysis," *RADC TR*, 71, 177, Vol. 12, Defense Documentation Center, 732235, 732236.
2. Sammon, J. W., Jr., "Interactive pattern analysis and classification," *IEEE Transactions on Computers*, Vol. C19, July 1970.
3. Foley, D. H., "Considerations of Sample and Feature Size," *IEEE Transactions on Information Theory*, Vol. IT18, September 1972.
4. Stowens, D., Sammon, J. W., Jr., Proctor, A., "Dermatoglyphics in Female Schizophrenia," *The Psychiatric Quarterly*, Vol. 44, July 1970.

Auto scan—Techinque for scanning masses of data to determine potential areas for detailed analysis

by DAVID L. SHIPMAN and CLARENCE R. FULMER

National Aeronautics and Space Administration
Huntsville, Alabama

INTRODUCTION

The Skylab Program is expected to provide advanced technology to assist in the development of large, permanent space stations. Skylab consists of four modules; the Orbital Workshop (OWS) which provides crew quarters and commodities, the Airlock Module (AM) which provides power distribution, atmospheric conditioning and services for Extra Vehicular Activity (EVA), the Multiple Docking Adapter (MDA) which provides for Command and Service Module (CSM) docking, and the Apollo Telescope Mount (ATM) which is essentially a manned solar observatory. Skylab will carry on board a number of experiments in scientific, technological, engineering, and medical areas including those in the ATM. The ATM is also an experiment. Several types of data will be gathered from the experiments including film, samples, observations, as well as transducer and thermocouple measurements. There will also be data from Skylab subsystems measurements since several subsystems have never been flown. The total number of measurements from the experiments and the Skylab subsystems will exceed two thousand measurements, a large number of which will be operating for twenty-four hours per day over the entire mission of eight months.

One million pages of computer printout for each twenty-four hour period may well be required to present all the data gathered from the measurements. The analysis of this volume of data either requires an excessively large number of engineers for the allotted twenty-four hour period or the development of some method for automatically selecting only those data which require some indepth analysis. The AUTO SCAN program is a computer program which searches all incoming data for data points which exceed some predetermined limit. The data points which are outside the limits are sent to the system analysts for detailed analysis. The system analyst is then relieved of the requirement to review all data and can concentrate on analyzing the exceptional cases. This corresponds to the management by exception principle.

DATA TRANSMISSION

The data from the Skylab measurements will be transmitted to the ground receiver station via telemetry at predetermined intervals. The receiving ground station will remove all redundant data points using a zero order algorithm according to a predetermined priority scheme and will transmit the resulting compressed data stream to the using NASA Center via data lines where it will be stored on magnetic tapes. The using NASA Center will then remove overlapping ground station coverage, chronologically order the data, and insert data from onboard recorders into the data stream where there are gaps in ground station coverage: the final data stream will then be put into a specific format called All Digital Data Tape or ADDT. One ADDT will be made for the AM telemetry system and one for the ATM telemetry system. These ADDT will become the primary input variable for the AUTO SCAN program.

PROGRAM CONSTRAINTS

Several major constraints and/or groundrules were baselined for the AUTO SCAN program as the first step in the development process. These constraints were:

1. The program must accept the ADDT and execute as the ADDT becomes available.
2. The program must be written in FORTRAN IV for use on the UNIVAC 1108 EXEC VIII computer using a maximum of 65K word core storage. Other languages are permissible only when it can be demonstrated that, for certain segments, another language is more desirable.
3. The program must be modular, such that modules can be removed or inserted without affecting the operational capability of the program.
4. The program is intended for use on the Skylab Workshop TM data to identify anomalous data and should not be designed to perform analysis or evaluation.
5. The program must be able to identify out of limit data and confirm the occurrence of specific events.
6. The computer run time must not exceed two hours for each set of data transmission which will occur about every six hours.

PROGRAM REQUIREMENTS

The next step in the development of AUTO SCAN was the establishment of the requirements for the program. The engineers responsible for each of the systems and subsystems as well as the experiments were expected to be the potential users of the AUTO SCAN program. Requirements submitted by these users were synthesized to identify the following types of information needed for the program development.

1. The measurement title and description and the associated processing priority required.
2. Related measurements that may be correlated with the desired measurement should be out of tolerance condition develop.
3. Measurement characteristics such as discrete/event, steady state or other (i.e., slopes, cyclic, consumables, etc.).
4. The time span over which an AUTO SCAN run is desired.
5. Special calculations that may be required before an AUTO SCAN run can be made.
6. The nominal value and the upper and lower test limit for each measurement. Changes in either of the limits that can be predetermined should be specified.

In reviewing user requirements, it was noted that many measurements were requested by several different users whose purpose for the measurement were completely different. As a result, the requests for measurement were completely different. As a result, the requests for measurement scans was significantly higher than the requests for measurements (on the order of about 1.5). This additional scanning capability compounds the core storage and computer run time problems. Accordingly, considerable coordination with the users was required to fit the requirements within the current AUTO SCAN program constraints. Ideally, the AUTO SCAN program's physical development should have begun only after virtually all the requirements had been generated or at least until after a wide range of requirements had been received. Both Skylab development problems and manpower limitations precluded this ideal approach. An iterative approach was thus adopted whereby the overall planning was made for all anticipated requirements, but programming was done only on parts of a minimum capability program and as these parts were developed, additional capability was added.

PHYSICAL DESCRIPTION OF THE SYSTEM

A pictorial view of the AUTO SCAN program is presented in Figure 1. After several iterations, a baseline AUTO SCAN program was developed. This baseline program actually consists of four large subprograms.

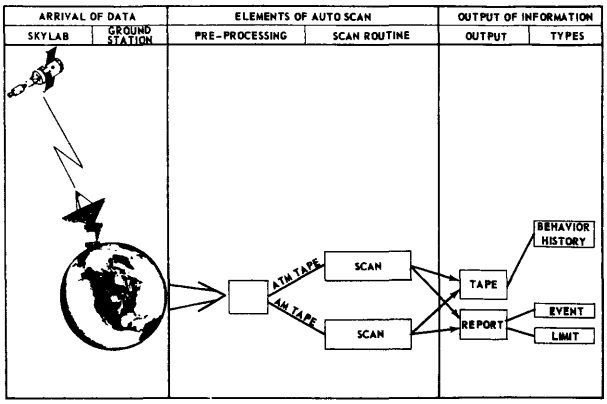
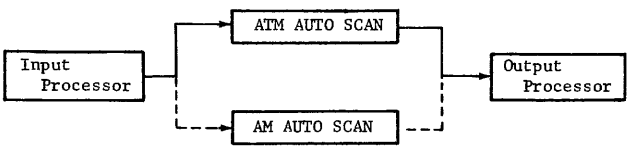


Figure 1—Description of the physical system

1. *Input Processor*—The input for this subprogram is either cards or magnetic tapes reflecting the various requirements which are received. This subprogram arranges the requirements by system and provides the necessary scan instructions to the other subprograms.
2. *ATM AUTO SCAN*—This subprogram scans the ATM ADDT using instructions from the Input Processor and creates magnetic tapes containing the out-of-limit violations, the keying required to obtain plots or tabulations from another program during the out-of-limit interval and the data which may be required for some statistical calculations. Storage was allocated for 2047 scans of the ATM ADDT.
3. *AM AUTO SCAN*—This subprogram is essentially the same as the ATM AUTO SCAN except that the operations are performed on the AM ADDT.
4. *Output Processor*—This subprogram operates on the data generated in the two scan subprograms to presend usable hardcopy output or printout.

The input to and output from the subprograms of the baseline AUTO SCAN program are shown in Figure 2. Some of the terms or acronyms used in this figure are defined in the Appendix.

Flow through these subprograms is expected to be as follows. The path is first through the solid lines and then through the dashed lines.



A generalized flow diagram of this baseline program is shown in Figure 3. This baseline program, with required storage for the telemetered measurements, uses about 50K word core storage for the UNIVAC 1108 computer. A number of Special Computation Modules are being developed and will be added to the baseline program as core storage and run time permits. The baseline AUTO SCAN

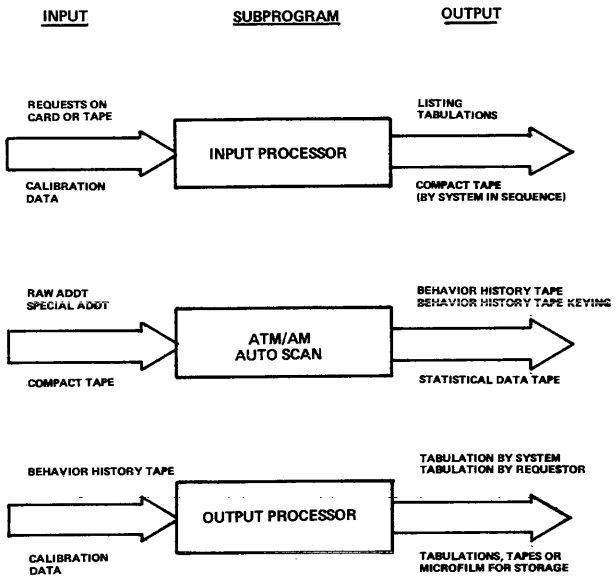


Figure 2—Baseline auto scan program

program is essentially input/output bound, i.e., the clock time is much larger than the CPU time. The Special Computation Modules will utilize the multiprogramming capability of the UNIVAC 1108 resulting in co-processing

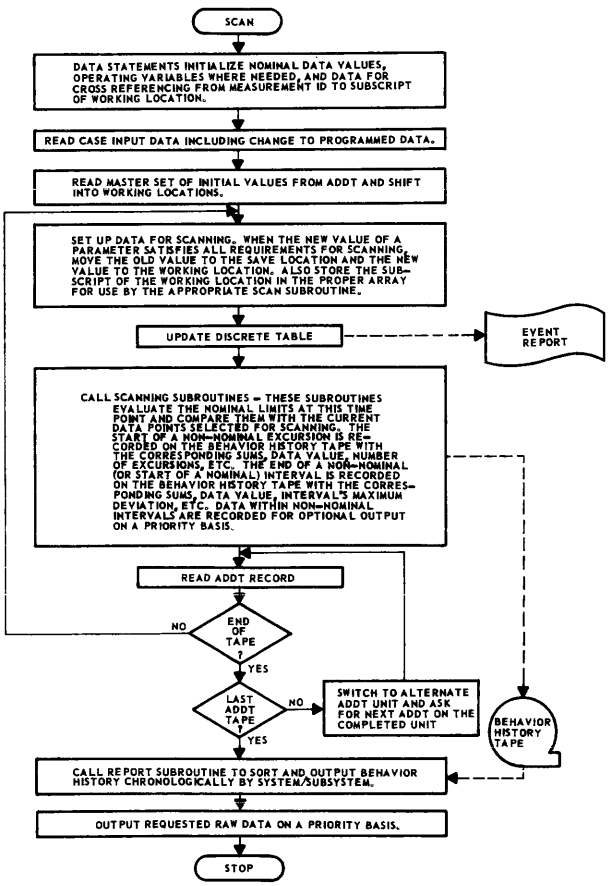


Figure 3—Generalized flow diagram

of the Special Computation Modules and the base line AUTO SCAN program, thus providing for more efficient utilization of CPU time. In addition, use will be made of mass storage devices to reduce requirement for computer core storage.

PROGRAM OUTPUT

There will be two basic outputs from the AUTO SCAN program; discrete events and out-of-limits intervals (flags). All the hard-copy output of the program will be in the form of tabulations. The information provided on each discrete event occurrence is as follows:

Measurement Number
Measurement Title
Time of Detection
True or False Indication

The information provided on each out-of-limits interval is as follows:

Measurement Number
Measurement Title
Time of Detection—This is different from the start of the out-of-limit interval in that a persistency factor is applied to minimize the effect of noise.
Time of Interval Start—When measurement initially goes out-of-limits.
Time of Interval Stop—When measurement returns within limits.
Upper Test Limit
Lower Test Limit
Number of Data Points Outside the Limits
The Maximum Excursion from the Limits
The Average Value While Outside the Limits

A sample output of the AUTO SCAN program for discrete is shown in Figure 4 and for flags is shown in Figure 5. On this particular test run, there were more than three times as many discretes detected as flags raised. The data used for the test run was a magnetic tape generated during the quality assurance checkout tests for the ATM only. Even with this enormous reduction in volume of data (approximately 10^3) to be analyzed, additional effort is required to further reduce the volume of data to manageable proportions.

ADDITIONAL DEVELOPMENT NEEDS

Research in two areas could lead to additional reduction in volume of data to be analyzed: (1) Assessment of filtering feasibility and streamlining the AUTO SCAN modules and (2) development of onboard data redundancy removal and scanning. This second item merits some additional discussion.

Only meaningful data should be transmitted from the spacecraft to the ground station to reduce the load on

DAY 126 PPR - 36433 09:13:40 TO 09:29:50 GMT AUTOMATIC SCAN 31 AUG 72 PAGE
ASTR-II

			TIME OF DETECTION	START STOP	LTV UTV	NPTS	MAX AVG
.	ACQUISITION OF SIGNAL TEX			9:13:40			
KD395-702	ASAP TAPE RECDR NO. 1 RECORD	TRUE	9:13:40				
KD396-702	ASAP TAPE RECDR NO. 2 RECORD	FALSE	9:13:40				
KD407-702	ASAP O/P TO RF ASSY NO. 1 RT PCM TO RF ASSY	TRUE	9:13:40				
KD392-702	RT PCM TO RF ASSY NO. 1 ASAP O/P TO RF ASSY	TRUE	9:13:40				
KD393-702	RT PCM TO RF ASSY NO. 1 AND NO. 2	FALSE	9:13:40				
KD394-702	ASAP O/P TO RF ASSY NO. 1 AND NO. 2	FALSE	9:13:40				
KD519-702	ASAP TAPE RECORDER NO. 1 P/B	FALSE	9:13:40				
KD520-702	ASAP TAPE RECORDER NO. 2 P/B	FALSE	9:13:40				
KD395-702	ASAP TAPE RECDR NO. 1 RECORD	FALSE	9:14:19				
KD519-702	ASAP TAPE RECORDER NO. 1 P/B	TRUE	9:14:19				
KD395-702	ASAP TAPE RECDR NO. 1 RECORD	TRUE	9:20:26				
KD519-702	ASAP TAPE RECORDER NO. 1 P/B	FALSE	9:20:26				
KD395-702	ASAP TAPE RECDR NO. 1 RECORD	FALSE	9:20:33				
KD396-702	ASAP TAPE RECDR NO. 2 RECORD	TRUE	9:22:27				
. . . .	LOSS OF SIGNAL TEX			9:29:59			

Figure 4—Discretes

both these systems and amount of data that would have to be processed by a ground based program such as AUTO SCAN. One way to accomplish this is by the use of algorithms. These algorithms should be developed for onboard data redundancy removal and should automatically scan the spacecraft system parameters as a minimum prior to transmission to the ground site. Criteria for removal and scanning algorithms are:

1. Redundancy removal techniques should have the capability of eliminating both totally redundant and near redundant data from the data stream.
2. Practical pattern recognition schemes should provide for the storage and transmission of the basic data characteristics rather than the total data stream.
3. Automatic scanning techniques should be capable of detecting data that falls outside a prespecified corridor and transmitting only this data.

DAY 126 PPR - 36433 09:13:40 TO 09:29:50 GMT AUTOMATIC SCAN 31 AUG 72 PAGE
APOLLO TELESCOPE MOUNT
ENVIRONMENTAL CONTROL SYSTEM
THERMAL CONTROL SYSTEM

		TIME OF DETECTION	START STOP	LTV UTV	NPTS	MAX AVG
.....	ACQUISITION OF SIGNAL TEX		9:13:40			
CD286-703	TEMP, M/W OUTLET OF HEATER	9:13:55	9:13:40 9:29:58	9.5 12.2	3910	27.6 26.7
CD283-703	TEMP, EXT SURFACE OF RAD PAN NO. 9	9:13:55	9:13:41 9:29:58	-55.8 10.2	3910	30.0 25.7
CD282-703	TEMP, EXT SURFACE OF RAD PAN NO. 3	9:13:56	9:13:41 9:29:58	-55.8 10.2	3910	30.1 25.9
CD285-704	TEMP, M/N OUTLET OF PUMP PACKAGE	9:13:55	9:13:41 9:29:58	9.5 12.2	3909	29.0 22.9
DDDD3-704	PRESS, DIFF M/W ACROSS CAN PANELS	9:13:55	9:13:40 9:29:58	3.5 8.1	3910	56.6 56.0
CD281-703	TEMP, EXT SURFACE OF RAD PAN NO. 2	9:13:55	9:13:40 9:29:58	-56.0 10.1	3910	31.0 26.3
CD287-704	TEMP, DIF, M/W IN/OUT CAN PAN, MDA	9:13:55	9:13:41 9:29:58	.0 1.6	3910	-.9 -.7
CD295-704	TEMP, EXT SURFACE OF HEATER	9:13:56	9:13:41 9:29:58	9.5 12.3	3910	27.0 25.2
DDDD7-704	PRESS, DIFF M/W ACROSS HEATER	9:13:56	9:13:41 9:29:58	.0 2.0	3909	.0 .0
CD288-703	TEMP, DIF, M/W IN/OUT CAN PAN, SUN	9:13:58	9:13:57 9:13:59	.0 1.6	7	-.6 -.9
CD280-703	TEMP, EXT SURFACE OF RAD PAN NO. 1	9:14:32	9:13:41 9:29:58	-55.8 10.2	3910	84.6 83.8
.....	LOSS OF SIGNAL TEX		9:29:59			

Figure 5—Flags

These particular algorithms would apply basically to the onboard computer and obviously if implemented would perhaps impact both its design and capacity.

CONCLUSION

The ATM Baseline AUTO SCAN program has been developed and has received considerable testing during the August-January 1973 time period. Several Special Computation Modules are currently being developed and will be added to this baseline program to provide full capability for this Skylab module. Both the ATM AUTO SCAN and the AM AUTO SCAN subprograms have been completed, including the necessary checkout. The two subprograms have been used during Skylab ground checkouts and will be updated based on the results of its operation during these tests.

The final test of the AUTO SCAN program will occur during the mission. The program will be refined during the mission as experience is gained and will become a primary system analysis tool for future missions. The developers of the system are enthusiastic about the capability of AUTO SCAN and are already considering its use to handle the masses of data created in civil sectors as in the medical and environmental fields.

APPENDIX

Requests—Requirements submitted by the potential users of the AUTO SCAN program which are punched onto cards and subsequently onto magnetic tapes.

Calibration Data—Measurement calibration tape which is also transmitted via data lines from other Centers and used to convert data to engineering units.

Compact Tape—Magnetic tape consisting of the requirements and specific instructions to be used in subsequent subprograms.

ADDT—All Digital Data Tape—Telemetered measurements reformatted, chronologically ordered with redundant ground station coverage removed and onboard tape recorder data inserted where data gaps occur.

Special ADDT—ADDT created by another major program which can be directly input to the AUTO SCAN program without the necessity of using an ADDT read routine which saves core storage and computer time.

Behavior History Tape—Magnetic tape containing the information relative to the time interval when the measurement exceeded the out-of-limit tolerances.

BHT Keying—Output which indicates to subsequent programs that either correlated or other measurement data either in tabulation or plots are required during the out-of-limit intervals provided on the BHT.

Statistical Data Tape—A magnetic tape that will contain data required in the event that some statistical calculations are needed.

SPIDAC—Specimen input to digital automatic computer

by ROBERT S. LEDLEY, H. K. HUANG, THOMAS J. GOLAB, YESHWANT KULKARNI, GERARD PENCE and LOUIS S. ROTOLO

National Biomedical Research Foundation
Georgetown University Medical Center
Washington, D.C.

INTRODUCTION

The SPIDAC system is an automatic optical-microscope scanner for on-line input of pictorial data from a glass slide directly into the core memory of a digital computer. In this paper we describe both hardware and software systems and give a specific application. The major hardware subsystems include the optical microscope with highspeed automatic stage movers and an instantaneous continuous automatic-focusing scheme, high-resolution video reading circuitry, a video memory, and interfacing circuitry. The software system includes programs to automatically move the microscope stage with an accuracy of 1.25μ to any point on the slide, to scan and digitize pictorial data, to examine the scanned image for areas of interest, and then to determine if the pictorial data is amenable to analysis by the digital computer. After the slide is placed on the microscope stage, the system is fully automatic, ending when the entire slide area has been completely examined.

The system can operate in two modes. In the first, the entire slide is scanned and the coordinates of areas of interest are determined, to be used for later detailed examination of the slide. In the second mode, as the areas of interest are determined, they are automatically centered in the optical system and immediately analyzed.

In this paper, we present a detailed discussion of the use of the SPIDAC for automatic chromosome analysis. We also describe the utilization of our developed interactive graphic system (MACDAC) in the same application.

Background

Since the discovery of the compound microscope by Anton van Leeuwenhoek in the 17th century, scientists have been thinking about techniques for obtaining quantitative data from the optical images present in the microscope. Early attempts to elicit and analyze this quantitative data were abandoned, however, because of the vast amount of time and human effort required to produce statistically meaningful data.

With the advent of the automatic digital computer and concurrent developments in electronic and electro-optical devices, new interest was generated in the development of photometric instruments which could aid in obtaining and analyzing this pictorial data. One of the first such attempts was by Ledley and his group whose FIDAC* instrument (Film Input to Digital Automatic Computer)[1,2] was designed to automatically analyze photomicrographs of biomedical interests.

Photometric instruments which have been developed for direct attachment to the microscope can be broadly categorized into three classes: (1) flying-spot scanners, (2) mechanical scanners, and (3) television-like scanners.

Basically, the flying-spot microscope scanner consists of a conventional optical microscope in which a cathode ray tube (CRT) is used as the light source. Examination of a field is accomplished by the CRT scanning spot, which enters the microscope and, by means of an objective lens, is focused onto the glass slide. The light is modulated by the specimen, and the modulated light signal is focused onto a photomultiplier tube (PMT). At any moment of time, the analogue signal output of the PMT represents the optical density of a particular point on the slide and can be converted to digital form for storage, say, on magnetic tape for later analysis by a digital computer. Movement to the next field is accomplished by digital stepping motors that are attached to the microscope state.

One of the first such instruments was developed by Professor J. Z. Young and his colleagues at University College, London, in the early 1950's. The instrument was used to count red blood cells and nerve cells. A similar instrument of this type, called CYDAC (CYtophotometric DAta Conversion), was built by Airborne Instruments Laboratory for preliminary screening of cytological smears.[3] A more advanced version of this unit was used by Mendelsohn and his group in the Department of Radiology of the University of Pennsylvania for research on leukocyte and chromosome classification.[4] The CYDAC,

* Trademark Reg. U.S. Pat. Off.

however, did not include automatic slide-movement and focusing provisions.

The mechanical types of scanners can be subdivided into three classes: (*i*) light chopper, (*ii*) mirror galvanometer, and (*iii*) moving stage. In the light-chopper system, a constant uniform source of light illuminates an area in the object plane. A mask, which determines the type of scanning pattern, is placed in the image plane, and immediately behind the mask is a photodetector. A scanner of this type, using a Kohler lighting system to illuminate an area on a glass slide in the object plane, was described by Sawyer and Bostrom.[5] A microscope objective formed a 60X image in the image plane, at which was placed a Nipkow disk rotating at 1800 RPM and containing 48 scanning holes (equivalent to 2 microns on the object plane). Light transmitted by these holes was focused onto a photomultiplier tube. Two sets of scanning holes were provided on the disk, a spiral pattern being used to generate a raster to examine an entire field and circular pattern being used to generate a line scan. In the later case, the mechanical stage was moved in the perpendicular direction to produce the effect of a raster generation.

The mirror-galvanometer scanning microscope consists essentially of an optical microscope in which two mirrors are placed perpendicular to each other. Light leaving the objective lens passes through a length-correction lens and falls onto the pair of mirrors. The mirrors are mounted on delicate galvanometers which can be deflected independently of each other. The image formed by the lenses is reflected by the mirrors and can be moved electrically by varying the currents through the galvanometers. A small aperture is placed at the image plane, and behind the aperture is a photomultiplier tube. Thus the deflecting mirrors cause the image to move past the fixed aperture in the image plane and create the effect of a raster scan. A system of this type is being used by Dr. Lewis Lipkin of NIH as a tool for biological and medical research, and also to evaluate optical, electronic, and computer system requirements for the solution of specific problems.[6]

The essential element of the moving-stage scanner is a standard microscope to which a mechanized microscope stage, a photometer head, and a detection assembly have been added. Scanning of a field of interest is accomplished by moving the microscope stage by two stepping motors, one each for the x and y directions. In general, three scanning patterns may be selected: (1) single scan lines across any x axis of the object field; (2) a "comb" scan, in which a succession of parallel scan lines of a preselected spacing dissect the sample field, the sampling circuits being inactive during horizontal flyback and movement in the y direction; and (3) a "meander" scan, in which the sampling circuits are active during the back and forth scanning in the x direction and are mute when advancing in the y direction. The photometer head has an optical system for the remagnification of the microscope image. Measuring apertures are placed in the plane of the remagnified image; the light which passes through the measuring apertures impinges upon the photocathode of a photomultiplier tube, which is housed in the detector assembly. A unit of this type was incorporated in the TICAS[7] (Taxonomic Intra-Cellular Analytic System) instrument, which has been used by Wied and his associates of the University of Chicago. A similar unit of this type is commercially available from Zeiss.

The principle of the television-like scanner is simple. A TV camera scans the microscope image, and its output signal is fed to a closed-circuit television monitor and a video-detector unit. Recently several commercial units of this type have become available from various manufacturers (Leitz, Metals Research Instrument Corporation, Zeiss, etc.). These include stepping motors for automatically moving the microscope stage in the x and y directions and small general- or special-purpose digital computers for making simple on-line calculations.

The SPIDAC system which we will describe in this paper is a television-like scanner used under the control of a digital computer. However, it incorporates certain novel features which will be described below.

General description of the SPIDAC system

The SPIDAC system is used in two successive modes. In the first, or *search* mode, the objects of interest are detected under a low-power objective lens, and their centers are located. In the second, or *analysis* mode, a high-power objective lens is used and the microscope stage is automatically moved to center these detected objects in the field for detailed analysis. The complete operational procedure is as follows: The technician puts a slide on the stage, selects the origin position (usually left upper corner of the cover slip), and turns to the low-power objective lens. He presses the "start" button, and the digital computer automatically directs the SPIDAC to move the stage past field after field and strip after strip continuously until all strips, each with a fixed number of fields, have been searched. As the microscope slide is being searched, the software program finds "covers," from which the centers of objects of interest are automatically located in each field; the coordinates of the object centers are saved for the analysis mode. The software also includes an optional visual display, which shows all the covers of the objects of interest for each field on a TV screen and on an associated interactive-graphics device, the MACDAC. Mode 1 terminates when all strips have been searched.

Mode 2 starts with the technician switching the objective lens from low power to high power* and pushing the start button for the analysis mode. The computer then directs the microscope stage to move successively to the centers of the objects of interest found in Phase 1. The magnified image of each such successively located object is displayed on the TV screen. The technician judges whether the field is good or bad; if bad, it is rejected and the next field is located; if good, the analysis button is

* This step will be automated in the future.

pushed which initiates a high-resolution scan and the detailed analysis of the field. When this is finished, the next object field is automatically located, and so on, until all the objects of interest have been processed. The software program then moves the microscope stage back to its original position and terminates the operation.

HARDWARE DESCRIPTION OF THE SPIDAC

The hardware of the SPIDAC system consists of three major component subsystems: First there are the essentially mechanical components, i.e. the light microscope with a motorized stage and an automatic focusing system, including the associated electronic controls. (We sometimes refer to this section as the "SPIDAC instrument.") The second subsystem, called the VIDIAC, handles the video images of the microscope fields; it consists of a vidicon camera for scanning the microscope image and converting it into an electronic signal, a TV screen to display the image, and a variable-speed electronic image-storage device, with the associated sweep generators and control circuits. And finally there is the MACDAC interactive-graphics unit, which converts the video signal to a digital signal, or series of numbers, at a speed slow enough to be accepted by the computer, and transmits this digital signal to the computer's core memory; displays the digitized pictures from the VIDIAC, or computer-generated pictures, or the location (and path, if desired) of an operator-controlled cursor on a storage cathode ray tube; and transmits to the computer the location coordinates of this cursor, together with code numbers that tell the computer program what use is to be made of these coordinates, at times specified by either the operator or the computer itself. This last feature allows the operator to edit a picture stored in the computer's memory to correct for faults and artifacts on the microscope slide.

Light microscope with motorized stage and automatic-focusing system

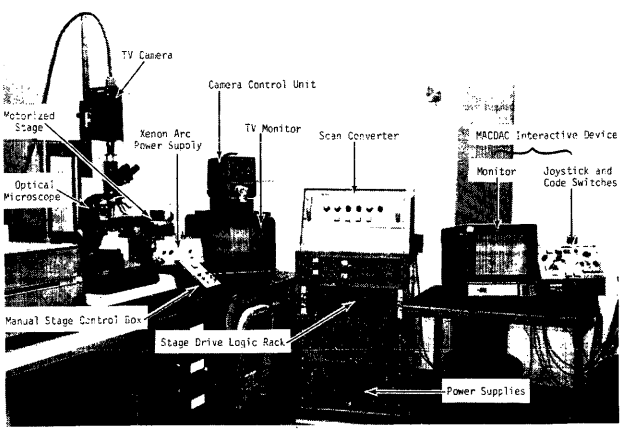


Figure 1—SPIDAC hardware system

The SPIDAC system (see Figure 1) uses a modified Leitz optical microscope whose stage and slide holder have been replaced with a motorized stage and a specially designed slide holder suitable for automatic focusing. The SPIDAC normally uses a 22× oil-immersion objective in the search mode and a 100× oil-immersion objective in the analysis mode, both lenses manufactured by Leitz; spacer rings are used to make the two objectives parfocal. The lighting system allows the use of either an incandescent lamp or a xenon-arc lamp, the latter being used almost exclusively in the SPIDAC system. When a change is made from one objective to the other, the combination of neutral density filters in front of the arc lamp is also changed so that the level of light intensity remains unchanged. The microscope, together with the motorized stage assembly, is mounted on a heavy aluminum plate, and the entire assembly rests on vibration isolators.

A high-precision Aerotech x-y stage is used in the SPIDAC because of its accuracy and repeatability, and two Sigma stepping motors drive it in the x and the y directions. Each motor step moves the stage 2.54 microns (0.0001") with an accuracy and repeatability to within + 1.27 microns. The stage can move in either direction at a directions. Each motor step moves the stage 2.54 microns (0.0001") with an accuracy and repeatability to within ±1.27 microns. The stage can move in either direction at a maximum speed of 650 steps, or 0.165 cm., per second, The specimen slide is held in a specially designed holder, which makes it possible to move the slide with negligible transmission of vibrations when the motors are stepping. The SPIDAC atuomatic focusing system is mounted on the same block that is moved to effect coarse and fine focusing operations, so that it is very easy to focus a slide initially. The focus thereafter is maintained dynamically and with an accuracy adequate for all magnification, including 100× objective.

VIDIAC system

The complete SPIDAC system is diagrammed in Figure 2. The vidicon camera scans the microscope field and displays its image on the monitor (a grey-level display) at a normal TV rate. The operator may assume manual control of the system by pressing a button on the manual control box; he can then position the slide by observing the monitor. He can also adjust the vidicon beam current and target voltage using the monitor display. In this model of the SPIDAC, the vidicon runs only at a TV rate, so that a silicon video memory (SVM) is needed as an intermediate stage between the vidicon and the computer. Therefore, as part of the set-up procedure, the operator writes one frame of the video picture onto the silicon video memory (in 1/30 of a second) and views the SVM output on the TV screen, adjusting the modulation and bias into the SVM tube to their optimum levels. These levels will remain correct throughout the slide searching and analysis.

The operator may now turn control over to the computer, which proceeds with the search for objects of interest.

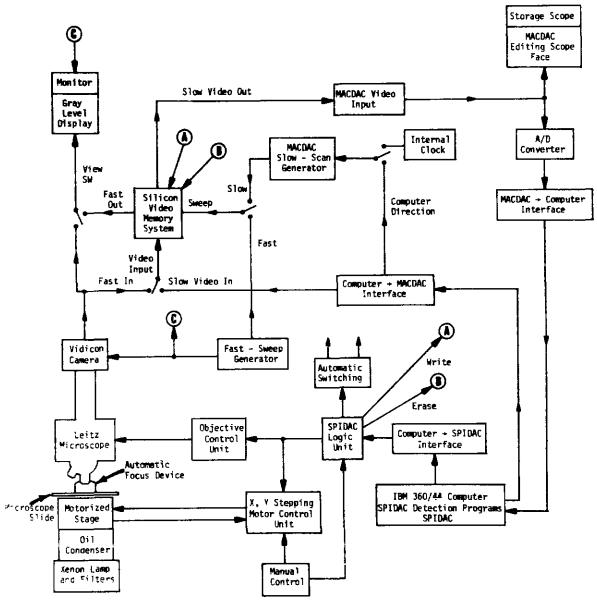


Figure 2—Block diagram of SPIDAC

The instrument momentarily displays each field on the TV screen as it is automatically moved into view by the computer. The stage is moved a field at a time, under computer control. Four computer commands accomplish this: (1) move x forward, (2) move x backward, (3) move y forward, and (4) move y backward. When each new field is in position, the SVM sweep is switched to the fast mode (1/30 sec. per frame), the SVM is erased, and the video picture is written onto the target of the SVM. Then the sweep is set to the slower, MACDAC speed and the picture is read from the SVM target into the computer's core memory.

The vidicon camera is a high-resolution Telemation television camera, whose sweep circuits are synchronized with the fast-sweep generator of the SVM system. The SVM tube is similar to a vidicon tube, except that the "target" of the SVM is made of a small (approximately 1 inch) disk covered with minute, discrete areas of silicon, which act as tiny capacitors to store an electrostatic representation of the video picture. After an image has been stored on the target of the SVM at one speed, it may be read or converted to a video signal, at the same or any other speed. Thus it may be read at high speed (TV rate) and displayed on the monitor or scanned at the slower MACDAC rate for input into the core memory. A great deal of switching is necessary to operate the SVM in each of its three modes, Write, Read, and Erase. The computer is able to initiate the three modes of the SVM so that all operations of the SPIDAC occur automatically.

MACDAC interactive device

The interactive device used in the SPIDAC system is called the MACDAC (MAn Communication and Display for an Automatic Computer, see Figure 3).[8,9] It serves two functions for the SPIDAC system: (1) displaying a field during the search mode, and (2) editing classical chromosome karyotyping during the analysis mode. The MACDAC system allows a rapid man-machine interaction in the automatic analysis of chromosomes. It enables a human operator to edit the pictures before they are put into the computer and provides a convenient display as the analysis progresses. From the control panel, individual spots on the (nominal) 700×500 spot picture can be addressed by moving a "joystick" lever. The position of the lever coincides with a cursor, or visible spot, which can be seen, along with the picture, on a Tektronix/611 storage oscilloscope. If desired, the operator can *store* the path of the cursor on the scope face.

When the operator pushes a button, the picture being scanned by SPIDAC is *stored* on the storage-scope CRT. The operator can then move the cursor to an object in the stored image which he recognizes as, say, a piece of "dirt" in the picture. He selects the proper switch for a 4-bit code for an unwanted particle and pushes the "READ SPOT" button. The MACDAC logic interrupts the computer, from its work of analyzing the previous picture, to read in a 24-bit word which contains (a) four bits of notational information, e.g. "dirt," (b) 10 bits for the x coordinate of the spot, and (c) 10 bits for the y coordinate. This editing information is used as an aid to analyzing the picture in the VIDIAC, which will be the next picture read into the computer. The operator continues to point to various trouble spots and sends the coordinates and the proper codes to the computer. After each spot is read into the computer, the operator may make (store) a mark on that spot for his own reference. When the operator has finished sending the editing data, he pushes the END button, which turns control of the storage scope over to the computer. Under computer control, the MACDAC has the capability of lighting any designated spot on the storage scope, so that the computer can also display the results of the analysis on the MACDAC storage scope.

The MACDAC system contains a number of other features, such as a bias control to center any portion of

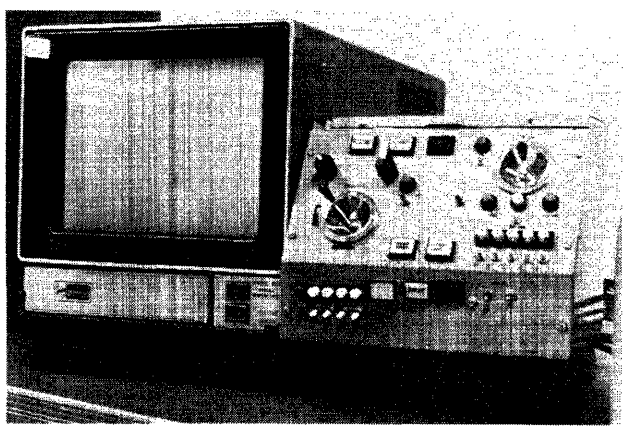


Figure 3—MACDAC display screen and control panel

the picture on the CRT and a "*zoom*" control to expand the selected portion. These features enhance the presentation on the CRT area, which is 8″×6 1/2″.

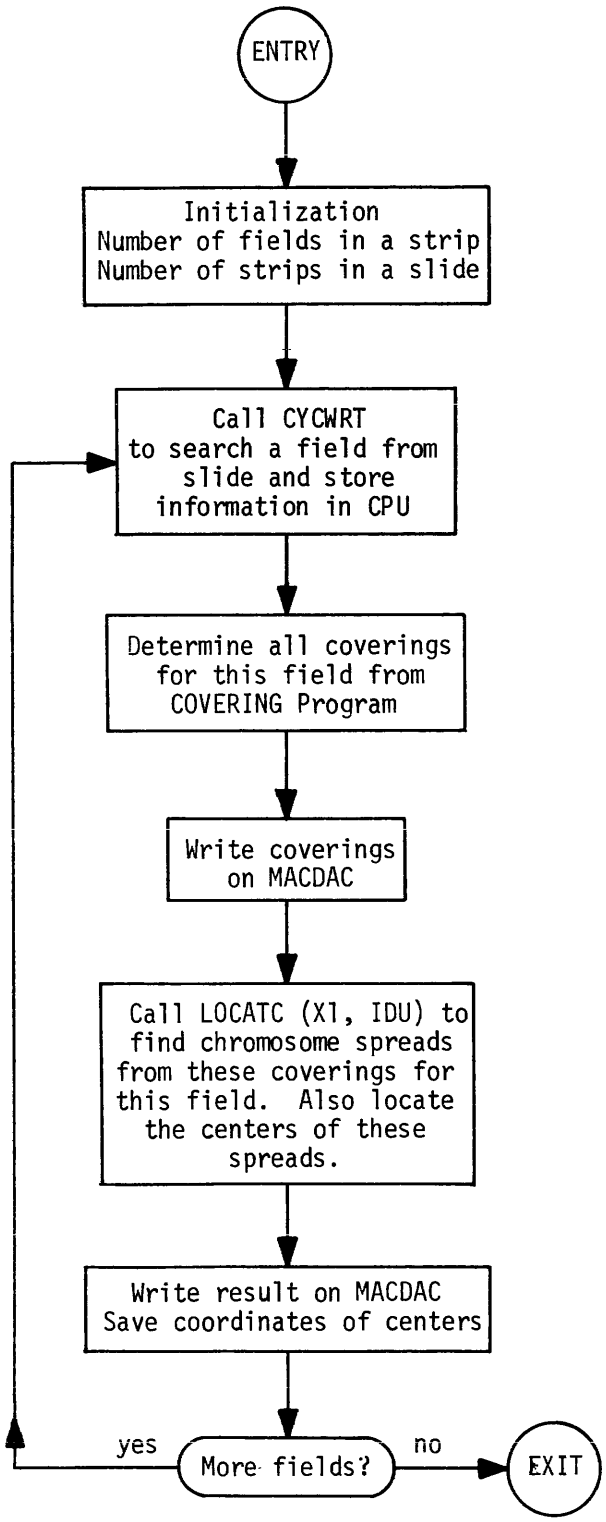


Figure 4—Flow chart of search mode

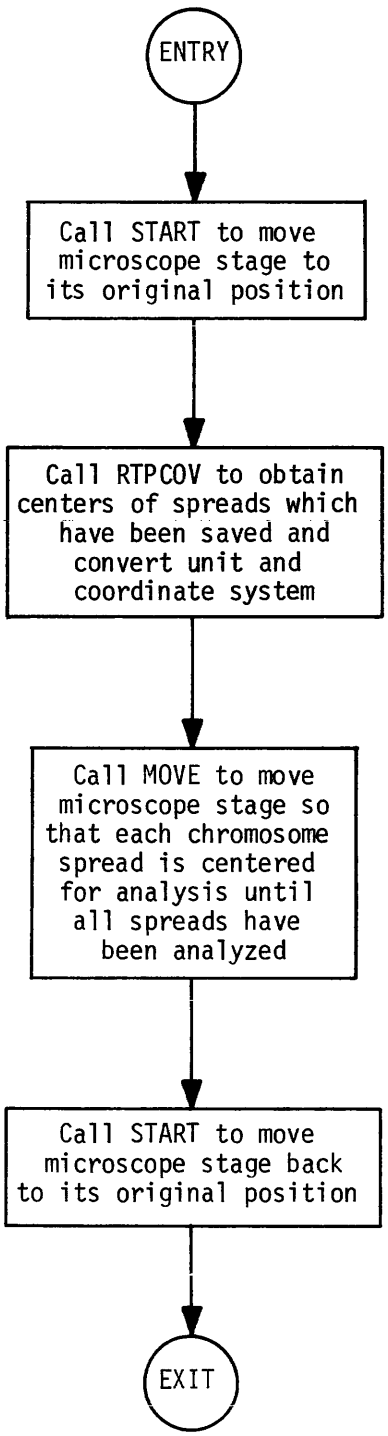


Figure 5—Flow chart of analysis mode

SOFTWARE DESCRIPTION

The software package was originally designed for locating metaphase cells on a glass slide and for karyotyping chromosomes.[10] This goal has been accomplished, and routine runs are being made. The system can also be used for automated cytology, automated bacteriological microscopy, and so forth. Figures 4 and 5 show the flow

charts of the software system. In the search mode, the total scan area (TSA) i.e. the portion of a slide which can be traversed by the hardware is scanned in strips 310μ wide, each strip being composed of fields 310μ by 250μ. The *origin* is the left "upper" corner of the TSA; the *positive x-axis* is taken as the line starting from the origin and extending longitudinally (toward the right) along the upper edge of the TSA; the *positive y-axis* is taken as the line starting from the origin moving laterally (downward) along the left edge of the TSA.

After a field is scanned in the search mode, a command is sent by the program through the computer-interface channel to the stepping motors to move the stage a certain number of steps in the x or y direction, so that either the next consecutive field or the first field of the next strip can be centered under the objective lens, depending upon the location of the last field. The program has an option in controlling the total movement of the stage in either direction. To scan and digitize the picture, the program sends a "READ" command to the video hardware. The hardware then begins transmitting digitized grey-level values to the computer, 24 bits at a time. Each line of picture information constitutes a discrete "record" of data, so that the program can control the placement in memory of each line individually. In this manner, the complete picture, as stored in the memory, consists of a nominal 500 lines of 700 spots each. Alternatively, a part of the picture can be read-in in greater detail and placed on a disk of the computer.

Depending on the nature of the applications, methods for locating objects of interest are different. The SPIDAC system has been designed in such a way that this part of the system is an independent component; i.e., different programs for different applications may be inserted prior to the execution time. The method for locating metaphase cells will be described in detail below.

After objects of interest have been located in a field, the coordinates of the centers of these objects are recorded. The next field is then scanned and more objects of interest are located, this procedure continuing until the total scan area has been searched. The stage then moves back to the origin. When everything is ready, the operator initiates the analysis phase and the computer moves to the centers of the objects of interest located earlier for detailed analysis. During this cycle of operation, proper focus is automatically maintained by the hardware. As each object of interest is centered under the microscope and displayed on the VIDIAC's TV screen, the operator decides if it is suitable for analysis. If it is, the analysis mode of the software package is initiated and the proper analysis will be performed on the object automatically.

APPLICATION TO CHROMOSOME ANALYSIS

As an example let us consider in detail the method of locating the metaphase cells. In a binary digitized field, we define a possible chromosome spread or an object consisting of at least n adjacent units (but no more than $2m+2-n$ units) separated from other groups of units by at least n zeros on either side. The parameters n and m are chosen as those most typical of the width of a chromosome and the spacing between chromosomes in a metaphase spread; they depend on the magnification of the image and the detail with which it is read in by the SPIDAC. A *covering* is, by definition, a horizontal string of at least p such objects, where the distance between two neighboring objects must be less than q points (p and q again being parameters typical of a metaphase spread as read in). The starting coordinate of a covering is the left-hand coordinate of its leftmost object and the ending coordinate of this covering is the right-hand coordinate of its rightmost object. The difference between these two coordinates must be greater than some typical r but less than some typical s. A *spread* then becomes a collection of at least two vertically proximate coverings. Its size must lie within a square of certain dimensions. which are determined by the magnification used. The *center* of the spread is determined as the arithmetic means of the maximum coordinates of all the coverings.

To reduce processing time, the program for finding metaphase spreads uses a scanned field containing only every ith* line of the complete field. For each line, the program looks for objects which are candidates for metaphase spreads and tries to join them into coverings. This procedure continues until all the possible coverings are found and recorded for each entire field. The chromosome spreads are then formed from these coverings. Thus the program for locating metaphase cells in each field is separated into two parts; cover detection and chromosome-spread detection.

As soon as the total scan area has been searched for possible chromosome spreads, the program requests permission from the operator to turn control over to the analysis mode. The analysis-mode programs actually serve as a driver for our already existing chromosome-analysis programs.[11,12] As such, their main purpose is to move the microscope stage to the locations where suitable chromosome spreads have been found by the search-mode programs. Once the microscope stage has been moved, they turn control over to these detailed-analysis programs, which analyze the chromosome spread and return control to the analysis-mode programs when they are finished. The analysis-mode programs move the microscope stage and the complete procedure repeats itself to the end of the list of spreads.

Figure 6 shows some metaphase cells detected by the method. Figure 6a shows two metaphase cells; Figure 6b shows the corresponding coverings and spreads for each cell. Figure 6c shows an enlargement of a different field, and Figure 6d shows the covers found by the program.

* This parameter is a property of the hardware and can be selected beforehard.

Classical karyotyping

The use of the MACDAC interactive device essentially corrects all errors in the chromosome analysis which are due to artifacts and faulty preparation and enables emendations to be made in the final analysis, not only in classification but in measurement.

Six editing codes are now used: (1) separate; (2) connect; (3) erase; (4) partially fence, from end fence posts to border; (5) completely fence, connecting the last fence post to the first; and (6) slice, separating two touching chromosomes. The fence routine joins the "fence-post" coordinates, which are read in from the MACDAC, with a line of spots of special value 1. Then the fenced area is specially bounded using special values 2 and 3, so that when the internal programmed search is carried out, it will be disabled in the fenced region. The erase routine bounds the object surrounding the point read in with 2s and 3s, thereby disabling the search routine for that object. The separating routine requires the coordinates of two points to be input, and changes the picture points to 0 along a straight line joining these given points. The connecting routine also joins two coordinates, but with points of unit value. The slice is a separating routine which analyzes the area around a single input point at which two chromosomes touch and separates them along a "best" line by changing the appropriate spots to 0.

After the picture in the memory has thus been completely edited, the individual chromosomes are located by the internally programmed search, their centromeres are located, and their dimensions are measured. The computer-analysis programs then perform a karyotyping of the spread, using the arm-length, arm-length-ratio, area, and area-ratio measurements as the criteria for classifying the chromosomes into each of ten human chromosome groups. These ten include the seven major types, *A, B, C* $+X$, *D, E, F,* and $G+Y$, and also distinguish $A1$, $A2$, $A3$, and $E16$. When the analysis if completed, the data on chromosome arm length, area, perimeter, and position in the cell are saved for statistical analysis and abnormality considerations. A typical karyotype, as displayed on the MACDAC, is shown in Figure 7.

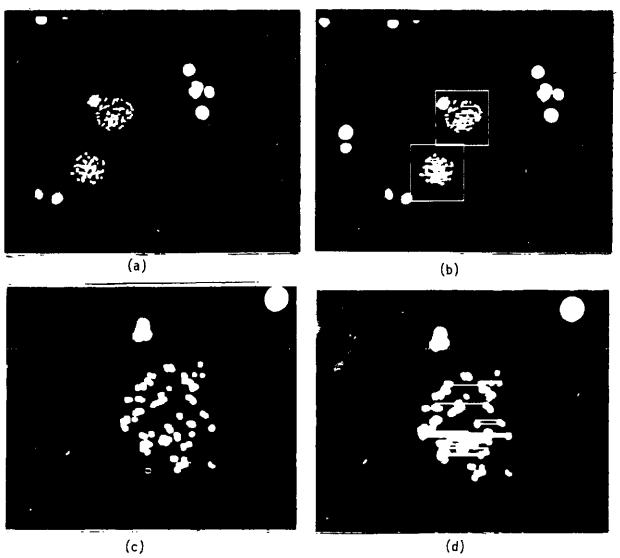


Figure 6—(a) Two cells in metaphase; (b) corresponding covers and spreads; (c) another metaphase spread at greater enlargement; (d) covers found by the program for this cell

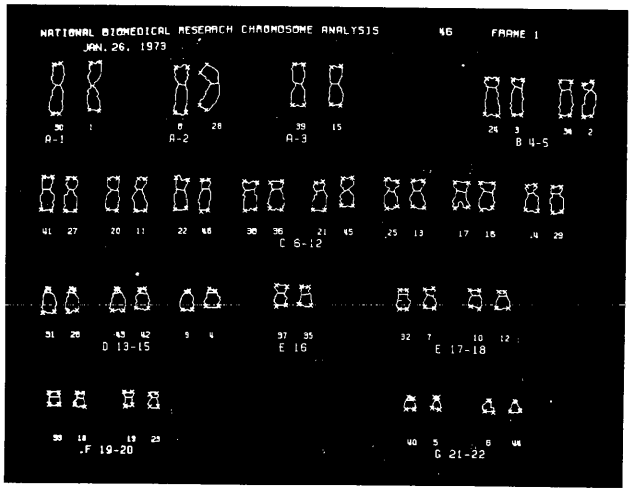


Figure 7—MACDAC display of human chromosome karyotype

Banding analysis

The primary purpose of the banding-analysis approach to karyotyping is to refine the already successful classical result and to further classify the chromosomes into their exact homologues through detailed analysis of their banding patterns. Briefly, banding pertains to the differential staining of chromosomes such that different regions along the length of each chromosome take on differing amounts of stain, according to their differing chemical subcompositions. This results in a light-dark pattern which is characteristic of each of the 24 types of human chromosomes and enables positive identification of each type, rather than just classification into ten groups.[12]

Previous to the classical analysis described above, using a 16-grey-level picture, the program computes the average grey levels along lines perpendicular to a parabola fitted to the long axis of each chromosome. The profiles so constructed are then stored and the classical karyotyping is performed, using only chromosome dimensions. The profiles within each of the seven main groups are then compared with stored profiles previously found to be typical of the 24 types, using chromosomes identified by an expert cytologist; the present chromosomes are then rearranged within groups as necessary according to the exact individual types. The rearrangement and comparison procedure stated so easily in the last sentence is of course a major problem requiring a great deal of analysis, although a large measure of success has been met already

using both Fourier-coefficient maximum-likihood methods and ad hoc descriptions of the grey-level profile curves.

CONCLUSION

An automatic optical-microscope scanner has been described for on-line input of pictorial data from a glass microscope slide directly into the core memory of a digital computer. An application of the system to chromosome analysis was also given. At the present time, with the SPIDAC system, we can automatically search to locate all chromosome spreads on a glass slide, and scan to completely analyze the chromosomes of suitable spreads, including some interactive editing of the picture.

ACKNOWLEDGMENTS

This work was supported by grants HD-05361, RR-05681, GM-15192, and GM-10797 from the National Institutes of Health, Public Health Service, Department of Health, Education, and Welfare, to the National Biomedical Research Foundation.

REFERENCES

1. Rotolo, L. S., Wilson, J. B., Ginsberg, M. D., Ledley, R. S., "Digital Computer Picture Processor," *Proceedings of the 16th ACEMB*, November 18-20, 1963, Baltimore, Maryland.
2. Golab, T., Ledley, R. S., "FIDAC—Film Input to Digital Automatic Computer," *Pattern Recognition* 3(2), pp. 123-156, 1971.
3. Bostrom, R. C., Holcomb, W. G., "CYDAC—A Digital Scanning Cytophometer," *IEEE International Convention Record*, Part 9, pp. 110-119, 1963.
4. Mendelsohn, M. L., Mayall, B. H., Prewitt, J. M. S., Bostrom, R. C., Holcomb, H. G., "Digital Transformation and Computer Analysis of Microscopic Images," *Advances in Optical and Electron Microscopy*, Vol. II, V. Cosslett and R. Barev, eds., Academic Press, 1968.
5. Sawyer, M. S., Bostrom, R. C., "A New Nipkow-Disk Scanner for Accurate Cytological Measurements," *IRE Convention*, Technical Session on Medical Electronics, 1958.
6. Stein, P. G., Lipkin, L. E., Shaprio, H. M., "Spectre II—General Purpose Microscope Input for a Computer," *Science*, 166, pp. 328-332, 1969.
7. Wied, G. L., Gahr, G. F., Bartel, P. H., "Automatic Analysis of Cell Images by TICAS," *Automated Cell Identification and Cell Sorting*, G. L. Wied and G. F. Bahr, eds., pp. 195-260, Academic Press, 1970.
8. Golab, T. J., "MACDAC—An Inexpensive and Complete Biomedical Input and Output Display System," *Proceedings of the 23rd Annual Conference on Engineering in Medicine and Biology*, November 15-19, 1970.
9. Pence, G., "MACDACSYS—A Programming System for Manual Assist to Automatic Chromosome Analysis," *Proceedings of the 23rd Annual Conference on Engineering in Medicine and Biology*, November 15-19, 1970.
10. Huang, H. K., *A Preliminary Report on the SPIDAC Software Package*, NBR Report No. 72-501-10797/05361, 1972.
11. Ledley, R. S., Golab, T., Pence, R. G., "An Inexpensive On-Line Interactive Computer Display," *Proceedings of the IEEE Regions III Convention*, April 26-28, 1971, Charlottesville, Virginia, pp. 369-374.
12. Ledley, R. S., Lubs, H. A., Ruddle, F. H., "Introduction to Chromosome Analysis," *Computers in Biology and Medicine*, Vol. 2, pp. 107-128, 1972.

A method for the easy storage of discriminant polynomials

by RANAN B. BANERJI

Case Western Reserve University
Cleveland, Ohio

INTRODUCTION

One of the purposes of feature extraction is to save on the memory required to store the descriptions of the patterns learned. It is, however, the opinion of this author that a far more important function of feature extraction is to attach statistical significance to the patterns learned[1] and to change the measured variables for future experiments in the same pattern recognition environment.

The present paper provides a method for fulfilling the above first or "simplification" aspect of feature extraction. Unfortunately, the nature of the method is such that very little light seems to be cast on the latter "significance" aspect of feature extraction.

Since the method involved deals heavily on the theory of finite fields, an area of mathematics not of wide usage in pattern recognition, we shall include in the next section a short tutorial on the subject together with an explanation of the method. The third section will discuss some of the algorithms involved and give an estimate on the memory saving and the computational work involved.

FIELDS

A *field* is a set of elements on which addition, subtraction and multiplication can be carried out and division by any non-zero element is possible. Fractions and real numbers are examples of fields. Positive integers are not fields since 4 cannot be subtracted from 2. Positive and negative integers do not form fields either since 3 cannot be divided by 2.

Fractions and reals are infinite in number. On the other hand, a good example of a finite field is the set of integers $\{0, 1, 2, \ldots (p-1)\}$ (where p is some prime number) in which all addition and multiplication is carried out "mod p." That is, the sums and products, if they exceed $(p-1)$, are divided by p and the remainder taken as the result. As an example, the field of integers mod 3 have the following "addition" and "multiplication" tables

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Finite fields having q elements exist only if q is prime or is an integral power p^n of some prime integer p. These latter fields, however, do not have the "modulo q" structure that the prime fields have. For instance, division by 2 is impossible modulo 8—no integer less than 8, multiplied by 2, yields 3, for instance, showing that $\frac{3}{2}$ is undefined modulo 8. The construction of the addition and multiplication tables of such "prime-power" fields need some detailed explanation, which we proceed to give below. The methods are intimately tied to polynomials, the major topic of this paper.

Given any field F, one can form polynomials in a "variable" x with coefficients from F. Polynomials are added and subtracted "componentwise" as usual. That is, the sum of

$$a_0+a_1x+a_2x^2 \ldots a_nx_n{}^n$$

and

$$b_0+b_1x+b_2x^2 \ldots b_nx^n$$

is

$$(a_0+b_0)+(a_1+b_1)x+(a_2+b_2)x^2 \ldots +(a_n+b_n)x^n$$

Note that both a_n and b_n need not be non-zero—we have made the "degrees" equal only for convenience. The next thing to be noted is that for our present purposes, a polynomial can be merely considered as n-tuples of field elements, the x being a mere "placeholder." However, the polynomial format for exhibiting the n-tuples takes great mnemonic significance when one defines multiplication of polynomials by saying that the product of the two polynomials above is

$$a_0b_0+(a_1b_0+a_0b_1)x+(a_2b_0+a_1b_1+a_0b_2)x^2+ \ldots a_nb_nx^{2n}$$

yielding a $2n$-tuple.

Polynomials do not form fields: although addition and subtraction is possible, division cannot always be performed: $2+x$, for instance, cannot be divided by $3+x$. However, just as finite fields can be produced by performing all operations modulo a prime, polynomial fields can also be formed by performing addition and (especially) multiplication module a prime or *irreducible* polynomial, i.e., one which cannot be factored into polynomials with coefficients from the same field.

In the field modulo 3, for instance we find that

$$(x+1)(x+1)=x^2+2x+1$$

$$(x+1)(x+2)=x^2+2$$

$$(x+2)(x+2)=x^2+x+1$$

and these are the only polynomials which can be factored into other polynomials (we are neglecting the cases where the coefficients of the highest power of x are not 1, since one can clearly "divide these out," i.e., $2x+1$ is the same as $2(x+2)$). Hence, x^2+x+2 (or more conveniently written x^2-2x-1 for future purposes; note that since $1+2=0$, $2=-1$) is an irreducible polynomial. We give below some examples of operations on polynomials mode $x^2=2x+1$. These operations will be restricted to polynomials whose degree is less than 2 and whose coefficients, clearly, come from the field of integers mod 3. As can be seen, there are nine (3^2) such polynomials, all the way from $0=0+0x$ to $2+2x$. $(1+x)(2+x)=2+x^2$ which, on division by x^2-2x-1 yields $2+1+2x=2x$.

The major fact which we shall use for our purposes here is that in any finite field all the non-zero elements can be obtained by raising some element of the field to successively higher powers. For instance, the polynomial $2+2x$, raised to successive powers, yields all the 3^2 elements of the field mod x^2-2x-1 and 3. To illustrate,

$$(2+2x)^2=4+2x+x^2\equiv 1+2x+(1+2x) \quad \text{mod } x^2-2x-1, 3$$
$$=2+x$$
$$(2+2x)^3\equiv(2+x)(2+2x)\equiv 1+2x^2 \quad \text{mod } x^2-2x-1, 3$$
$$\equiv 1+2(1+2x)\equiv x$$

and similarly

$$(2+2x)^4\equiv x(2+2x)\equiv 2x+2(1+2x)\equiv 2 \quad \text{mod } x^2-2x-1, 3$$
$$(2+2x)^5\equiv 1+x \quad \text{mod } x^2-2x-1, 3$$
$$(2+2x)^6\equiv 1+2x$$
$$(2+2x)^7\equiv 2x$$
$$(2+2x)^8\equiv 1$$

Two things are to be noted about this table which shows the eight polynomials as powers of x. The first is that the elements 1 and 2 behave just like they did in the field mod 3, i.e.,

$$2\times 2$$
$$=(2+2x)^4(2+2x)^4$$
$$=(2+2x)^8$$
$$=1$$

These form a *subfield*—and they occur in the right places ($2\equiv(2+2x)^4$) to make this possible. This is true in any field having q^n elements, as we shall illustrate further as we go on.

The second important thing to be noted is that the polynomial $2+2x$ actually does "satisfy" the equation x^2-2x-1, i.e., if its value is "plugged in" for x in the above polynomial, the result is

$$(2+2x)^2-2(2+2x)-1\equiv 2+2x+(2+2x)+2\equiv 0!$$

As a matter of fact, x also satisfies x^2-2x-1 as can be seen from the above table; also the elements of the field can be obtained by successive powers of x itself since $x=(2+2x)^3$; $x^2=(2+2x)^6=1+2x$, $x^3=(2+2x)^9=2+2x$ and so on.

Not all irreducible polynomials have the property that the successive powers of any of its solutions generate all the non-zero elements of a field. The polynomial x^2+1 is irreducible in the field mod 3 and yet the only two elements which satisfy it are 2 and 1 whose successive powers merely generate a subfield.

Now if we can find an irreducible polynomial of degree n in a field of q elements, such that its solution generates all the q^n polynomials of degree less than n by its successive powers (such an element is called a *primitive element*), then we can express any polynomial by an integer—its "logarithm" with respect to the primitive element (which can safely be taken to be x). In the above field $2+2x$ could be represented by 3 and its value as a polynomial could be obtained by dividing x^3 by x^2-2x-1 yielding, as expected, $2+2x$ as a remainder

$$x^2-2x-1)x^3 \qquad (x+2$$
$$\frac{x^3-2x^2-x}{2x^2+x}$$
$$\frac{2x^2-4x-2}{2x+2}$$

The memory saved by using this technique will be discussed in the next section.

Our "trick" for storage of a large number of polynomials involves the discovery of an irreducible polynomial and an element of the field mod this polynomial which is primitive, i.e., whose successive powers generate all the polynomials in the field. For instance, in the field of polynomials with coefficients in the mod 3 field, taken modulo x^2+1, we see that $x+1$ is primitive, since $(x+1)^2=x^2+2x+1=2x$; so $(x+1)^2-2(x+1)-1\equiv 2x-2-1\equiv 0$, i.e., that $x+1$ "satisfies" x^2+2x+1, and therefore behaves just like $2+2x$ in the field of polynomials mod x^2-2x-1.

Once such a primitive element α is found, any polynomial can be represented by its "logarithm" with respect to α. In the field mod x^2+1, for instance, $2x+2\equiv(x+1)^5$ as can be seen by carrying out the following calculation

$$(x+1)^2=2x$$
$$\therefore\ (x+1)^4=(2x)^2\equiv x^2\equiv -1\equiv 2$$
$$\therefore\ (x+1)^5=2(x+1)=2x+2$$

It will be noted how the fourth power of the primitive element is again 2, just as in the case modulo x^2-2x-1.

In the next section we shall discuss an algorithm for finding irreducible polynomials and discuss possible ways of finding primitive elements of polynomial fields and the manipulations needed for converting an integer to its "exponential." In the process we shall be exemplifying a process of constructing fields of polynomials in more than one variable.

COMPUTATIONAL METHODS

Knuth and Alanen[4] have given a method for finding irreducible polynomials with coefficients in a prime field such that its solution is primitive in the polynomial field. The method generalizes to prime-power fields also. Basically it is a search method—but there are two restrictions which reduce the search somewhat.

The first restriction states that the constant term in any polynomial of this nature must be either a primitive element of the coefficient field (if the degree is even) or the negative of one (if the degree is odd). After a polynomial of this nature is chosen, one can raise the variable to successive powers modulo the polynomial $(q^n-1)/q-1$ times (notice that $(3^2-1/(3-1)=4$ and $x^4\equiv 2 \bmod x^2-2x-1$). If the last operation yields a primitive element of the coefficient field, then the structure of the polynomial field is known.

This latter test is somewhat laborious; however, the test can be shortened somewhat if the polynomial is tested for irreducibility first. In this case, one can try any polynomial (including the variable) for being primitive by raising it to successive powers to the largest divisor of $(q^n-1)/(q-1)$. It may be possible, given any prime polynomial $u(x)$, to calculate directly a primitive polynomial in the field modulo u: but this author is not aware of any such method.

Testing a polynomial for irreducibility in a finite field can be carried out by using a modification of the Berlekamp[2] method suggested by Knuth.[5] The method is described by him for a prime field, but can be applied to prime power fields also. We discuss this in what follows, using the nine-element polynomial field of the previous section as the coefficient field. Since the elements of this field are polynomials in x, the results of this section will illustrate the use of the method to polynomials in more than one variable.

Since there are 8 linear polynomials of the form $y+\alpha(x)$ when $\alpha(x)$ is some element of the field of 9 elements, they are $(8\times 4)/2=16$ factorable quadratic polynomials whose constant terms are primitive elements of the coefficient field (only α, α^3, α^5 and α^7 are primitive). There are a total of $9\times 4=36$ quadratics with coefficient for y^2 and a primitive constance term. Hence, the probability that a quadratic with a primitive constant chosen at random will be factorable is $16/32\approx 1/2$. Hence the chances are less than 3 percent that an irreducible polynomial will not be found in 5 trials. Also, since 80 (9^2-1) is divisible by 2, 4, 5, 8, 10, 16, 20 and 40, 72 of the two-variable polynomials in the resulting field will be primitive—so the chance of one chosen at random being primitive is large indeed. Once a primitive element can be found, one can find another polynomial which it satisfies by testing its successive powers for linear independence (the way we found x^2-2x-1 for $x+1$ in Section 2 in the field modulo $x+1$), and this new equation (which, according to the theory of fields is bound to be irreducible) will have y itself as a primitive element.

Let us now describe the test for irreducibility for a polynomial $u(y)$. We shall assume for simplicity that $u(y)$, if it is factorable at all, has no repeated factors. This can be tested quite easily by seeing if $u(y)$ and $(du)/(dy)$ have any common factor by taking their greatest common divisor by the usual method. If this g.c.d. is not 1, then it has repeated factors and hence is not irreducible. If it does have a g.c.d. of 1, we use the following factorization technique.

Take the polynomials $f_i(y)=y^i-y$ modulo $u(x)$ for $i=1, 2, \ldots [n/2]$ where $[n/2]$ is the largest integer less than $[n/2]$. $u(y)$ is irreducible if the g.c.d. of $f_i(y)$ and $u(y)$ is 1 for every i.

It is not as hard as it seems to calculate the polynomials $f_i(y)$ as the large values of q^i might indicate. We can see this as follows. Suppose there was a matrix

$$Q=\begin{vmatrix} q_{00}q_{01}\ldots q_{0,n-1} \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ q_{n-1,0}\ldots q_{n-1,n-1}\end{vmatrix}$$

where $y^{qi}\equiv q_{i0}+q_{i1}y\ldots q_{i,n-1}y^{n-1} \quad \bmod u(y)$

Then if there is a polynomial

$$\omega(y)=a_0+a_1y+\ldots a_{n-1}y_{n-1}$$

Then $\omega(y)^q$ can be readily seen to be

$$a_0{}^q+a_1{}^qy^q+a_2{}^qy^{2q}\ldots a_{n-1}{}^qy^{q(n-1)}$$

since all the product terms in the expansion are multiples of q and $q\equiv 0 \bmod q$ and $u(y)$. Also, since each a is a power α^t of some primitive element $a^q=(\alpha^q)^t\equiv 1$ so that

$$\omega(y)^q=a_0+a_1y^q+a_2y^{2q}\ldots a_{n-1}y^{q(n-1)}$$

replacing x^{qi} by

$$\sum_{j=0}^{n-1} q_{ij}x^j$$

it can be seen that $\omega(y)^q$ can be obtained by multiplying the transpose of vector $(a_{n-1}, a_{n-2}\ldots a_1, a_0)$ by Q.

Calculating the matrix Q is not such a difficult thing either.

If $u(y)=u_0+u_1y+\ldots u_{n-1}y^{n-1}+y^n$ and $\omega(y)$ is a polynomial of degree less than n

$$\omega(y)=\omega_0+\omega_1y+\cdots+\omega_{n-1}y^{n-1}$$

then the result of multiplying $\omega(y)$ by y and dividing by $u(y)$ leaves a remainder

$$-u_0\omega_{n-1}+(\omega_0-u_1\omega_{n-1})y+\ldots(\omega_{n-2}-u_{n-1}\omega_{n-1})y^{n-1}$$

which can be obtained by multiplying the transpose of the vector $(\omega_{n-1}, \ldots \omega_1, \omega_0)$ by the matrix

$$T=\begin{vmatrix} -u_{n-1} & 1 & 0 & 0\ldots 0 \\ -u_{n-2} & 0 & 1 & 0\ldots 0 \\ \vdots & \cdot & \cdot & \cdot\ldots\cdot \\ -u_1 & 0 & 0 & 0\ldots 1 \\ -u_0 & 0 & 0 & 0\ldots 0\end{vmatrix}$$

Hence, to find the successive polynomials x^q, x^{2q} we merely take the transpose vectors $(0, 0, \ldots, 1)$ and multiply successively by T. Raising T to the qth power is accomplished most readily by expanding q in binary. If $q=9$, then $T^9=T^8$. T and $T^8=T^4T^4$ when $T^4=T^2T^2$ so that a total of 4 multiplications suffice.

Let us now exemplify the method by testing $y^2-y-x=u(y)$ for irreducibility. We are taking x to be the primitive element of the field of the polynomials of degree less than 2 and coefficients from the field mod 3. Its addition and multi-

plication table can be gleaned from the previous section. We first notice that the derivative of $u(y)$ to be $2y-1$ (this derivative also must be taken mod the field of 9 elements also—in this case it makes no difference). $2y-1$ and y^2-y-x have no common factor as can be seen by the following g.c.d. calculation in the field of 9 polynomials

$$\begin{array}{rl} & 2y+2 \\ 2y-1 \;\overline{)}\; & y^2-y-x \\ & \underline{y^2-2y} \\ & \quad y-x \\ & \quad \underline{y-2} \\ & \quad\quad 2-x \end{array} \qquad \begin{array}{rl} & xy \\ 2-x \;\overline{)}\; & 2y-1 \\ & \underline{2y} \\ & \quad -1 \end{array}$$

Notice that $2-x=2+2x\equiv x^3$ and $2\equiv x^4$ in the 9-element field so that $(2-x)\cdot x=2$.

We now set up the matrix

$$T=\begin{vmatrix} 1 & 1 \\ x & 0 \end{vmatrix}$$

and raise it to the power 9

$$T^2=\begin{vmatrix} 1+x & 1 \\ x & x \end{vmatrix}\equiv\begin{vmatrix} x^7 & 1 \\ x & x \end{vmatrix}$$

$$T^4=\begin{vmatrix} x^6+x & \\ & x^6+x^2 \end{vmatrix}=\begin{vmatrix} x^3x^2 & \\ & x^31 \end{vmatrix}$$

$$T^8=\begin{vmatrix} x^6+x^5 & x^5+x^2 \\ x^6+x^3 & x^5+1 \end{vmatrix}=\begin{vmatrix} x^4x^7 & \\ & 1\;x^2 \end{vmatrix}$$

and finally

$$T^9=\begin{vmatrix} x^4+1 & x^4 \\ 1+x^3 & 1 \end{vmatrix}=\begin{vmatrix} 0 & x^4 \\ x^5 & 1 \end{vmatrix}$$

Thus the successive rows of the matrix Q are obtained by multiplying the transpose of (0, 1) giving the matrix, written left to right instead of bottom to top. This yields

$$Q=\begin{vmatrix} 0 & 1 \\ 1 & x^4 \end{vmatrix}$$

y^9 therefore is $y+x^4$ mod y^2-y-x.

The g.c.d. of $y+2$ and y^2-y-x is 1, hence y^2-y-x is irreducible.

To test whether y is a primitive element of the field modulo y^2-y-x we note that $(9^2-1)/(9-1)=10$, if the first five powers of y are not primitive polynomials in x (in the field mod x^2-2x-1) then y is primitive.

The successive powers of y can be found by multiplying the transpose of (0, 1) by the matrix

$$\begin{vmatrix} 1 & 1 \\ x & 0 \end{vmatrix}$$

as before,[3] yielding the five polynomials

$$y,\ x+y,\ (1+y)y+x,\ (1+2x)y+1,\ (2+2x)y+(2+2x)$$

so that y is indeed primitive. Also, $y^5\equiv(2+2x)y+(2+2x)=x^3y+x^3$ and so $y^{10}=x^6y^2+x^6+2x^6y=x^6(y+x)+x^6+x^2y=x^7x^6+(x^2+x^6)y=x^5+(1+2x+2+x)=x^5$, a primitive element of the coefficient field.

What would be the logarithm of $y+x+2\equiv y+x^6$? Since $x+y\equiv y^2$ we can write

$$\begin{aligned} y+x^6&\equiv x^5(x+y)+(1-x^5)y\equiv x^5y^2+x^7y\equiv y(x^7+x^5y) \\ &=y(x^6(x+y)+(x^5-x^6)y)\equiv y^2(x^7+x^6y)=y^2(x^6(x+y)) \\ &\equiv y^4x^6. \end{aligned}$$

Now since $x^5\equiv y^{10}$, then $x^6\equiv(x^5)^6$ since $x^{30}=x^{8\cdot3+6}=1^3\cdot x^6$. Hence, $x^6=y^{60}$ and therefore $y+x+2\equiv y^{64}$ and the integer 64 is stored to represent the polynomial.

At retrieval time we recall that $y^{64}=y^{60}\cdot y^4$ and $y^{60}=x^6$. y^4 can be obtained by the matrix multiplication to yield x^2y+1 as above (or by straight division of y^4) so that y^{64} is found to be equivalent to $x^6(x^2y+1)=y+x^6=y+x+2$. It ought to be borne in mind, of course, that these operations all have to be recursive, with one level of recursion for each variable involved.

The above calculations give the reader some idea of the amount of calculation involved. The initial discovery of the irreducible polynomials and the primitive elements is a "once-for-all calculation" and hence not of great importance. However, the work consists of setting up the matrix T and then the n successive multiplications of vectors to obtain the Q matrix. Getting T^q only takes at most $2\log_2(n+1)$ multiplications. The test of the primitiveness of the variable need not be done by successive $(q^n-1)/(q-1)$ multiplications—only the factors of this number are important. Thus in our example we needed only calculate yT^5 for our test. The "exponentiation" method is equally efficient. However, at present we have no good estimate of the effort involved in the finding of the "logarithm." It must be recalled that after the discriminant polynomials are learned, these are "once-for-all" operations also.

The memory saving, however, is considerable. Suppose we want to store N polynomials in V variables and the highest degree to which any variable is involved is n. Let p be the smallest prime larger than all coefficients, considered integral. Then to apply our method we need V irreducible polynomials, each of degree n, needing the storage of nV coefficients (recall that any polynomial which is used as a coefficient can be stored as its logarithm and hence its own coefficients need not be stored). We might perhaps need the storage of some of the $(q^n-1)/(q-1)$ powers of some of the primitive elements as elements of the coefficient fields—but these can also be sorted as the integer "logarithms." Thus a maximum of $(n+1)V+N$ integers need be stored. This is much less than $N\cdot(n+1)^V$ coefficients to be stored initially.

ACKNOWLEDGMENTS

The research that led to the preparation of this paper was supported by the Air Force Office of Scientific Research under Contract Number AFOSR 71-2110B.

REFERENCES

1. Banerji, R. B., "Some Linguistic and Statistical Problems in Pattern Recognition," *Pattern Recognition*, 3, 409, 1971.
2. Berlekamp, E. R., "Factoring Polynomials over Finite Fields," *Bell System Technical Journal*, 46, 1853, 1967.
3. Elspas, B., "The Theory of Autonomous Linear Sequential Networks," *Trans. IEEE*, PGCT-6, 45, 1959.
4. Knuth, E., Alanen, J. D., *Tables of Finite Fields*, Sankhya, Ser A, 26, 305, 1964.
5. Knuth, D. E., *Seminumerical Algorithms*, Addison-Wesley, New York, p. 381, 1971.

A non-associative arithmetic for shapes of channel networks*

by MICHAEL F. DACEY

Northwestern University
Evanston, Illinois

INTRODUCTION

The purpose of this paper is to describe a method for analysis of one type of pictorial information that is abstracted from maps. The picture is a line diagram or graph that, in the language of graph theory, is a planted plane tree in which each vertex has a valency 1 or 3. In hydrology and geomorphology this type of graph is interpreted as a channel network that encompasses the topological properties of the network of rivers and streams comprising a drainage system. A recent survey paper by Dacey[2] identifies a large number of properties of channel networks. Considering that many of these properties are clearly displayed by sketches of channel networks, the mathematical derivations seem unnecessarily complicated. This disparity in level of difficulty may reflect that the pictorial representation of a graph has a structure that is more amenable to analysis than does the conventional linguistic (i.e., mathematical) representation.

This paper describes a formal model that is seemingly more adaptable to the analysis of properties of channel networks and similar graphs than is the combinatorial mathematics that is conventionally used. This model is an "arithmetic of shapes" that incorporates a relatively simple notation to express many of the attributes of graphs. It is an arithmetic in that the rules for operations on shapes and combinations of shapes are in many ways similar to the rules of Etherington's[3] formulation of non-associative arithmetics. The relation between non-associative arithmetics and channel networks was evidently first noted by Smart.[7]

A formal statement of the arithmetic for the shapes of channel networks is provided in this paper. While space limitations preclude demonstration of the utility of this arithmetic model, it evidently yields all basic properties of channel networks. Though this model is formulated in terms of channel networks, the same type of graph also serves as the representation of the partition of a class by bifurcations and, accordingly, has many interpretations other than as channel networks. One application, illustrated by Cavalli-Sforza and Edwards,[1] is for reconstruction of the evolutionary tree leading to the genetic characteristics of an observed population.

* The support of the National Science Foundation, Grant GS-2967, is gratefully acknowledged.

The first part of this paper delimits the domain of the arithmetic model by identifying the basic concepts and structure of channel networks. Then models that encompass this structure are displayed.

CHANNEL NETWORKS

The current study of channel networks largely derives from Shreve's[5,6] formulation of topologically random channel networks. A basic concept is that of topologically distinguishable channel networks. The following structure for analysis of properties of topologically random channel networks is adapted from Dacey.[2] The concepts, terminology and results of graph theory are largely used, though some of the terminology is modified to reflect the specialized terms commonly used in the study of channel networks. In the terminology of graph theory a channel network is a special type of graph consisting of a collection of edges and vertices that form a planted plane tree in which each vertex has valency 1 or 3. This graph is formulated in this study as a collection of *links* and *nodes*. The length and shape of links is not taken into account.

Definition 1. The two nodes of each link are distinguished as *up-node* and *down-node*. A *fork* is formed by the coincidence of nodes of three distinct links—the down-node of two distinct links and the up-node of a third link—and these three nodes are called *members* of a fork. The two links whose down-nodes are members of a fork are called the *branches* of a fork and these branches are oriented with respect to the third link and are distinguished as the *left-branch* and the *right-branch* of the fork. A *nodal point* is an isolated node that does not coincide with any other node. An up-node (down-node) that is a nodal point is called a *source* (*outlet*). An *exterior link* is a link whose up-node is a source, and a *terminal* (or *outlet*) *link* is a link whose down-node is an outlet. A *path* is a sequence of one or more links in which no link appears more than once.

Definition 2. A *channel network of magnitude* $n \geq 1$ is a collection λ_n of links, along with the resulting forks and nodal points, that have the following properties.

(a) Each node of every link in λ_n is a source, an outlet or a member of one fork.

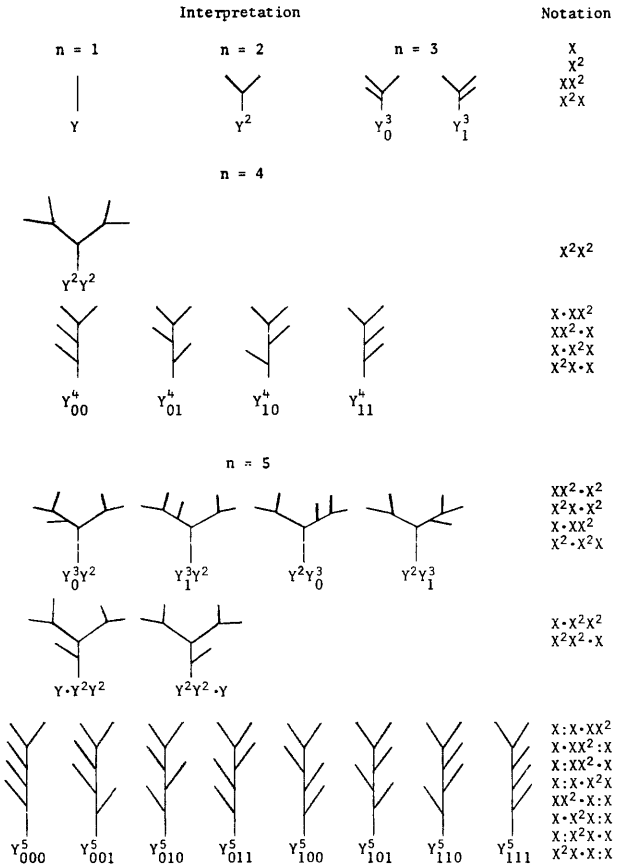


Figure 1—Each row, except the top row, illustrates the topologically distinct channel networks of magnitude n that belong to an ambilateral class. Within each class the top-to-bottom list of notation corresponds to the left-to-right display of diagrams

(b) There are exactly n links in λ_n that have an up-node that is a source and exactly one link in λ_n that has a down-node that is an outlet.

(c) There is exactly one path between every pair of nodes of links in λ_n.

Figure 1 illustrates channel networks of magnitudes 1 through 5.

Definition 3. Two channel networks of magnitude n are called *isomorphic* if there is a one-to-one mapping of the nodes of one onto those of the other which preverved adjacency. Two channel networks of magnitude n are called *map-isomorphic* if there is an isomorphism which preserves the terminal link and the cyclic order of links at corresponding nodes. Two channel networks are called *topologically distinguishable* if they are of different magnitudes or if they are of the same magnitude and are not map-isomorphic. Two channel networks are called *ambilaterally distinct* if they are of different magnitudes or if they are of the same magnitude and are not isomorphic.

Definition 4. Channel networks of magnitude n that are topologically distinguishable but not ambilaterally distinct are said to belong to the same *ambilateral class of magnitude* n.

The term "ambilateral" was suggested by R. L. Shreve and first used in the context of channel networks by Smart.[7]

Figure 1 illustrates the ambilateral classes formed by all distinguishable channel networks of magnitude 1 through 5. Two fundamental properties of channel networks are the number of topologically distinguishable channel networks of magnitude n and the number of ambilateral classes needed to account for these channel networks. A basic tool for the study of these properties is identified by

Definition 5. Let λ_n and λ_m' denote channel networks of magnitudes n and m. The channel networks λ_n and λ_m' and a new link l are said to form a (*terminal*) *splice* when a fork is formed by the up-node of l and the outlets of λ_n and λ_m' such that this fork is the only point common to λ_n and λ_m'. This splice is denoted by $L(\lambda_n, \lambda_m')$ when λ_n is the right-branch of this fork.

Figure 2 illustrates this operation.

Property 1. The $L(\lambda_n, \lambda'_{n-m})$ is a channel network of magnitude n.

Property 2. Each channel network λ_n, $n>1$, is formed by exactly one splice.

PRELIMINARIES

The arithmetic for shapes is defined in terms of rules for combining shapes. One possible interpretation of this arithmetic is for shapes that are channel networks and for a rule of combination that is the splice formed by pairs of channel networks. Although some of the elements of this arithmetic are illustrated by this interpretation, it is convenient to use a more general formulation. The notation for this formulation has been developed by Etherington.[3]

Upper case letters A, B, C, ... plus X are used to denote specific shapes. A rule for combining pairs of shapes is also specified and the combinations of shapes A and B is a shape and is denoted by the product notation AB. The shape AB may be combined with a shape C, the same or different from A and B, in two ways that are denoted by $C(AB)$ and $(AB)C$. Depending upon the shapes and rule of combination $C(AB)$ and $(AB)C$ may be the same or different shapes. Any number of shapes may be combined in this pairwise fashion. To avoid the cumbersome use of multiple brackets, frequent use is made of a notation in which groups of dots separate the factors and fewness of dots establishes the precedence in combining shapes. Thus $A:\cdot BC:CD^2\cdot CE=A\{(BC)[(CD^2)(CE)]\}$, where $D^2=DD$.

The following terminology reflects this notation.

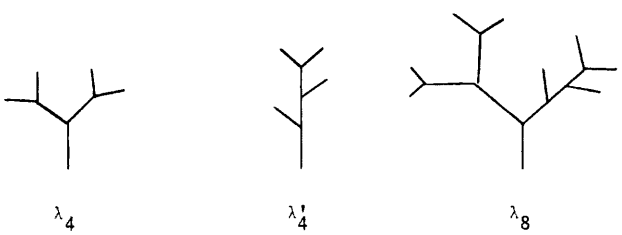


Figure 2—Example of the splice formed by λ_4 and λ_4'

Definition 1. The rule for combining shapes is called *multiplication* and the shape AB is called the *product* of A and B. If A is the product of $B, C, \ldots$, then $B, C, \ldots$, are called *factors* of A. A shape defined by factors that are absorbed one at a time is called a *primary shape.*

An example of a primary shape is $A{:}BC{\cdot}D{:}{\cdot}E$.

The notation simplifies for a system of shapes that are obtained as products of a unique shape. Let X denote this unique shape. The shape $XX=X^2$ is evidently also unique. However, for $n\geq 3$, there are alternative ways of expressing the product of n X's. The five possible products of 4 X's are $XX{\cdot}X{:}X$, $X{\cdot}XX{:}X$, $X{:}XX{\cdot}X$, $X{:}X{\cdot}XX$ and $X{\cdot}X{:}X{\cdot}X$. The first four of these products are primitive shapes. If the product rule is specified as commutative, the expressions for primitive shapes are indistinguishable, but they are distinguishable when the product rule is specified as non-commutative. It is this distinction that suggests the terminology of *commutative shapes* and *non-commutative shapes.*

A basic distinction between commutative and non-commutative shapes is that the 2^{n-2}, $n\geq 2$, ways of absorbing n factors one at a time are not distinguishable for commutative shapes and are distinguishable for non-commutative shapes. One consequence is that only one primary shape is the product of n factors of commutative shapes and it is represented by X^n. In contrast, a more elaborate notation is required to represent the 2^{n-2} primary shapes that are the product of n factors of non-commutative shapes.

The simpler notation used for commutative shapes is illustrated by the following examples. Two primitive shapes are represented by X^m and X^n. Additional shapes are represented as products, powers and iterated powers of these two shapes so that X^mX^n, $(X^m)^n$ and $(X^m)^n)^n$ are also shapes. The following operations are used to express these latter shapes as an index of X. The product of two powers of the same shape is indicated as a sum in the index of the shape, the power of a power is indicated as a product in the index of the shape, and an iterated power is indicated as a power in the index. For example, where X is the shape,

$$X^2X^3=X^{2+3} \qquad X^3X^2=X^{3+2}$$
$$(X^2)^3=X^{2\cdot 3} \qquad (X^3)^2=X^{3\cdot 2}$$
$$((X^2)^2)^2)=X^{2^3} \qquad (X^3)^3=X^{3^2}$$
$$(X^2)^{3+2}=X^{2(3+2)} \qquad (X^{3+2})^2=X^{(3+2)2}$$

This superscript notation is cumbrous and frequent use is made of the convention that X^a is represented by simply a. By this convention, the expression $a+b=c$ is in the arithmetic of indices and is equivalent to $X^aX^b=X^{a+b}=X^c$. Similarly, $ab=c$ is equivalent to $X^{ab}=X^c$. In the arithmetic of indices the letters $a, b, c, \ldots$ may be positive integers or sums, products and powers of positive integers, but the letters m and n are always positive integers.

The essential features of this notation are reflected in

Definition 2. Let A_1 and A_2 be any two shapes a_1 and a_2 that are formed by factors of X. Then a_1+a_2 is the shape of the product A_1A_2, and a_1a_2 is the shape of the product formed by substituting A_1 for each of the factors of A_2.

This notation is adapted to the structure of channel networks by the following interpretations. The basic shape X is a link along with its up-node and down-node. The shape a is

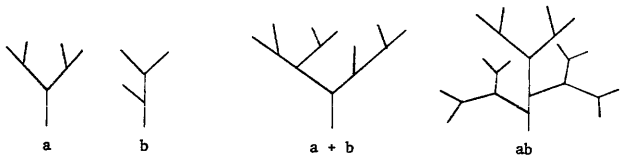


Figure 3—Example of the channel networks $a+b$ and ab that are obtained as products of the channel networks a and b

a channel network. The shape $a+b$ is a channel network and the rule of combination is the splice of the channel networks a and b. The shape ab is a channel network and the rule of combination is that each exterior link of b is replaced by a. Figure 3 illustrates these operations, while Figure 1 illustrates the representation of channel networks as factors of X.

ARITHMETIC OF COMMUTATIVE SHAPES

The system C of commutative shapes is defined by the following six rules.

C1 (Existence Rule). The shape $X=X^1$ is in C.

C2 (Closure Rule). If X^a and X^b are shapes that are in C then X^{a+b} and X^{ab} are shapes in C.

The following three rules pertain to shapes that are in C and are expressed in the arithmetic of indices. In this arithmetic, a stands for X^a.

C3 (Associative Rule). $(a+b)+c\neq a+(b+c)$ provided that $a\neq c$, and $ab{\cdot}c=a{\cdot}bc$.

C4 (Commutative Rule). $a+b=b+a$, and $ab\neq ba$ provided that $a\neq 1$ and $b\neq 1$. For $b=1$, $a{\cdot}1=1{\cdot}a=1$.

C5 (Distributive Rule). $a(b+c)=ab+ac$, and $(b+c)a\neq ba+ca$ provided that $a\neq 1$.

C6 (Formation Rule). If X^n is in C, then for $n\geq 2$, the shape $X^n=XX^{n-1}$.

Property 1. For each positive integer n, X^n is in C and is a primitive shape.

Definition 1. Two shapes X^a and X^b in C are said to be *isomorphic* if, and only if, $a=b$. Two shapes that are not isomorphic are called *ambilaterally distinct shapes.*

Property 2. The shape X^n is isomorphic with any shape that is a primitive shape formed by n factors of X.

Property 3. If a, b, c and d are shapes in C such that $a=c$ and $b=d$, then $a+b$, $b+a$, $c+d$ and $d+c$ are all equal and, hence, are isomorphic.

Property 4. If c is a shape in C, then $c=1$ or there exist in C chapes a and b such that $c=a+b$.

Definition 2. Let $\delta(a)$ be the value in ordinary arithmetic of the index a of the shape X^a, and $\delta(a)$ is called the *degree* of the shape X^a.

By Definition 1.4, the collection of all shapes of degree n that are mutually isomorphic is called an *ambilateral class* (of degree n).

Definition 3. Let $Q(n)$ be the minimum number of ambilateral classes necessary to contain all shapes in C that have degree n.

Property 5. For each of the degrees 1, 2 and 3 there is a single ambilateral class and, hence, a single ambilaterally distinct shape. The two ambilaterally distinct shapes of degree 4 are isomorphic with 4 or 2·2.

The proof of this property is displayed in order to illustrate the verification of statements about shapes in the arithmetic of indices. The statement is obvious for degree 1 and also for degree 2 since the only factoring of 2 is as 1+1. The four cases of degree 3 to consider are (1+1)+1, 1+(1+1), 2+1 and 1+2. By Property 2, the first two shapes are isomorphic with 3, while C6 implies (1+1)+1=2+1 and 1+(1+1)=1+2. To obtain the two distinct shapes of degree 4, first observe that

$$\begin{aligned} 4&=1+3 && [C6]\\ &=3+1 && [C4]\\ &=(1+2)+1=(2+1)+1 && [C6 \text{ and } C4]\\ 4&=(1+2)+1 && [C6]\\ &=1+(2+1) && [C3]\\ &=1+(1+2). && [C4] \end{aligned}$$

Alternatively, Property 2 establishes that the last four shapes are isomorphic with 4. Also, it establishes that any shape that is the sum of four 1's taken one at a time is isomorphic with 4. Next consider

$$\begin{aligned} 2+2&=(1+1)+2 && [2=1+1]\\ &\neq 1+(1+2) && [C3]\\ &=1+3 && [C6]\\ &=4. && [C6] \end{aligned}$$

Hence, 4 and 2+2 are distinct shapes. The only remaining indices of degree 4 are 4·1=1·4=4 and 2·2, but

$$\begin{aligned} 2\cdot 2&=2(1+1) && [C6]\\ &=2+2. && [C5] \end{aligned}$$

Accordingly, every shape of degree 4 is isomorphic with either 4 or 2·2=2+2.

To continue to enumerate in the preceding fashion shapes of degree 5 and higher is laborious. A more efficient enumeration makes use of an ordering principle suggested by Harding.[4]

Definition 4. The collection $\sum_n$ contains $Q(n)$ shapes of degree n, and

$$\sum = \bigcup_{n=1}^{\infty} \sum_n.$$

The i-th shape of the $Q(n)$ shapes of degree n in $\sum_n$ is denoted by $\mathbf{n}_i$, $1<i\leq Q(n)$. If $\mathbf{r}_h$, $\mathbf{s}_k$ and $\mathbf{n}_i$ correspond to the shapes a, b and c and if $c=a+b$, then an alternative notation for this shape is $\mathbf{n}_i=\mathbf{r}_h+\mathbf{s}_k$. The shape $\mathbf{1}_1$ is in $\sum$. The shape $\mathbf{r}_h+\mathbf{s}_k$ is in $\sum$ if, and only if,

$$\mathbf{r}_h \text{ and } \mathbf{s}_k \text{ are both in } \sum \text{ and either } r<s \text{ or } r=s \text{ and } h\leq k.$$

Property 6. If $\mathbf{n}_i$ is a shape in C and $n>1$, then there exist in $\sum$ shapes $\mathbf{r}_h$ and $\mathbf{s}_k$ such that $\mathbf{n}_i=\mathbf{r}_h+\mathbf{s}_k$ and either $r<s$ or $r=s$ and $h\leq k$.

Ordering Rule. Suppose $\mathbf{n}_i$ and $\mathbf{n}_j$ are shapes in $\sum$ such that

$$\begin{aligned} \mathbf{n}_i&=\mathbf{r}_h+\mathbf{s}_k, && (r+s=n,\ r\leq s,\ \mathbf{r}_h \text{ and } \mathbf{s}_k \text{ are in } \textstyle\sum),\\ \mathbf{n}_j&=\mathbf{t}_p+\mathbf{u}_q, && (t+u=n,\ t\leq u,\ \mathbf{t}_p \text{ and } \mathbf{u}_q \text{ are in } \textstyle\sum). \end{aligned}$$

If $\mathbf{n}_i$ and $\mathbf{n}_j$ are ambilaterally distinct shapes, then $i<j$ so that the shape $\mathbf{n}_i$ precedes $\mathbf{n}_j$ if, and only if,

$$\begin{aligned} r\leq t \text{ and either } & r<t,\\ \text{or } & r=t \text{ and } k<q,\\ \text{or } & r=t,\ k=q \text{ and } h<p. \end{aligned}$$

If $\mathbf{n}_i$ and $\mathbf{n}_j$ are isomorphic, then $i=j$.

Definition 5. For $n>1$, the shape $\mathbf{n}_i$, $1\leq i\leq Q(n)$, is a shape in $\sum_n$ that satisfies the preceding ordering rule with respect to all pairs of shapes in $\sum_n$.

Property 7. The ordering principle of Definition 5 unambiguously orders the $Q(n)$ ambilateral classes of degree n.

Since there is only one shape of degrees 1, 2 and 3, put $\mathbf{1}_1=\mathbf{1}$, $\mathbf{2}_1=\mathbf{2}$ and $\mathbf{3}_1=\mathbf{3}$. This ordering of classes of degree 5 and less is as follows.

$$\begin{aligned} Q(1)&=1. && \mathbf{1}=\mathbf{1}_1.\\ Q(2)&=1. && \mathbf{2}=\mathbf{2}_1=\mathbf{1}_1+\mathbf{1}_1.\\ Q(3)&=1. && \mathbf{3}=\mathbf{1}_1+\mathbf{2}_1=\mathbf{1}+\mathbf{2}.\\ Q(4)&=2. && \mathbf{4}_1=\mathbf{1}_1+\mathbf{3}_1=\mathbf{1}+(\mathbf{1}+\mathbf{2}),\\ & && \mathbf{4}_2=\mathbf{2}_1+\mathbf{2}_1=\mathbf{2}+\mathbf{2}.\\ Q(5)&=3. && \mathbf{5}_1=\mathbf{1}_1+\mathbf{4}_1=\mathbf{1}+(\mathbf{1}+(\mathbf{1}+\mathbf{2})),\\ & && \mathbf{5}_2=\mathbf{1}_1+\mathbf{4}_2=\mathbf{1}+(\mathbf{2}+\mathbf{2}),\\ & && \mathbf{5}_3=\mathbf{2}_1+\mathbf{3}_1=\mathbf{2}+(\mathbf{1}+\mathbf{2}). \end{aligned}$$

This structure may be used to verify the following basic, though well-known, result.

Theorem 1. $Q(1)=Q(2)=1$ and for $n\geq 2$,

$$Q(2n-1)=\sum_{i=1}^{n-1} Q(i)Q(2n-1-i)$$

and

$$Q(2n)=\sum_{i=1}^{n-1} Q(i)Q(2n-i)+\tfrac{1}{2}Q(n)[Q(n)+1].$$

A simple, non-recursive expression for $Q(n)$ is not known, but the values form the sequence 1, 1, 1, 2, 3, 6, 11, 23, 46, 98, ...

With the value of $Q(n)$ established, an algorithm given by Harding[4] may be used to identify the composition of $\mathbf{n}_i$, for specified values of n and i, without enumerating all shapes that precede $\mathbf{n}_i$ in $\sum$.

Definition 6. Put $q_n(0)=0$ and

$$q_n(j)=\sum_{i=1}^{j} Q(i)Q(n-i), \qquad 1\leq j\leq n-1.$$

Theorem 2. Suppose $\mathbf{n}_i$, $\mathbf{r}_h$ and $\mathbf{s}_k$ are shapes in $\sum$ such that $\mathbf{n}_i=\mathbf{r}_h+\mathbf{s}_k$, where $r<s$ or $r=s$ and $h\leq k$. Then the values n, r, s, h and k satisfy

$$i=\begin{cases} q_n(n-s-1)+(k-1)r+h, & r\neq s,\\ q_n(n-s-1)+\tfrac{1}{2}k(k+1)+h, & r=s, \end{cases} \qquad (*)$$

where $1\leq h\leq Q(r)$, $1\leq k\leq Q(s)$ and $r<s$ or $r=s$ and $h\leq k$.

To solve this equation, first find the maximum value of $n-s-1=r-1$ that is compatible with (*). Using this value, then find the maximum value of k that is compatible with (*). Finally, using these two values, find the value of h so that (*) is satisfied. In identifying all of these values, the constraints on r, s, h and k are obeyed.

This completes the display of the notation and ordering of shapes for the arithmetic model for the ambilateral classes of channel networks. The classes of topologically distinguishable shapes are of greater interest to the study of channel networks and an arithmetic model for the non-commutative shapes is displayed next.

ARITHMETIC OF NON-COMMUTATIVE SHAPES

The notation suggested by Etherington[3] and used to formulate the system C of commutative shapes is inadequate to express the system A of non-commutative shapes. One limitation is that the primary shape formed by n factors of the basic shape is unique for commutative shapes but for non-commutative shapes has 2^{n-2}, $n\geq 2$, possible interpretations. Accordingly, 2^{n-2} different symbols are required to represent the primary non-commutative shapes that correspond to X^n in C. To distinguish these shapes a new notation is devised. Comparison of the systems C and A is facilitated by using Y to represent the basic shape for the system A.

The following notation takes into account the 2^{n-2} primitive shapes formed by n factors of Y. Sequences of 0's and 1's are used to differentiate the primitive shapes. There are 2^n different sequences formed by n 0's and 1's. Let i_n, $1\leq i\leq 2^{n-2}$, stand for one of the sequences formed by $(n-2)$ 0's and 1's. If i_n and i_m' are identical sequences of 0's and 1's, so that $n=m$, this is denoted by $i_n=i_n'$ and otherwise $i_n\neq i_m'$. The symbols i_n0 and i_n1 stand for the sequences of $n-1$ 0's and 1's for which the first $n-2$ entries are identical to those of i_n and the $(n-1)$-th entry is, respectively, 0 and 1.

The primitive shapes formed by n factors of Y are represented by $\{Y_{i_n}{}^n : 1\leq i\leq 2^{n-2}\}$. This notation does not make explicit the particular sequence of $(n-2)$ 0's and 1's that is represented by an i_n, but this information is not needed. The shape represented by a known sequence of 0's and 1's is, however, determined by the notation. The shapes Y and $Y^2=Y_{i_2}{}^2$ are unique, while for $n\geq 3$ the $Y_{i_n}{}^n$ are defined recursively by

$$\begin{cases} Y_{i_{n-1}0}{}^n = YYi_{n-1}{}^{n-1} \\ Y_{i_{n-1}1}{}^n = Y_{i_{n-1}}{}^{n-1}Y. \end{cases} \qquad (\dagger)$$

This procedure uniquely determines the shape associated with a particular sequence of 0's and 1's. For example, the four possible interpretations of $Y_{i_4}{}^4$ are $Y_{00}{}^4$, $Y_{01}{}^4$, $Y_{10}{}^4$, $Y_{11}{}^4$, but these correspond, respectively, to $Y{:}Y\cdot YY$, $Y\cdot YY{:}Y$, $Y{:}YY\cdot Y$, $YY\cdot Y{:}Y$.

Figure 1 illustrates this notation.

The system A of non-commutative shapes is defined by the following six rules.

A1 (Existence Rule). The shape $Y=Y^1$ is in A.

A2 (Closure Rule). If Y^a and Y^b are shapes that are in A then Y^{a+b} and Y^{ab} are shapes in A.

The next three rules pertain to indices of Y.

A3 (Associative Rule). $(a+b)+c\neq a+(b+c)$ provided that $a\neq c$, and $ab\cdot c=a\cdot bc$.

A4 (Commutative Rule). $a+b\neq b+a$, and $ab\neq ba$ provided that $a\neq 1$ and $b\neq 1$. For $b=1$, $a\cdot 1=1\cdot a=a$.

A5 (Distributive Rule). $a(b+c)=ab+ac$, and $(b+c)a\neq ba+ca$ provided that $a\neq 1$.

These five rules are the same as the rules C1-C5 of the system C with the exception of the commutative rule for addition in the arithmetic of indices. Further, the following formation rule differs from C6 in order to account for the 2^{n-2} expressions required for primitive shapes.

Clearly, the shapes Y and $Y^2=YY$ are in A.

Definition 1. Put $Y_{i_0}{}^2=Y^2$ and each i_n, $1\leq i\leq 2^{n-2}$, represents a different sequence of $n-2$ 0's and 1's.

A6 (Formation Rule). For $n\geq 3$, if $\{Y_{i_n}{}^n : 1\leq i\leq 2^{n-2}\}$ are shapes in A, then these shapes are defined by (†) for each i, $1\leq i\leq 2^{n-3}$.

Property 1. For each integer $n\geq 3$ and $1\leq i\leq 2^{n-2}$, $Y_{i_n}{}^n$ is a shape in A.

Definition 2. Two shapes Y^a and Y^b in A are called *map-isomorphic*, which is written as $Y^a\sim Y^b$, if, and only if, $a=b$. Shapes that are not map-isomorphic are called *topologically distinguishable*.

Property 2. Two shapes $Y_{i_n}{}^n$ and $Y_{j_m}{}^m$ are topologically distinguishable if $n\neq m$ or if $n=m$ and $i_n\neq j_n$.

Property 3. Suppose A_1 and A_2 are shapes in A such that A_3 is a factor of A_1 but no product expression for A_2 includes A_3 as a factor. Then A_1 and A_2 are topologically distinguishable.

Property 4. If A is a shape in A, then there exists an exponent a such that $A\sim Y^a$. In particular, for any shape $Y_{i_n}{}^n$ in A there exists an a such that $Y_{i_n}{}^n\sim Y^a$.

Definition 3. Let A be a shape in A for which $A\sim Y^a$. The X-transform of A is X^a and is denoted by $X(A)$. The arithmetic for the indices of an X-transform obeys the rules C3, C4 and C5 of the system C.

Property 5. If A is a shape in A and X is a shape in C, then $X(A)$ is a shape in C.

Definition 4. Suppose Y^a and Y^b are shapes in A such that $X(Y^a)=X^\alpha$ and $X(Y^b)=X^\beta$. If, by the rules of the system C, $\alpha=\beta$, then Y^a and Y^b are said to belong to the same *ambilateral class*. Alternatively, if $\alpha\neq\beta$, Y^a and Y^b belong to different ambilateral classes and are called *ambilaterally distinct*.

Notice that two shapes may be classified as "topologically distinguishable" or as "ambilaterally distinct."

As in the previous section, a shape Y^a in A is frequently denoted by a.

Property 6. If a_1 and a_2 are shapes in A that are ambilaterally distinct, then a_1 and a_2 are topologically distinguishable.

Conversely, shapes that are topologically distinguishable may or may not be amiblaterally distinct.

Definition 5. Let $\delta(a)$ be the value in ordinary arithmetic of the index a of the shape Y^a and $\delta(a)$ is called the *degree* of Y^a.

Property 7. If a and b are shapes in A and $\delta(a) \neq \delta(b)$, then a and b are ambilaterally distinct.

Property 8. If a is a shape in A and $a=b_1+c_1=b_2+c_2$, then $b_1=b_2$ and $c_1=c_2$.

Theorem 1. Let $Q(n)$ denote the minimum number of ambilateral classes required to contain all shapes in A that have degree n. Then $Q(n)$ satisfies Theorem 3.1.

The remainder of this paper simply displays, without comment, the procedure for ordering shapes.

Definition 6. All shapes in A that are mutually map-isomorphic are said to belong to the same *topological* class. Let $N(n)$ denote the minimum number of topological classes required to contain all shapes in A of degree n.

Definition 7. The collection Γ_n contains $N(n)$ shapes of degree n, and

$$\Gamma = \bigcup_{n=1}^{\infty} \Gamma_n.$$

The i-th shape of the $N(n)$ shapes of degree n in Γ_n is denoted by $\mathbf{n}_i$, $1 \leq i \leq N(n)$. If $\mathbf{r}_h$, $\mathbf{s}_k$ and $\mathbf{n}_i$ correspond to the shapes a, b and c and if $c=a+b$, then an alternative notation for this shape is $\mathbf{n}_i=\mathbf{r}_h+\mathbf{s}_k$. Further, the shape $\mathbf{l}_1$ is in A and the shape $\mathbf{r}_h+\mathbf{s}_k$ is in Γ if, and only if, $\mathbf{r}_h$ and $\mathbf{s}_k$ are both in Γ.

Ordering Rule. Suppose $\mathbf{n}_i$ and $\mathbf{n}_j$ are shapes in Γ such that

$$\begin{aligned} &\mathbf{n}_i=\mathbf{r}_h+\mathbf{s}_k, && (r+s=n,\ \mathbf{r}_h \text{ and } \mathbf{s}_k \text{ are in } \Gamma),\\ &\mathbf{n}_j=\mathbf{t}_p+\mathbf{u}_q, && (t+u=n,\ \mathbf{t}_p \text{ and } \mathbf{u}_q \text{ are in } \Gamma). \end{aligned}$$

If $\mathbf{n}_i$ and $\mathbf{n}_j$ are topologically distinguishable, then $i<j$ so that $\mathbf{n}_i$ precedes $\mathbf{n}_j$ if, and only if, one of the following conditions is satisfied.

$$\begin{aligned} &r \leq s,\ t>u;\\ &r=s,\ t=u,\ r<t;\\ &r=s,\ t=u,\ r=t,\ h<p;\\ &r=s,\ t=u,\ r=t,\ h=p,\ k<q;\\ &r<s,\ t<u,\ s<u;\\ &r<s,\ t<u,\ s=u,\ r<t;\\ &r<s,\ t<u,\ s=u,\ r=t,\ h<p;\\ &r<s,\ t<u,\ s=u,\ r=t,\ h=p,\ k<q. \end{aligned}$$

Further, if $r>s$ and $t>u$, then $\mathbf{u}_q+\mathbf{t}_p$ precedes $\mathbf{s}_k+\mathbf{r}_h$ if, and only if, $\mathbf{n}_i$ precedes $\mathbf{n}_j$. If $\mathbf{n}_i$ and $\mathbf{n}_j$ are map-isomorphic, then $i=j$.

Theorem 2. $N(n)=(2n-2)!/(n-1)!n!$.

Definition 8. Put $p_n(0)=0$ and

$$p_n(j)=\sum_{i=1}^{j} N(i)N(n-i), \qquad 1 \leq j \leq n-1.$$

Theorem 3. Suppose $\mathbf{n}_i$, $\mathbf{r}_h$ and $\mathbf{s}_k$ are shapes in Γ such that $\mathbf{n}_i=\mathbf{r}_h+\mathbf{s}_k$. Then n, r, s, h and k satisfy the following conditions. If n is odd and $i \leq p_n(\frac{1}{2}n-\frac{1}{2})$ or n is even and $i \leq p_n(\frac{1}{2}n)$, then

$$i=\begin{cases} p_n(n-s-1)+(k-1)r+h, & r \neq s,\\ p_n(n-s+1)+\frac{1}{2}k(k+1)+h, & r=s, \end{cases} \qquad (\ddagger)$$

where $1 \leq h \leq N(r)$, $1 \leq k \leq N(s)$ and $r \leq s$. When i exceeds the upper bound, $\mathbf{n}_i$ is related to a shape $\mathbf{n}_j$ for which j does not exceed the upper bound and, hence, $\mathbf{n}_j$ is determined by (†). If n is odd and $i>p_n(\frac{1}{2}n-\frac{1}{2})$, then $\mathbf{s}_k+\mathbf{r}_h=\mathbf{n}_j$ and $j=2p_n(\frac{1}{2}n-\frac{1}{2})-i+1$. If n is even and $i>p_n(\frac{1}{2}n)$, then $\mathbf{s}_k+\mathbf{r}_h=\mathbf{n}_j$ and $j=2p_n(\frac{1}{2}n)-N(n)^2-i+1$.

The equation (‡) is solved by the same iterative procedure recommended for (*) of Theorem 3.2.

REFERENCES

1. Cavalli-Sforza, L. L., Edwards, A. W. F., "Phylogenetic analysis," *Amer. Jour. Human Genetics*, 19, pp. 233-257, 1967.
2. Dacey, M. F., *Summary of Magnitude Properties of Topologically Distinct Channel Networks and Network Patterns*, Paper delivered at International Geological Congress meetings, Montreal, August, 1972. To be published in Proceedings.
3. Etherington, I. M. H., "On Non-Associative Combinations," *Proc. Roy. Soc. Edinburgh*, 59, pp. 153-162, 1939.
4. Harding, E. F., "The Probabilities of Rooted Tree-Shapes Generated by Random Bifurcation," *Advances in Appl. Prob.*, 3, pp. 44-77, 1971.
5. Shreve, R. L., "Statistical Law of Stream Numbers," *Jour. of Geology*, 74, pp. 17-37, 1966.
6. ———., Infinite Topologically Random Channel Networks," *Jour. of Geology*, 75, pp. 178-186, 1967.
7. Smart, J. S., "Topological Properties of Channel Networks," *Geol. Soc. of America Bull.*, 80, pp. 1757-1774, 1969.

The description of scenes over time and space*

by LEONARD UHR

University of Wisconsin
Madison, Wisconsin

INTRODUCTION

Most pattern recognition research has been concerned with the assignment of a *single name* to an input field. But rarely do we find single, isolated objects in the real world.

Describing scenes of several objects that interact over time and space

Rather, we need programs that describe *scenes* of several interacting objects, and further describe each object, commenting upon parts, qualities, and other details of interest.

Just as they are not unitary things isolated in space, real-world objects are not isolated, as in a photo, in a static moment of time. But virtually no research has been done on the recognition of objects that come, go, move, and change over time.

Once we introduce time we raise a number of interesting issues: What kind of short-term perceptual memory is needed? How does the system handle, and coordinate, time for perception, and for response (in our case, for describing)? How can the system use information gathered so far during perceptual interaction with its environment in order to help it glance about and attend to objects as they come into view at future times?

These are extremely complex and subtle, and interesting, questions. This paper is a first attempt to tackle them.

The use of bare-bones EASEy programs to make things clear

Actual computer programs are presented, described, and discussed, in order to make completely clear exactly what is happening, and to allow us to examine a variety of variations. These programs are kept to their bare bones, and coded in a relatively simple English-like variant of SNOBOL called EASEy (an *E*ncoder for *A*lgorithmic *S*yntactic *E*nglish that's Easey). Programs and variants are numbered so that they can be compared one with another, and the Appendix and the EASEy primer[35] should be helpful when details of the code are not apparent.

These programs are designed to demonstrate a variety of possible mechanisms, to be compared and contrasted one with another. They depend upon the particular set of characterizers given them (by their programmers, or by learning routines). We have not been able to examine within the brief confines of this paper the kind of behavior they will exhibit, and the variety of sensed scenes they will handle, given a sufficiently large and appropriate set.

HISTORY

Relatively little research has been done on general systems that *describe* the various objects in a scene, along with their structure of parts, qualities, defects, or other characteristics. On the contrary, almost all pattern recognition research has concentrated on the assigning of a single name to an input. When scenes are examined, they tend to be treated in an *ad hoc* way, using routines designed to find special features of interest.

Systems for recognition of continuous handwriting and speech

Some research has been done on recognizing handwriting, where several letters continue into one another, to form words and lines.[4,8,24,41] And some attempts have been made in speech recognition to describe the spoken utterance in terms of their basic components.[7,27] But virtually all of this work first decomposes the scene, whether of letters or sounds, into individual units, thus reducing the problem to standard single-name pattern recognition, and then assigns a single name, using standard techniques.

Systems that build internal descriptions to name

Some programs that name develop a rich internal description of the pattern in order to achieve the name. The best examples of such an approach are probably the "syntactic"

* This research has been partially supported by grants from the National Institute of Mental Health (MH-12266), the National Science Foundation (GJ-36312), NASA (NGR-50-002-160) and the University of Wisconsin Graduate School.

recognizers[9,10,18,23,25,26,31,57] since they build up structures of larger and larger wholes from meaningful parts. But in fact almost any naming program that applies a set of characterizers to an input pattern can be thought of as building up an *internal* description, of those characterizers that succeed, and where, and those that fail. This information might be output, as a description; it would be useful and meaningful to the extent that the characterizers were ones that made sense to the human receiver. And any recognizer could "describe" by outputting some of the alternate implied names that it might have chosen, but didn't.

Systems that examine continuous fields of objects

A variety of important problem areas confront us with scenes of objects. These include many biological preparations, e.g., blood cells, nerve tissue, chromosomes; X-rays, e.g., of heart, lungs, or bones; and aerial photos of cities, country side, or cloud cover. Up to now such work has concentrated on extracting particular features of interest, e.g., an enlargement or other anomaly of an organ; an aberrant blood cell; a texture of a certain sort; a break in a bone; an edge of a cloud; a boundary between two fields of different crops. Rarely is a complete description asked for or given; rather, a special-purpose program is coded to analyze and search for particular signs of interest.[2,7,12,13,17,20,21,29,33]

Kirsch,[15] Londe and Simmons,[22] Fischler,[6] Firschein and Fischler,[5] Sauvain and Uhr,[30] and Uhr[36,39,40] are examples of research that attempts to develop more complete descriptions, though usually under the assumption that only one, or at most two, simple, standard, noise-free objects are present in the scene.

Systems for the description of three-dimensional objects

A good deal of interest has arisen in recent years in the problem of recognizing objects that overlap, often in three dimensions, in the fields of computer graphics[11,28] and robots.[3] But most of this work very carefully attempts to find the edges that are predicted to be present for one of the small number of alternate possible objects.

Recognition over time

Virtually no research has been done with objects that move or otherwise change over *time*, except for special-purpose systems, such as those that track clouds.[19,32] Nor is the author aware of any systems that build up a short-term perceptual memory in order to handle such continually changing inputs.

Glancing about, and conversation

Relatively little research has been done on pattern recognition systems that *decide where to look next* as a function of what information they have gathered so far. Most "concept formation" systems have a very simple and rigid structure of this sort.[14,16,34] Uhr[38,40] has examined more flexible systems of this sort.

A PROGRAM FOR TWO-DIMENSIONAL RECOGNITION AND DESCRIPTION

We will now examine two programs that explore the problems of describing objects that change over time. The first, *DESCRIBE-1*, makes minimal changes to a relatively typical configurational pattern recognizer[40] to allow it to describe.

DESCRIBE-1 handles recognition in two dimensions, using configurational characterizers that are sensitive to interactions among their parts. It insists that each part be *exactly* positioned to match; but characterizers are threshold elements, so that they can succeed when any sufficiently highly weighted subset of their parts succeeds. It gives a first approximation to a description, since it outputs *all* names that have been sufficiently implied to exceed a CHOOSE* level.

(OVERVIEW *DESCRIBE-1*. Uses weighted, positioned configurations in 2 dimensions.

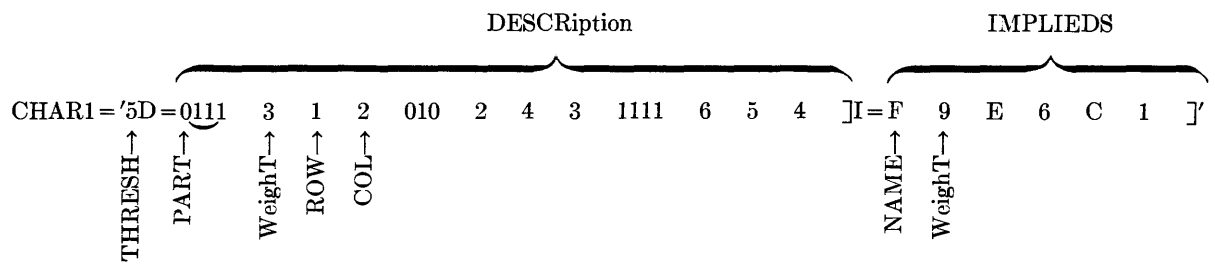
(*Outputs* all NAMEs* implied above CHOOSE level.	*DESCRIBE-1*
INITialize the CHOOSE level and the configuration CHARacterizers.	M1-MN
UPDATE *Erase* the old PRESENT and the ROW Present	1
IN *input* the new PRESENT, ROW by ROW: put the CHARacterizers on LOOKFOR	2-5
PERCEIVE Get each CHARacterizer, its THRESHold, DESCRiption, and IMPLIEDS	6-8
Get each PART, its ROW and COLumn location, and WeighT, from the DESCRiption, and look for it, positioned, in the PRESENT; and add the WeighT to TOTAL if the part is found	9-10 11
TEST This CHARacterizer succeeds if TOTAL is above THRESHold.	12
IMPLY Get each NAME and its WeighT from the IMPLIEDS, and add this WeighT into the TOTAL for this NAME on MAYBE	13 14-16
DECIDE Get each NAME and its TOTAL from MAYBE, and, if TOTAL is higher than the CHOOSE level, *output* it.	17 18-19

* Capitalized strings in the text and the OVERVIEWs refer to programmer-defined string names in the programs.

```
(DESCRIBE-1. (Handles 2-dimensional patterns. CHARacterizers are WeighTed, (positioned configurations   DESCRIBE-1
that succeed if match above THREshold. Applies (CHARacterizers till a NAME's TOTAL implied weight
exceeds CHOOSE level
        Set CHOOSE=30                                                                                    [1]M1
(Characterizers should go here.
UPDATE[2] erase PRESENT, ROWP                                                                            [3]1
(input the PRESENT pattern ROW by ROW)
IN      input TYPE, Row till ] [+ to $TYPE -to END][2]                                                    2
S       ROWP=ROWP+1                                                                                      3
        On PRESENT set] ROWP] ROW ] [to IN]                                                              [4]4
R       Set LOOKFOR='CHAR1 CHAR2 . . . CHARN'                                                            5
PERCEIVE from LOOKFOR get CHAR=[-DECIDE]                                                                 6
(Look for each OBJect in the DESCRiption at ROW and COLumn specified)
        from $CHAR get THRESH 'D=' DESCR ']I=' IMPLIEDS ] [-PERCEIVE]                                    [5]7
        erase TOTAL                                                                                      8
P1      from DESCR get PART WT ROW COL=[-TEST]                                                           [6]9
        from PRESENT get ] that ROW ] call COL symbols LEFT, get that PART [-P1]                         [3]10
            TOTAL=TOTAL+WT [P1]                                                                          11
TEST    is THRESH lessthan TOTAL? [-PERCEIVE]                                                            12
(If total WeighT of OBJects got exceeds THRESHold, merge IMPLIEDS NAMEs onto MAYBE
IMPLY   from IMPLIEDS get NAME WT=[-PERCEIVE]                                                            13
        from MAYBE get # that NAME # TOTAL=[-I1]                                                         [4]14
            WT=TOTAL+WT                                                                                  15
I1      on MAYBE list NAME WT [IMPLY]                                                                    16
(output description with all NAMEs implied above CHOOSE level.
DECIDE  from MAYBE get NAME TOTAL=[-UPDATE]                                                              17
        is TOTAL lessthan CHOOSE ? [+DECIDE]                                                             18
OUT     output 'THERE IS' NAME [DECIDE]                                                                  19
end[2]                                                                                                   —
S 00111110]     (example data cards for an 'F' in an 8 by 8 matrix)                                      I1
S 00110000]                                                                                              I2
S 00100000]                                                                                              I3
S 00011110]                                                                                              I4
S 00011111]                                                                                              I5
S 00110000]                                                                                              I6
S 00110000]                                                                                              I7
S 00100000]                                                                                              I8
R ]                                                                                                      I9
```

[1] NOTE. See the Appendix for a brief description of EASEy programs. (Examples of the major program constructs discussed in the Appendix are superscripted 1 through 6 in this program.) Caps are used in the text to refer to program constructs.

If given good characterizers for the letters, *DESCRIBE-1* will output the name F and, probably, a few other names like C, I, and, possibly, E and T. For good performance, the program would need several hundred characterizers, of the following sort:



And it would also need more code to allow these characterizers to succeed within some *region*[38,40] rather than in an exact position (alternately, it could be given a separate characterizer for every position; but this would clog memory with far too many characterizers, and slow down processing time). But this kind of program, with an adequate set of characterizers, performs well[1,41] possibly as well as any approach (see Zobrist[42] for comparisons).

Other possibilities for descriptive information

DESCRIBE-1 output as its description all the NAMEs IMPLIED above the CHOOSE level. *DESCRIBE-2* and its extensions will explore a variety of richer descriptive information, including the objects' parts, structure, salience, qualities, and location.

DESCRIBE-1 has a bit more descriptive information that it could make available fairly easily. The NAME's TOTAL weight could be *output*, to indicate its salience, and/or the program's certainty. The names of the CHARacterizers, the PARTs of their DESCRiptions that succeeded, and their locations could be stored with the names, and output as qualifying information (this will be done in DESCRIBE-*2*).

Weights and thresholds

DESCRIBE-1 TOTALs the weight of each PART of a DESCRiption (statement 11). The CHARacterizer succeeds if TOTAL exceeds the THRESHold (12). This allows the programmer to design characterizers that have any desired amount of looseness, in the sense of a threshold decision element that succeeds when any of a large number of combinations of subsets of its input PARTs succeed. If the threshold is lower than the weight of *any* of the PARTs, such a characterizer is equivalent to an "OR" operator; if the threshold equals the sum of all the weights of all the PARTs, it is equivalent to an "AND" operators.

The programs in this paper add weights together, since this is the simplest thing to do. But it is easy to have the program multiply weights, or compute whatever other combining function is deemed appropriate.

Note how similar are the THRESHold for deciding whether a CHARacterizer has succeeded and the CHOOSE level for deciding that a NAME has been sufficiently highly implied (by one or more characterizers) to be *output* as part of the program's description of the *input* PRESENT.

Describing vs. (merely) naming

The typical pattern recognition program simply chooses and outputs a single name that it assigns to the input.

This is usually done by having the program choose from MAYBE the *single* most highly implied NAME, rather than *all* the NAMEs implied above a CHOOSE level.

Sometimes the program will merely choose and output the first name whose TOTAL implied weight exceeds some minimum level for choosing. This would simplify DESCRIBE-1, which could now put the test that compares TOTAL weight with CHOOSE (statement 18) right after statement 15, and eliminate the DECIDE loop through MAYBE (statement 17)—with suitable changes in gotos.

DESCRIBING SCENES OF OBJECTS, AND THEIR PARTS OVER SPACE AND TIME

DESCRIBE-2 begins to handle descriptions *over time*, and descriptions that talk about the *parts* and the *subparts* of which the recognized wholes are composed.

To keep it short, it has been over-simplified, so that it handles only 1-dimensional string inputs (e.g., English sentences), and uses characterizers that do not handle position or interactions among parts (except to the extent that parts are explicitly put together, as though into a rigid template, and all possible combinations of this sort are added to its memory, as separate characterizers—a theoretically possible but practically unfeasible procedure).

In addition to describing the scene using all names implied above a CHOOSE level, it further describes each name by outputting all its parts (each with its column location) and all of each part's parts, until it hits the lowest level. It further does this from the point of view of each name—that is, it says in effect, "If this name is present, then these parts are present."

Time is handled by merging each PRESENT moment into a SEEN list, where OBJects are made salient (by high weights) when they first appear, and to the extent that they move, but then gradually FADE away when they are no longer present.

(OVERVIEW *DESCRIBE-2*. Builds short-term memory over TIME and space.		*DESCRIBE-2*
INITialize memory; UPDATE TIME, *input* PRESENT		M1-3
SENSE	Merge each OBJect in PRESENT into what has recently been SEEN, its WeighT a function of newness and movement.	4-8
FADE	Down-weight, and erase, OBJects no longer in PRESENT.	9-11
PERCEIVE	Get the IMPLIEDS names for each OBJect still in SEEN,	12-14
IMPLY	Put them on LOOKFOR and merge onto MAYBE, building a list of PARTS.	15-19
EVAL	*output* each NAME on MAYBE whose TOTAL weight exceeds its THRESHold.	20-23
DESCRIBE	*output* all PARTS of the NAMEd object, and PARTS2 of each, that have public TRANSforms (giving their COLumn locations).	24-28

```
(DESCRIBE-2. (Merges short-term SENSory memory over time and space.
(OBJects are salient when NEW or move; FADE out when no longer PRESENT
(Advance TIME, INPUT the new PRESENT)                                    1      2
                    (characterizers should go here)
UPDATE    erase COLP                                                    1.V    1
          set TIME=TIME+1                                               .1     2
IN        input PRESENT till ][-END]                                    2.V    3
(Merge each OBJ in PRESENT with those already SEEN)
SENSE     from PRESENT get OBJ=[-FADE]                                  .1     4
              COLP=COLP+1                                               .2     5
          from SEEN get # that OBJ # WT COL=[-NEW]                      .3     6
          on LOOKFOR list OBJ WT+ABS(COLP-COL) COLP [SENSE]             5.V    7
NEW       on LOOKFOR list OBJ 9 COL [SENSE]                             .1     8
FADE      from SEEN get OBJ WT COL=[-PERCEIVE]                          .2     9
          is WT lessthan 1? [+FADE]                                     .3     10
              on LOOKFOR list OBJ WT-1 COL [FADE]                       .4     11
PERCEIVE  SEEN=LOOKFOR                                                  .5     12
(Get NAMEs implied by each OBJect on LOOKFOR)
P1        from LOOKFOR get OBJ WT COL=[-ITERATE]                        6.V    13
          from $OBJ get 'I=' IMPLIEDS][-P1]                             7.V    14
(Put TOTAL of WeighTs for each IMPLIEDS NAME on MAYBE)
IMPLY     from IMPLIEDS get NAME WT=[-P1]                               13.V   15
          on LOOKFOR list NAME WT COL                                   .1     16
          from MAYBE get # that NAME # TOTAL PARTS]=[-I1]               14.V   17
              WTL=TOTAL+WTL                                                    18
I1        on MAYBE list NAME WT+WTL PARTS OBJ COL][IMPLY]               16.V   19
(EVALuate total WeighT against NAME's THRESHold, and output if greater)
EVAL      from MAYBE get NAME TOTAL PARTs]=[-UPDATE]                    17.V   20
          from $NAME get 'T=' THRESH                                    .1     21
              is TOTAL lessthan THRESH? [+EVAL]                         18.V   22
(Describes giving overlapping object NAMEs and overlapping parts and parts (of parts)
          output 'AT TIME='TIME' IT MIGHT BE='NAME DESCRIBED'           19.V   23
(outputs all the public PARTS, down to the lowest level)
DESCRIBE  from PARTs get OBJ COL=[-EVAL]                                       24
          from MAYBE get # that OBJ # TOTAL PARTS2][-DE1]                      25
              on PARTS set PARTS2                                              26
(If OBJect has an external TRANSform, OUTPUTs it)
DE1       from $OBJ get 'T=' TRANS][-DESCRIBE]                                 27
              output TRANS 'AT' COL [DESCRIBE]                                 28
END                                                                            —
LEFT-ARM TRUNK LEGS RIGHT-ARM EYE NOSE EYE CHIN]                               I1
DOG-EAR SNOUT LEG LEG]                                                         I2
TRUNK]                                                                         I4
LEG LEG TAIL]                                                                  I4
```

NOTE that size ABS(. . .) is a function that computes the ABSolute value of the expression within parentheses; it is defined and written by the programmer (though I don't bother to show the needed code).

DESCRIBE-2 needs a set of characterizers (put before statement 1) of the following simple sort, where each thing points up to the things implied:

```
LEFT-ARM='I=PERSON 3]'                                  M1
TRUCK    ='I=PERSON 5 DOG 3]'                           M2
EYE       'I=HEAD 3]'                                   M3
NOSE     ='I=HEAD 3]'                                   M4
HEAD     ='I=PERSON 5]P=EYE 5 EAR 3'
+                'NOSE 7 CHIN 4 MOUTH 6]'               M5
```

These (plus additional characterizers for LETS, CHIN, DOG-EAR, SNOUT, LEG) would allow it to notice PERSON and DOG. Since EYE, NOSE, etc. point to HEAD, and HEAD points to PERSON, it is also capable of describing a PERSON as having a HEAD, which has an EYE, NOSE, etc. (But I1 would not be described as containing an EAR.)

Note how characterizer M5 shows links from HEAD back to its parts (EYE, EAR, etc.)—links that are not actually

used. With additional code, *DESCRIBE-2* could add these to LOOKFOR, and use them to describe what is *missing* from a particular object.

To handle an interesting variety of inputs, we would need to give *DESCRIBE-2* a much larger set of characterizers. *DESCRIBE-1* is actually the much more powerful pattern recognizer since it handles configurations of interacting parts and does not rely upon a space (which is reasonable for sentences, and in fact typically used by parsing programs) to delimit objects.

Notice how a pattern can be recognized over time, even though at no single moment are all of its parts present, because characterizers look at the SEEN list, which only gradually fades away. Thus e.g., a suitable characterizer would output DOG in response to Inputs I2-I4. The description of the DOG (and of the PERSON Input in I1) would depend upon the particular parts present.

STILL FURTHER POSSIBILITIES FOR DESCRIPTIONS

We are now in a position to make a number of simple variations, as described below.

Describing one vs. several objects

DESCRIBE-2 describes rather exhaustively. For each NAME whose TOTAL weight exceeds THRESHold, and therefore is *output*, it also *outputs* all PARTS (that have public TRANSforms), and PARTS2 of parts, down to the lowest level. Thus the description can contain much overlap, of names and of parts. This can be varied in a number of ways.

A. The simplest would have the program output only one NAME of an object and its PARTS description:

```
(DESCRIBE-2-A. outputs only one object's NAME and
description of PARTS.                                  2-A
DESCRIBE from PARTS get
    OBJ COL=[-UPDATE]                                  24.V*
```

B. A slight change:

```
(DESCRIBE-2-B. Gives non-overlapping descriptions
    from MAYBE get # that OBJ # TOTAL PARTS2]
    =[-DE1]                                            25.V
```

would give non-overlapping descriptions of non-overlapping things, starting with whatever OBJect happened to come first on the MAYBE list. This would make more sense if the NAME with the MAXimum TOTAL weight were got from MAYBE in statement 20 (this entails a simple loop through the NAMEs on MAYBE, to get the one with the highest associated weight).

C. Still another variant would give overlapping descriptions of non-overlapping NAMEd objects:

```
(DESCRIBE-2-C. Describes several non-overlapping ob-
    jects                                              2-C
  set COPYM=MAYBE                                      20.1
  from COPYM get # that OBJ # TOTAL PARTS2]
    =[-TO DE1]                                         25.V
```

Details vs. wholes

D. All of these go from top down, from wholes to details. The following simple variant would dip down to details, and then go up:

```
(DESCRIBE-2-D. Dips down to give details first.        2-D
  at start of PARTS set PARTS2                         26.V
```

This gives a funny kind of order, wandering from top to bottom, and then back up.

E. A slightly more complex program would set all the PARTS2 at the start of an ALLPARTS list, and only then start developing the description, from ALLPARTS.

Keeping descriptions short

These all give descriptions that are far too long, since they contain all details. What is really needed is a system that chooses to output only the *pertinent* details—but this is an extremely complex matter, as will be discussed below, since it depends upon a deep semantic understanding of the objects in the scene, their import, and their import to the *hearer* of the description. So for now we can only examine the simplest of methods for keeping descriptions from growing unreasonably long.

F. First, we might have the program put only highly weighted OBJects onto the PARTS list (by checking the weight at statement 18)

G. Second, only parts of a specified QUALity might be *output*:

```
(DESCRIBE-2-G. outputs OBJect only if it is of
    the QUALity specified.                             2-G
  QUAL='SHAPE'                                         M1.1
  from $OBJ get that QUAL[-to DESCRIBE]                27.1
```

H. Third, the program might output only up to a fixed number of object parts:

```
(DESCRIBE-2-H. outputs only a specified
    Number of PARTS.                                   2-H
  ENOUGHP=I2                                           M1.1
UPDATE erase COLP, NPARTS                              1.V
  is NPARTS lessthan ENOUGHP?
    [-to UPDATE]                                       24.1
  NPARTS=NPARTS+1                                      24.2
```

I. Similarly, fixed numbers of objects, and/or of characterizers to be looked for, could be set (this is best done along with a function that gets the MAXimum implied).

* Numbers like 24.*V* indicate Variations to the corresponding statements in *DESCRIBE-2*.

GLANCING AROUND AND ACTING OVER TIME

Glancing, noticing, and focusing attention

DESCRIBE-2 "glances around," looking for higher-level wholes as a function of things already implied, because statement 16 puts an implied NAME onto LOOKFOR, so that the program will later look for any names that *it implies*, and so on up the hierarchy. (Note that this feature can easily be added to *DESCRIBE-1*.)

J. As a variant, we might use:

```
(DESCRIBE-2-J. Tends to LOOK FOR NAMEs at
    higher levels.                          2-J
  on LOOKFOR list NAME WT+WTL COL           16.1.V
```

(or some other function of the weights of both the NAME and the OBJect that implied it), so that a name at a higher level has a higher weight, reflecting the weights of all its lower levels.

K. Alternately, we might add the statement:

```
(DESCRIBE-2-K. Looks only with NAMEs chosen
    to output.                              2-K
  on LOOKFOR list NAME TOTAL                22.1
```

so that only at the end, if it has been chosen for *output* because its TOTAL implied weight has exceeded its THRESHold, is the NAME put onto LOOKFOR. This will put many fewer NAMEs on LOOKFOR, since it requires a more stringent procedure for evaluating the importance of each.

Glancing over time

It also has the interesting characteristic that it adds a NAME to LOOKFOR to be processed at the *next* moment of TIME (since it loops back to UPDATE in statement 20), whereas the addition after statement 16 will affect processing immediately, on the PRESENT moment.

We are thus beginning to introduce a second source of short-term-memory that gives continuity over time—not only in the SEEN list, that only gradually fades away, but also in the LOOKFOR list.

L. Still other variants would have the program a) add implied names to a NEXTLOOK list at 16.1, and only at UPDATE time set LOOKFOR=NEXTLOOK, so that the casually got names would not be processed until the next time: b) loop back from EVALuation to PERCEIVE some more *if* new NAMEs have been put onto LOOKFOR (22.1), so that they are processed immediately, at *this* time.

DISCUSSION

The short-term perceptual memory

Merging of OBJects from the recent past into the SEEN list, where their salience is a function of their newness and motion, and they fade away only slowly after having disappeared from the environment, appears to be simple, elegant, and sufficient to allow systems to handle environments that continue and change over time. But it may also be useful to introduce further inertia over time by using a separate list of CHARacterizers to LOOKFOR, where LOOKFOR also continues over time. There are many interesting alternative possibilities here, only a few of which have been touched upon in this paper.

Focusing attention and noticing

As soon as we let a program add characterizers to its LOOKFOR list we introduce a whole range of possibilities for focusing attention and concentrating on certain things, and type of things. In *DESCRIBE-2* what has already been noticed implies new characterizers, and new objects, which can themselves imply their parts, and characterizers that would imply them.

We can, if we wish, *initialize* our programs to contain one or more names of objects to LOOKFOR. This will focus attention on these objects, and on the characterizers that imply them, and their subparts. The strength of this focusing will be a function of their weights. Depending upon details of thresholds and choose levels, the system will now find more of the things it has been set to look for, with less certainty thus leading to false positives, and it will tend not to notice other things—all rather reminiscent of human beings.

The influence of internal needs and external suggestions

These systems are now in a position to have their processes influenced from a variety of sources. Commands and conversational suggestions from the external world can suggest what to look for, and what kinds of descriptions to output. Internal needs and goals can also play a role. In all cases, the various sources of set are merged into the LOOKFOR list, which controls processing.

Changing points of view

People will tend to describe scenes as though they are fields of physical objects, with only one object at one place at one time. We will point to and describe a number of faces, but without adding, "oh there's still another face that's made of the left ear of face 3 and the right chin of face 7"; nor will we go on to describe the details of faces, saying "there's a leaf; there are two fish."

But we can very easily shift to different attitudes. For example, if we're shown a drawing and told it contains 82 hidden faces and 212 leaves, we will almost immediately see overlaps.

The germ of this ability appears to lie in the different variant attitudes for outputting overlapping, or non-overlapping, descriptions in *DESCRIBE-2-A* through *2-E*. What still needs doing is to give the program control over which of these attitudes it will take, and let it decide as a function of a

variety of pieces of information that it has gathered during its conversational interaction with its environment, including the hearers of its descriptions.

Recognizing and acting over time

It is unrealistic to have a system apply no matter how many characterizers, and output no matter how complex a description, in a single moment of time. Time is needed to recognize, describe, notice, and act. *DESCRIBE-2-K* and *2-L* begin to take this into account, but once again there is a large variety of other possibilities. Ideally, we should consider what is the common real time in which the environment, the program's perceptual processes, and its motor actions must all take place, and we should assign appropriate real times to each separate process. This makes apparent the issues of parallel vs. serial vs. parallel-serial processes, and time needed for feedback loops that monitor action.

What is a description?

The concept of a "description" is hazy, and not easily defined. Most people will look at a scene and say something like, "There's a man walking a dog through the woods." A fastidious few will say instead, "There's a man with fingers circling around the loop of a leash, whose other end appears to be attached to a dog via a collar; there are also 5 trees in the picture." A detective might say, "There's a tall man in a coonskin cap with a black handle-bar mustache and a smudge on his left cheek," while a dog-nut might say, "There's a Siberian husky with eyes that are too slanted," and a nature lover, "there's a mixture of honeysuckle, maple and pine, and it looks like Spring, but I don's see any birds."

It is hard to conceive of a description in which the describer does not (1) make major *judgments* as to what is *important* and, further, (2) superimpose his own *"understanding"* of the objects in the scene, and their interrelationships and their *import*.

Sometimes the scene will be impoverished to the point where things seem relatively simple, at least on the surface. Thus almost everybody will look at a sheet of paper on which letters have been written and say, "There's an 'E' " or "There's the word 'THE' ". If we press further most people will say, "The 'E' has a vertical bar with short horizontal bars extending to the right from top, middle and bottom"—if it is a standard, well-drawn 'E'—and they will think us a bit crazy for asking (why?—I think because such a description feels like a tautology, possibly because it is a *constructive definition* of an 'E', one that we have pretty generally agreed to use).

If the E is sloppy, and/or we press the describer to say more, we will begin to get statements like, "The top bar wavers and has breaks," "It's long and skinny." A more compulsive person might say, "The top bar angles down 20° for ¼ inch, then curves up for ½ inch, until it is ¼ inch above its start, then goes straight for ½ inch."

This description probably sounds contrived to most readers. But that leads into a third important characteristic of descriptions. Once we are pushed beyond the ordinary level of description we must grope for terms and framework. In general, the describer (3) says what he *infers* his *hearer* will consider *pertinent*. Thus the diagnostician will tell the neurologist, "the wavering strokes have the quality of palsy rather than brain damage", but he will tell the accountant, "the smudges come because the pencil lead is too soft."

Description is now squarely in the middle of conversational interaction—just where I think it should be, but this raises even more complex and subtle problems. Now we must worry not only about (1) the actual objects in the scene, and their parts and relations, and (2) the describer's understanding of the objects in the scene, but also (3) the *hearer's* understanding of these objects, (4) the describer's understanding of the hearer's understanding, and even (5) the hearer's understanding of the describer's understanding of the hearer's understanding. For example, the dog-nut might assume that the hearer is a secret dog-nut, or at least realizes that many people are dog-nuts and probably also the describer. And the hearer might infer that the describer knows he the hearer likes dogs, and wants to suck him into an interest in the fine points.

In addition to arguing for the complexity and subtlety of a description, this is to argue that it is intimately related to a rich semantic understanding that describer and hearer have in common—at least to some extent, along with an understanding by each of the situation of communication, in which one tries to impart suitable information to the other.

We cannot expect pertinent, sensitive descriptions until we have programs that have the necessarily rich semantic understanding of the world whose scenes are being described, and of what this world means to their hearers. But we can still extend pattern recognition programs that merely name to the point where they also describe, albeit either in exhaustive detail or in overly-rigid conventional ways. And we can begin to tailor their descriptions to their hearers, as a result of simple conversational interactions.

APPENDIX—A NOTE ON PROGRAMS[35]

1. Numbering at the right identifies statements, and allows for comparisons between programs. M indicates initializing *M*emory statements: I indicates cards that are *I*nput by the program. .V indicates a *V*ariant, .1 an additional statement.
2. A program consists of a sequence of *statements*, and *END* card, and any *data* cards for *input*. (Statements that start with a parenthesis are *comments*, and are ignored.) Statement *labels* start at the left; *gotos* are at the right, within brackets (+ means branch on *success*;—on *failure*; otherwise it is an *unconditional* branch).
3. Strings in capitals are programmer-defined. Strings in *underlined* lower-case are system commands that *must be present* (they would be keypunched in caps to run the program). These include *input, output, erase, set, list, get, start, call, that,* and the inequalities. Other lower-case strings merely serve to help make the program understandable; they could be eliminated.

4. EASEy automatically treats a space following a string as though it were a delimiter; it thus automatically extracts a sequence of strings and treats them as names. The end-bracket] acts similarly as a delimiter, but the programmer must specify it. The symbol # is used to stand for *any* delimiter (a space,] or #).
5. The symbol $stringI is used to indicate "get the *contents* of string I, and treat it as a name and get *its* contents" (as in SNOBOL).
6. Pattern-matching statements work just like SNOBOL statements: there are a) a name, b) a sequence of objects to be found in the named string in the order specified, c) the equal sign (meaning replace), and d) a replacement sequence of objects (b, c, and/or d can be absent). *that* string I means "get that particular object"—otherwise *a new string is defined* as the contents of stringI, which is taken to be a variable name.

REFERENCES

1. Andrews, D. R., Atrubin, A. J., Hu, K., "The IBM 1975 Optical Page Reader: Part III: Recognition and Logic Development," *IBM J. Res. and Devel.*, 1968, *12*, pp. 364-372.
2. Ausherman, D. A., Dwyer, S. J., Lodwick, G. S., "Extraction of Connected Edges from Knee Radiographs," *IEEE Trans. Comput.*, 1972, *21*, pp. 753-758.
3. Brice, C. R., Fennema, C. L., "Scene Analysis Using Regions," *Artificial Intelligence*, 1970, *1*, pp. 205-226.
4. Eden, M., "Handwriting Generation and Recognition," in *Recognizing Patterns* (P. A. Kolers and M. Eden, Eds.), Cambridge, MIT Press, 1968.
5. Firschein, O., Fischler, M. A., "Describing and Abstracting Pictorial Structures," *Pattern Recognition*, 1971, *3*, pp. 421-443.
6. Fischler, M. A., "Machine Perception and Description of Pictorial Data," *Proc. 1st Joint Int. Conf. Artificial Intell.*, Washington, D.C., 1969, pp 629-635.
7. Forgie, J. W., Forgie, C. D., "Results Obtained From a Vowel Recognition Computer Program," *J. Acoust. Soc. Amer.*, 1959, *31*, pp. 1480-1489.
8. Frishkopf, L. S., Harmon, L. D., "Machine Reading of Cursive Script," in *Proc. 4th London Symposium on Information Theory* (C. Cherry, Ed.)., London: Butterworth, 1961, pp. 287-299.
9. Fu, K. S., Swain, P. H., "On Syntactic Pattern Recognition," in *Software Engineering, Vol. 2*, (J. T. Tou, Ed.) New York: Academic Press, 1971.
10. Grimsdale, R. L., Sumner, F. H., Tunis, C. J., Kilburn, T. A., "System for the Automatic Recognition of Patterns," *Proc. Inst. Elect. Engrs.*, 1959, *106 (pt. B)*, pp. 210-221.
11. Guzman, A., "Decomposition of a Visual Scene into Three-Dimensional Bodies," *Proc. AFIPS FJCC*, 1968, *33*, pp. 291-304.
12. Hall, D. J., Endlich, R. M., Wolf, D. E., Brian, A. E., "Objective Methods for Registering Landmarks and Determining Cloud Motions from Satellite Data, *IEEE Trans. Comput.*, 1972, *21*, pp. 768-776.
13. Hall, E. L., Kruger, R. P., Dwyer, S. J., Hall, D. L., McLaren, R. W., Lodwick, G. S., "A Survey of Preprocessing and Feature Extraction Techniques for Radiographic images," *IEEE Trans. Comput.*, 1971, *20*, pp. 1032-1044.
14. Hunt, E. B., *Concept Formation: An Information Processing Approach*, New York: Wiley, 1962.
15. Kirsch, R., "Computer Interpretation of English Text and Picture Patterns," *IEEE Trans. Comput.*, 1964, *13*, pp. 363-376.
16. Kochen, M., "Experimental Study of 'Hypothesis-Formation' by Computer," *Trans. 4th Sympos. on Info. Theory* (C. Cherry, Ed.), London, Butterworth, 1960.
17. Ledley, R. S., "Analysis of Cells," *IEEE Trans. Comput.*, 1972, *21*, pp. 740-753.
18. Ledley, R. S., Ruddle, F. H., "Chromosome Analysis by Computer," *Sci. Amer.*, 1965, *40*, pp. 40-46.
19. Lillestrand, R. L., "Techniques for Change Detection," *IEEE Trans. Comput.*, 1972, *21*, pp. 654-659.
20. Lipkin, B., Rosenfeld, A., *Picture Processing and Psychopictorics*, New York: Academic Press, 1970.
21. Lipkin, L., Watt, W. Kirsch, R., "The Analysis, Synthesis, and Description of Biological Images, *Ann. N.Y. Acad. Sci.*, 1966, *128*, pp. 984-1012.
22. Londé, D., Simmons, R., "NAMER: A Pattern-Recognition System for Generating Sentences About Relations Between Line Drawing," *Proc. ACM 20th National Conf.*, 1965, pp. 162-175.
23. Marill, T., Hartley, A. K., Evans, T. G., Bloom, B. H., Park, D. M. R., Hart, T. P., Darley, D. L., "CYCLOPS-1; A Second-Generation Recognition System," *Proc. AFIPS FJCC*, 1963, *24*, pp. 27-34.
24. Mermelstein, P., *Computer Recognition of Connected Handwritten Words*, Unpubl. Sc.D. Thesis, MIT, 1964.
25. Narasimhan, R. "Syntax-Directed Interpretation of Classes of Pictures," *Comm. ACM*, 1966, *9*, pp. 166-173.
26. Narasimhan, R., Reddy, V. S. N., "A Syntax-Aided Recognition Scheme for Handprinted English Letters," *Pattern Recognition*, 1971, *3*, pp. 345-362.
27. Reddy, D. R., "Computer Recognition of Connected Speech," *J. Acoust. Soc. Amer.*, 1967, *42*. pp. 329-347.
28. Roberts, L., "Machine Perception of Three-Dimensional Solids," in *Optical and Electro-Optical Information Processing* (J. T. Tippett et al., Eds.), Cambridge, MIT Press, 1965.
29. Rosenfeld, A., *Picture Processing by Computer*, New York: Academic Press, 1969.
30. Sauvain, R., Uhr, L., "A Teachable Pattern Describing and Recognizing Program," *Pattern Recognition*, 1969, *1*, pp. 219-232.
31. Shaw, A. C., "A Formal Picture Description Scheme as a Basis for Picture Processing Systems," *Info. and Control*, 1969, *14*, pp. 9-52.
32. Smith, E. A., Phillips, D. R., "Automated Cloud Tracking Using Precisely Aligned Digital ATS Pictures," *IEEE Trans. Comput.*, 1972, *21*, pp. 715-730.
33. Sutton, R. N., Hall, E. L., "Texture Measures for Automatic Classification of Pulmonary Disease," *IEEE Trans. Comput.*, 1972, *21*, pp. 667-676.
34. Towster, E., *Studies in Concept Formation*, Unpubl. Ph.D. Diss., Univ. of Wis., 1969.
35. Uhr, L., *A Primer for EASEy (An Encoder for Algorithmic, Syntactic English that's easey*, Computer Sciences Dept. Tech. Report, Univ. of Wis., 1973a.
36. Uhr, L., "Feature Discovery and Pattern Description," in *Pattern Recognition*, (L. Kanal, Ed.). Washington: Thompson, 1968, pp. 159-181.
37. Uhr, L.,"Flexible Linguistic Pattern Recognition," *Pattern Recognition*, 1971, *3*, 363-384.
38. Uhr, L., *Flexible Pattern Recognition* Computer Sciences Department Technical Report 56, Univ. of Wis., 1969.
39. Uhr, L., "Layered Recognition Cone Networks that Preprocess, Classify and Describe," *IEEE Trans. Comput.*, 1972, *21*, pp. 758-768.
40. Uhr, L., *Pattern Recognition, Learning and Thought*, Englewood-Cliffs: Prentice-Hall, 1973b.
41. Uhr, L., and Vossler, C., "A Pattern Recognition Program that Generates, Evaluates and Adjusts its Own Operators," in *Computers and Thought* (E. Feigenbaum and J. Feldman, Eds.), New York, McGraw-Hill, 1963, pp. 251-269.
42. Zobrist, A. L., "The Organization of Extracted Features for Pattern Recognition," *Pattern Recognition*, 1971, *3*, pp. 23-30.

Tuning the hardware via a high level language (ALGOL)

by RONALD BRODY

Burroughs Corporation
Paoli, Pennsylvania

ABSTRACT

This paper will discuss how the computer may be tailored to the problem it is solving while still programming the problem in a higher level language. The vehicle to illustrate the technique is the "Stack machine," which is a virtual machine implemented in firmware on a Burroughs microprocessor. The "Stack machine" was designed to compile and execute programs written in Algol. Since, the microprogram memory of the microprocessor is read/write, the stack machine can be extended by special purpose firmware operators designed for a particular application program. This system permits jobs to be programmed and debugged in Algol, and then fine tuned to run at a much greater speed.

This paper describes the stack machine, the methods used to fine tune a program, and some sample results of fine tuning.

Computer on a chip and a network of chips

by GARY HUCKELL

Naval Electronics Laboratory Center
San Diego, California

ABSTRACT

The paper will first discuss the various approaches to a computer on a chip as advanced by the various semiconductor vendors and in the various R&D programs sponsored by the government. This discussion will include a description of the work being performed for the Navy who are developing the technology for a 5000-10,000 gate chip. Among the questions to be discussed are: given the fact that 10,000 gates can be packaged, what function should it contain? Should it be independent and programmable or should it be a part of a system?

Following the chip discussion the subject of the ways and the whys of interconnecting these chips will be discussed.

10^{-5}—10^{-7} cent/bit storage media, what does it mean?

by JOHN C. DAVIS

Department of Defense
Ft. George Meade, Maryland

ABSTRACT

This paper will discuss the various advanced mass memory techniques and will attempt to put them in perspective as to the reasons why the approaches are being pursued. The techniques to be covered will include systems based on tape, the various laser activated systems, and the holographic memory approaches. At least one of these approaches is directed at a homogeneous replacement for the entire memory hierarchy of the present day system. Other random access memories can reduce the need for conventional main frame memory if the system programmer properly writes his operating system. The low cost (10^{-6} cents/bit) memories will permit the economic storage of large data bases for a variety of users and will force the system architect to solve the large storage management problem for the mini computer user.

Computer architecture and instruction set design*

by P. C. ANAGNOSTOPOULOS, M. J. MICHEL, G. H. SOCKUT, G. M. STABLER, and A. van DAM

Brown University
Providence, Rhode Island

INTRODUCTION

A group of computer scientists and mathematicians at Brown University has been engaged in the study of computer graphics for the past eight years. During the course of these studies a variety of topics has been investigated, in particular, during the last few years, the use of microprogramming for implementing graphics systems.[20,21] In early 1971, Professor Andries van Dam and his associates submitted a threefold research proposal to the National Science Foundation. The problems to be investigated were:

(1). Inter-Connected Processing (ICP-ing) between a central computer and an associated satellite processor, with the goal of a dynamically alterable solution to the "division of labor" problem; program modules would be dynamically linked in either machine as a function of availability and cost of resources and response time;
(2) Programming aids at the source language level for the automatic generation of data structure manipulation subroutines and symbolic debugging of data structure oriented applications programs;
(3) The development and use of the Language for Systems Development (LSD),[22] a high-level systems programming language, for generating the applications *and* systems software for both the central computer and the satellite in such systems:

An interactive graphics system is an excellent paradigm for such investigations since graphics applications.

(1) are typically very large in terms of memory space required;
(2) maintain large data bases, many with intricate (list-processing oriented) data structures;
(3) have processing requirements that change dynamically, varying from very heavy (e.g., structural analyses of a bridge) to very light (e.g., inputting a command); and
(4) require real-time response.

The Brown University Graphics System (BUGS)[18] was designed as the vehicle for performing this research. Principally, the configuration consists of an IBM S/360-67 running the CP-67/CMS time-sharing system,[10] used by the entire Brown University community, and a satellite display station, as illustrated in Figure 1. This reasonably powerful satellite configuration provides such facilities as program editing and compilation, debugging tools, and most importantly, application processing power and data storage. However, because of the two rather distinct demands placed upon the local processor, that of display generation and general computing, and because these two capabilities could run in parallel, it was further determined that the inclusion of two separate processors in the graphics station would be in order. In particular, the first of these processors would be of a general-purpose nature, while the second would be designed specifically for maintenance and regeneration of the display. Figure 2 illustrates the division of these processing capabilities. Unfortunately, the configuration shown in Figure 2 was far removed in scope from any commercially available equipment, and the purchasing of a general-purpose computer from one manufacturer, a graphics processor from another, and perhaps even a display from a third would prove not only unworkable in terms of compatability, interfacing, and programming, but also unadaptable to the implementation desired. It became apparent that it would be necessary to design the satellite system from the ground up. This could be accomplished by building the hardware at Brown; however, the lack of engineering manpower ruled out this possibility. The one other method that could be employed would be to purchase a pair of user-microprogrammable host computers; a few such computers were available at the time. Microprogrammable computers provide the system designer with the hardware upon which he can base a novel system, presenting him with the opportunity, but also the problem, of writing software from the ground up, and with actually designing and implementing his own target architecture and instruction set.

The problem of computing system architecture has been of major importance since the dawn of computers in the late 1940's. The computer user, however, has had little or nothing to do with this problem; scientists and engineers at the manufacturing companies (or universities) have done all the design in seclusion. Once designed,

* This work is sponsored in part by the National Science Foundation, grant GJ-28401X, the Office of Naval Research, contract N000-14-67-A-0191-0023, and the Brown University Division of Applied Mathematics.

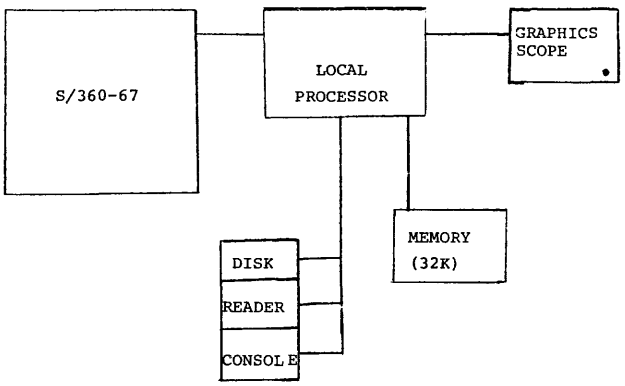


Figure 1

it was then up to salesmen to sell the machine to the unsuspecting public, which accepted it on faith or out of necessity.

Over the last ten years things have begun to change. People have realized that their applications, be they business data processing, process control, or bio-medical research, are distinct and have peculiar computational requirements. The advent of the reasonably cheap minicomputer has allowed users to program their own monitor systems and software packages, oriented toward their specific needs. Regardless, the target architecture of these machines was still fixed and unchangeable, and could not be tailored to a user's specific needs in order to increase effectiveness. However, this latter problem is now being alleviated by the introduction of user-microprogrammable host computers. The purpose of such computers is to allow the user himself to design an appropriate target architecture and instruction set for this particular application, implement this architecture, and perhaps change it after he has learned more about what he needs. A good overview of microprogramming in general is found in Reference 16. Microprogramming trade-offs for user applications are discussed in Reference 5.

It is at this point that a clear distinction between a target architecture and a target instruction set must be made. The architecture defines the basic relationships

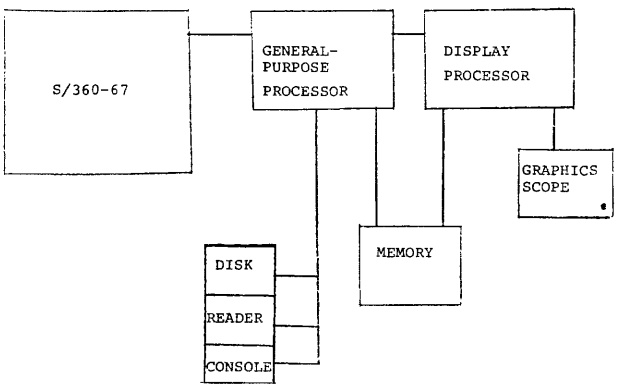


Figure 2

between the various components of the machine, e.g., storage, registers, control, arithmetic units, etc. On the other hand, the instruction set is simply the array of discrete operations which may be utilized by the programmer. A specific example is the comparison of the Burroughs family of stack machines[3] and the IBM S/360 family.[9] The target architectures are entirely different, whereas the instruction sets are similar.

The purpose of this paper is to discuss the problems of machine architecture and instruction set design in general, while referring to the specific BUGS implementation. Based on this discussion, a set of ideas and suggestions is presented to form an initial guide for future implementers of microprogrammed machine architectures.

CHOOSING A MICROPROGRAMMABLE HOST COMPUTER

As stated in the Introduction, much of the rigidity of conventional computers can be overcome if the user is willing to microprogram his own target architecture and instruction set. Although it has been said too many times already, it remains necessary to point out that the aura of complexity surrounding microprogramming is purely a product of scientific mysticism. Microprogramming is not much more than fairly conventional programming at a different level, perhaps requiring greater attention to efficiency;[12] anyone who has coded a simulator or interpreter has already programmed at that level. Microprogramming therefore, being programming at a lower level, transforms the problem of rigidity of the target level architecture into the lower-level problem of host architecture rigidity. After all, how can one design a 24-bit target architecture if he knows it will be implemented on a 16-bit host? And one might as well give up if a decimal machine is desired without decimal hardware in the host. Such conflicting features are not impossible to implement, but they will be extremely inefficient and difficult to microprogram.

At the time the microprogrammable hosts for BUGS were chosen, there were none available that were sufficiently adaptable to allow a wide choice of target architectures. In other words, the rigidity of the host architecture limited the range of target architectures almost entirely to the standard Von Neumann variety. Most users would not consider this limitation a hindrance; they are used to standard architectures and would be at a loss to design an alternate one. However, it is becoming more and more apparent that the barriers to increasing computational effectiveness today are a factor not so much of the crudity of the instruction set as of the unyielding nature of unadaptable hardware. Even the simplest instruction set can simulate a Turing machine and hence compute any function, but the ease with which these functions can be performed depends on the overall blend of machine facilities. Burroughs has begun an attempt at solving the rigidity problem with the introduction of the

B1700 variable-micrologic processor,[23] which takes a first step toward eliminating certain inherently structured components. However, the B1700 cannot as yet be considered an inexpensive user-microprogrammable computer.

All in all, there were four hosts from which to choose, including the Interdata Model 4[11] Microdata 800[13] Digital Scientific META 4,[6] and the Nanodata QM-1.[15] It is immediately apparent that the machines vary widely in architecture. Our consideration was narrowed down very quickly by the fact that the Interdata and Microdata machines have two major deficiencies. The first is the 8-bit microregisters, which would prove horribly inefficient for implementing the 16- or 32-bit arithmetic required for even basic numerical computing. Two or four registers would be required per operand, and multiple-precision arithmetic would have to be performed. The second deficiency is the unavailability of on-site user microprogramming (let alone writeable control storage), making experimentation and redesign virtually impossible. For these two reasons the choice was narrowed down to the META 4 and the QM-1.

The QM-1 had two major features in its favor. The first was the abundance of microregisters and large amount of storage, while the second is that of writeable control storage. However, it appeared that the machine would not be available for many months, whereas the META 4 was immediately deliverable. Furthermore, the impressive control cycle speed of the META 4 was enticing. On these grounds it was decided to purchase two META 4's, the first of which would be the general-purpose processor called the "META 4A"[2] and the second the graphics processor called the "META 4B."[19]

DESIGNING A TARGET ARCHITECTURE

Designing a target architecture should not have to be a major research effort. Unfortunately, however, there is a plethora of considerations which will greatly affect the ultimate usefulness of any design, and yet there are no available guidelines to help cope with them. If these considerations are not dealt with and ultimately synthesized in a reasonable manner, the architecture may fail to be of any use whatsoever.

At first glance, the most important consideration may appear to be the application for which the architecture is intended. Although every application requires certain basic computational capabilities, the strength of a specific architecture lies in its ability to simplify the problems at hand. For example, process control requires a fast and flexible interrupt handling mechanism, whereas information retrieval necessitates powerful data structure manipulation operations. So it might well be concluded that an optimal architecture contains basic arithmetic, logical, and decision-making tools plus facilities oriented toward the ultimate application(s).

The above assumption should be examined at a lower level. Suppose there is a machine with an instruction (call it SEARCH) which scans a linked list for an entry with a specific key. Such an instruction is immensely useful for operating systems with queue-searching requirements, for information retrieval, or for computer graphics. Consider now the level of programming available for the machine. If programmers are coding in assembly language, the instruction can probably be utilized; the determination of when it can be used is up to the programmer. However, if a higher-level language is available, it may be impossible for even a very sophisticated compiler to determine when such an instruction can be generated without an explicit SEARCH primitive, because the fact that the programmer is performing a queue search is hidden in a four or five statement loop. A vast amount of research concerning the relationship between compilers and instruction sets has yet to be conducted.

The SEARCH difficulty is only indicative of a basic contradictory aim in the current design of computers. In most cases, the designers are thinking in terms of assembly language programming, and hence produce an instruction set with an abundance of special-purpose operations that can be used only by a resourceful assembly language programmer. As soon as the compiler designer begins considering the type of code his compiler is to generate, however, these instructions prove useless and perhaps cumbersome.

So where does that leave us? It is at this point that the user must decide how much expertise he has available, how much time he is willing to devote, and how much money he has to spend. If he decides to bend over backwards, then he can purchase something like a B1700, spend time experimenting with the architecture design, and probably produce a fine target machine. Indeed, such a processor is a particularly convenient vehicle for tackling the hardware-firmware-software problem,[14] i.e., the problem of how to distribute the processing function between hardware, microprogram, and software for optimal performance. Many users might complain that they have not the money nor desire to spend excessive time in the design of a system. In this case a somewhat narrower approach to a conventional architecture is in order, with perhaps only a few basic improvements. If the user chooses to go the B1700 route, then the conflict between assembly and high-level languages can be eliminated by maintaining multiple emulators, one for each language. However, most users will probably choose the more conventional direction, in which case the problem still remains, and has a few solutions. If only assembly language will be used, then there is no reason not to include SEARCH. The trend today, though, is toward higher-level languages, particularly with programmers becoming more enlightened to the successes of the structured programming approach.[3,7] The compiler designer can simply ignore the SEARCH instruction, or he can add a SEARCH statement to the language. If he chooses to ignore it, then its inclusion in the instruction set is questionable and must be reconsidered. In this manner, each

special-purpose feature of an architecture must be evaluated separately to determine its ultimate usefulness.

Another major consideration is that of the ultimate speed of the target machine. Most users might immediately say that "the faster it goes, the better I like it." Many applications indeed require such speed, particularly real-time systems. Unfortunately, because of the aforementioned rigidity of host processors, it is impossible for a user to reduce emulation time by putting often-used functions into the hardware. In the work at Brown it has been found that, because of this fact, the speed of the individual functions in a target machine varies directly with their complexity. A good example is that of memory addressing schemes. Simple absolute addressing is extremely fast, while base register-displacement addressing is much slower. Add on an index register and an indirect flag and memory referencing will slow to the speed of cold molasses. The question that must be asked about each of these functions is: How much speed am I willing to trade off for useful complexity? You may come to the conclusion that speed is all important. Then again, programming and compilation ease may be the biggest factor, in which case a more powerful instruction set is desired. It must be kept in mind that the execution speed of a simple, fast architecture and a complex slower one may be equalized by the fact that the slowness of the latter is made up for by its more powerful instructions, i.e., the faster machine requires more instructions to perform the same function. This implies an advantage to choosing the latter architecture, since programs coded on it should be generally smaller than those coded on the other architecture (see the IBM 1130/META 4A benchmark in a later section for a specific example).

One of the most probable misconceptions in evaluating speed requirements is that of determining how much computing per unit time is actually going to be done. If the application is input/output bound, or if it processes human requests, a somewhat slow CPU may go completely unnoticed. Another possibility is that of a multiprocessing system—upon consideration it may be determined that one processor can be slow and more complex even if the other needs to be fast.

An unfortunate problem with most microprogrammable processors today is the very limited amount of control storage which can be included (one to four thousand words in most cases). Once a basic target instruction set is microprogrammed, there may be little space left for application-oriented or experimental facilities. As in all programming, the space/time tradeoff is thus present, requiring the speed and space considerations to be evaluated in parallel.

The above considerations lead directly to a related consideration, that of tuning the target architecture to fit the host computer. For reasons of speeding up the target machine and simplifying the microprogramming task, certain functions that the host machine does poorly should be avoided. One example is a host computer without bit testing facilities; this suggests that a TEST BIT target instruction would be unwieldy. Another example is that of the word size of the target machine—it should optimally be the same as the host machine word size, and at worst a multiple thereof. The speed consideration is by far the most frustrating of all. It may require leaving out many features of the target machine that would otherwise be desirable, simply because they are uselessly slow or impossible to implement. It has also been shown that many functions may run almost as fast in the software as in the firmware,[5] in which case, for the purpose of saving control storage or for making the function more easily changeable, they should not be microprogrammed.

A final consideration is that of the human programming factor. There may well arise a situation in which there are a handful of expert programmers trained on a specific machine, but it is decided for one reason or another to replace the machine with a microprogrammable processor. Certainly, if this new processor were to support a target architecture similar or identical to the original machine, the programmers would be doing useful work much sooner than if they had to be retrained. Furthermore, any existing software packages could be converted in much less time, a factor that may well prove to be the saving or the death of the conversion. Perhaps the new processor is to run as a satellite to a bigger computer. In this case, programmers may be writing assembly language code for both machines. If they had identical instruction sets, then these programmers' sanity could be preserved, whereas if they were somewhat different the programmers may not be able to switch machines with much alacrity. Furthermore, similarity between the two machines would allow compilers to be written which could produce optimized code for both machines using identical algorithms.

The evaluation of all these interrelated considerations can add up to a staggeringly complex problem. Indeed, many decisions cannot be finalized until after the system is implemented and used for a while. If the host computer is equipped with writeable control storage, post-implementation decisions are no problem. However, most hosts available today do not have such control storage, so that the design must be fairly well finalized before it is implemented. The best tool in this case is a good microcode simulator for the host, equipped with timing and debugging features.[24] Using the simulator, small applications programs and system software can be written in the target instruction set and tested. This simulation should turn up not only design and microprogramming errors, but also help determine the usefulness of experimental features and perhaps point out missing features. Uncountable hours of headaches can be saved in this way.

The first design of the BUGS general-purpose processor (META 4A) began with what were thought to be fairly concrete decisions concerning the considerations discussed above:

Application facilities. The intent of the META 4A design was to produce a general-purpose processor which could

support a variety of applications, the most important of which was graphics (keeping in mind that actual display regeneration was to be done with the META 4B). Therefore, complete data structure searching and manipulation operators, plus operators for manipulating arbitrary length character strings were included. In addition, requirements for communication with the IBM S/360-67 necessitated the inclusion of a microprogrammed interface between the META 4A and the IBM multiplexor channel, plus target instructions to control this communication.

Programming languages. This area was of definite concern in choosing the target instruction set. The ultimate goal was to use the LSD language for all programming, but, it would not be available for at least a year. Hence, for the interim, a powerful assembly language was needed, but it was necessary to think ahead and include facilities useful to a compiler. Unfortunately, the knowledge of just what these facilities should be was inadequate due to the fact that the compiler was only partially designed and partially implemented at the time. A limited set of instructions for procedure entry and exit were included, plus the idea of automatic storage was formalized and included in the firmware. Furthermore, it was decided to go ahead and include whatever instructions would be useful for assembly language programmers, and simply let the designers of LSD ignore them if they were of no use.

Speed. Because the actual graphics display regeneration was to be done in the META 4B, it was felt that the speed of the META 4A was not as crucial as its power and flexibility. The overall philosophy was to derive as much speed as possible without deterring from producing a powerful and easy-to-use instruction set.

Emulator size. The size of the emulator was limited to 1500 microinstructions due to available funds. For this reason there was not much choice but to code so as to save control storage space at the expense of speed.

Host considerations. The META 4 host seemed general enough so that any feature could be implemented; as it turned out, this was definitely not the case. Examples of the inadequacies that were discovered are discussed in the following section.

Human factors. This, too, was a distinct problem. Programmers would be working on three separate processors (S/360-67, META 4A, and META 4B) in parallel, and therefore the idea that the local processors should look like S/360's was a strong one. On the other hand, these programmers were also experienced on other processors and felt that working in two different environments simultaneously would not be entirely out of the question. It was decided that architectural similarity was of only secondary importance.

A FIRST ATTEMPT

The first design of the META 4A was begun by considering the great variety of computers already on the market and classifying them into categories. The evaluation of these computers would allow a choice of a base architecture, which could then be improved and customized in light of the considerations outlined previously. The following basic architectures were considered:

IBM 1130-like. This category is considered to be composed of computers with relatively simple architectures that would be easy to implement and run efficiently on the META 4. Such things as simple addressing schemes, short instruction formats, and a small number of instructions would contribute toward this ease and efficiency. On the other hand, programming on such a machine would be slow and tedious, and there were no high-level facilities for use by the compiler designer. Furthermore, the integration of data structure and character manipulation features would be difficult, due to the lack of a sufficient variation of instruction formats and too few operation codes. Experience with IBM 1130's, and the fact that the META 4 host was reasonably powerful, indicated that this was not the way to go.

IBM S/360-like. In this category were computers with more complex architectures, offering the programmer more instructions and more power. Such an architecture may include multiple target registers, general addressing schemes, and a wider range of application facilities, such as character manipulation, that make the programming problem simpler and smaller. However, with this power came complex instructions that require more time to emulate and more control storage to contain the emulator. One advantage to be gained by emulating an instruction set like that of the S/360 was the pre-existence of useful software such as assemblers and linkage editors. In a previous microprogramming project[V2], an S/360-like instruction set had been microprogrammed on an Interdata Model 3 with reasonable success.

DEC PDP-11-like. This category basically included only the PDP-11 family[D3]. It was considered separate and distinct simply because the PDP-11 contained a blend of features not found on other machines, such as stack operations and a highly flexible addressing mechanism. Instructions were generally variable in length, so that only necessary fields need be included—this was felt to be an advantage since the BUGS configuration had only 32K bytes of storage. Some considered the variable formats to be unnecessarily complex and confusing; one prospective user went so far as to state that he would refuse to code for the system if such an architecture were adopted.

Stack machine. The final category was that of a stack architecture. Although such an architecture was ideal for high-level languages, it was difficult to program in assembly language. More importantly, such an architecture requires special hardware to make up for having the stack in core (e.g., the A and B registers on the Burroughs machines,[3] and without this hardware on the host, execution could be intolerably slow.

After evaluating the above architectures, the 1130-like architecture and the stack machine were ruled out for the reasons mentioned. At this point the decision became difficult, but the PDP-11-like architecture seemed to have a better blend of instruction power and size than the S/360-like architecture. The fact that it was perhaps overly complex seemed somewhat irrelevant since those people who were to do the initial assembly language programming were extremely experienced. For these reasons it was decided to go with the PDP-11-like architecture. The implementation of the PDP-11 architecture proceeded smoothly during the summer of 1971, until finally, when the bulk of the microprogramming was complete, timing measurements were made on the resulting emulator. It was discovered that a register/storage ADD instruction, requiring only three storage cycles (approximately three microseconds), took a total of 10 microseconds to execute. This time was considered to be completely inadequate.

In retrospect, the bottleneck became glaringly apparent. The instruction formats that had been adopted for this architecture consisted of many small fields (two or three bits) of information specifying such things as register numbers, addressing modes, and operation codes. These fields had to be isolated into various microregisters in order to fetch the target registers, branch on the operation codes, and to perform other necessary functions. Not enough attention had been paid to the META 4 host to realize that such isolation would require many control cycles since the only shifting that could be performed was right and left shifts of one or eight bits. If a 3-bit field must be isolated from bits 8 - 10 of a 16-bit word, for example, five shifts of "right one" must be performed, requiring about half a microsecond on the META 4. Performing such shifting many times in the course of a target instruction decreased the efficiency of the emulator drastically.

From the failure of this first design attempt came the knowledge that tailoring the target architecture to the host machine is of great importance and cannot be underestimated. As stated, the speed of the META 4A was not the most important factor, thus the slowness could perhaps be justified by arguing that programs would be significantly smaller with the PDP-11-like format than with the other architectures considered. To ascertain the validity of the justification, a set of benchmark programs was written using the PDP-11-like instruction set and a slightly modified S/360 set. Such programs as storage allocation routines, matrix inversion algorithms, and text processing functions indicated that not only did the S/360 instruction set outperform the PDP-11 by a speed factor of two to one, but that the PDP-11 programs were never more than 10 percent smaller than the others.

A SECOND ATTEMPT

Once the PDP-11-like architecture was abandoned, the only remaining possibility indicated by the investigations outlined above was an S/360-like architecture. The previous microprogramming of an S/360 emulator had been done in order to investigate the properties of the META 4 host, and this microprogramming indicated that S/360 instruction formats would be relatively free of complex shifting operation and hence faster to decode as compared to the PDP-11 formats. Indeed, when the second microprogramming task was finished, it was found that an equivalent ADD instruction took only 4.5 microseconds, as compared to the 10 for the PDP-11-like set; this was considered a satisfactory improvement, particularly in light of the small difference in program size.

The final implementation of the META 4A general-purpose processor, although S/360-like in nature, has many major departures from the actual S/360. In terms of architecture it is almost identical, except for the fact that the major numeric data type is the 16-bit (halfword) integer rather than the 32-bit (fullword) integer, due to the fact that the META 4 host has 16-bit registers. The two features omitted were the decimal data type, as this was considered unnecessary, and the floating-point data type. Floating-point is not included for two reasons. The first is that it is extremely difficult to implement in microcode without any hardware assistance; the resulting instructions would be extremely slow and consume a tremendous amount of precious control storage. The second reason is that any large amount of floating-point processing could be performed in the S/360-67 and the results transferred across the multiplexor channel to the META 4A. The META 4A has 16 target registers, implemented using 16 of the META 4 host's 32 registers, and instruction formats identical to those of the S/360. In terms of instruction set, however, it has many improvements over a S/360.

(1) The instruction address register, or Program Counter (PC) as it is called on the META 4A, is actually target register 1. This feature allows more complex branching techniques, such as can be obtained by performing an addition into the PC, or by loading an address from storage into the PC. Although this is a powerful facility, it does add to the inscrutability of the user's program logic. More importantly, as long as all local data is placed beyond any instructions which refer to it, the PC can be used as the program base register, thus freeing another precious general-purpose target register from this function.

(2). If the Pc is used as the program base register, it is impossible for an instruction to perform a backwards reference. This is no problem for data references, but branch instructions must be capable of diverting control to a previous instruction. For this reason, the format of the branching instructions has been changed from including a base-displacement address to including simply a *signed* displacement considered relative to the PC. Not only does this alleviate the backward branch problem, but it makes the decoding of branch instructions much faster, since a base-displacement address, requiring a register number isolation, fetch, and addition,

does not have to be performed. Branch instructions on the META 4A execute faster than those on an S/360-50!

(3) A new instruction format, called Register-Immediate (RI) format, has been added to the instruction repertoire. This format allows the programmer to perform the most common arithmetic and logical instructions using a register and an immediate halfword as the operands, thus saving both a base-displacement calculation and the halfword of storage that would be required for the remote constant. This proves to be a major factor in making most META 4A programs smaller than the equivalent S/360 programs. Additionally, the RI format instructions execute anywhere from one to two microseconds faster than the equivalent register/storage instructions.

(4) Instructions are provided which operate upon arbitrary length character strings. With these instructions the programmer can assign, compare, scan, translate, and initialize character string up to 64K bytes in length.

(5) SEARCH, ENQ, and DEQ instructions are provided for manipulating linked lists and tables. The SEARCH instruction can scan a table or a linked list for an arbitrary length key anywhere in its members which is a logical function of an argument key. If an entry satisfying the function is found, a register is set to point to it. Once a SEARCH is performed on a linked list with DEQ, the satisfying entry can be deleted from the list, or a new entry can be added following it with ENQ. These instructions have proven invaluable time and space savers for implementing queue searches in the BUGS operating system.[17] Such queues as the free storage queue, dispatch queue, and the interrupt exit queue, are searched and maintained by these three instructions. Unfortunately, they will not be generated by the LSD compiler, except perhaps via an explicit primitive.

(6) One interesting architectural experiment which was performed was to include a set of crude stack manipulation instructions, allowing the programmer to set up an arbitrary size stack and then push and pop information into and from it. The point of such an experiment was to learn whether or not programmers who were not experienced with a stack machine could learn to make use of such facilities, e.g., for expression evaluation. To date, no BUGS software has utilized the stack instructions.

(7) Two instructions, ENTER and RETURN, exist in order to simplify, and particularly to speed up, the subprocedure entry and exit protocol. Each procedure has associated with it a Stack Frame area, which contains a register save area and an arbitrary amount of automatic storage. These Stack Frame areas are maintained by the META 4A operating system. Experimentation with the format and content of these areas is continuing today.[17]

(8) Special facilities were included in the firmware to support the addition of Extended instructions. An Extended instruction is an instruction which has an operation code not recognized by the firmware, but which has special meaning to the operating system. When such an instruction is executed, the firmware stores the parsed instruction into a special area in storage, and causes a target-level interrupt. At this point the operating system can simulate any function desired and then return control to the program. Such a facility has proven invaluable for testing a new instruction before it is placed in the firmware, and for use as a communication method between user programs and the operating system software.

In addition to the major points listed above, a great number of miscellaneous instructions have been added to the standard S/360 repertoire in order to fill out what were considered "gaps" in the instruction set. This included such things as storage to storage arithmetic, more address manipulation features, and indirect addressing on certain instructions. Although these instructions are sometimes used and can help decrease the size of a program, they also tend to add to the difficulty of learning and digesting the instruction set. In particular, they increase the number of ways in which a problem can be coded and make it extremely difficult to select the most optimal algorithms. The question which remains, and which is currently being investigated, is just how much it is worth adding facilities which, although they may cut the size of a program by 5 percent, clutter up the instruction set and use up control storage. In addition, although these instructions were added to fill gaps in the S/360 repertoire, they have introduced their own gaps. An example is the addition of address manipulation instructions. Such an addition suggests that perhaps the address should be recognized as a valid data type, just as an integer is, and a full address manipulation instruction subset should be added. The META 4A does not have such a full subset, so that a new gap is introduced. Similarly, many a META 4A programmer has been heard to mutter such things as "if I can add two halfwords in storage, why can't I OR two halfwords?"

Another problem with the addition of "random" instructions is exemplified in an instruction by instruction analysis of the code to be generated by the LSD compiler. A full third of the META 4A instruction set is never utilized by the compiler; once the programming load is shifted over to LSD, these instructions will become virtually useless. The control storage taken up by them could probably be put to a much better use.

The BUGS system has been in standalone production use since August 1972, primarily for research into N-dimensional mathematics. The performance of the META 4A has proven to be rather impressive. Based on

the applications programs written so far, a typical META 4A program is between 3/4 and 2/3 the size of the equivalent S/360 program. Furthermore, the speed of execution of these programs (using halfword operands) lies somewhere in between the speed of an S/360-40 and an S/360-50, which is extremely pleasing considering the cost ratio between the META 4 host (about $30,000) and an S/360.

One interesting benchmark was a comparison of the IBM 1130 emulator supplied with the META 4 host (which is twice as fast as an actual 2.2 microsecond IBM 1130) and the META 4A emulator. A version of the META 4 simulator used for debugging microprograms[1] was written for both of these target machines and compared. The comparisons showed that the simulation speed of these two programs were indistinguishable. However, the META 4A version is 1/3 the size of the 1130 version, due primarily to the lack of character manipulation on the 1130.

In light of the above benchmarks and hand simulations of test algorithms written for the Data General NOVA and DEC PDP-11 series, it appears that arbitrary algorithms typically do not run appreciably slower and do use less storage on the META 4A, while algorithms that take advantage of the special purpose instruction on the META 4A both run considerably faster and use considerably less storage.

CONCLUSION—A RATIONAL APPROACH VIA FORMALIZATION

Although an initial framework around which a user can design and implement his own target machine has been built, there is nothing to prevent this design from being *ad hoc*. Indeed, in light of the many considerations which must be synthesized, an *ad hoc* design is inevitable. It has been shown that an architecture can be both an improvement over previous architectures and reasonably efficient while still being disorganized and incomplete. Such deficiencies cause the assembly language programmer many headaches in terms of choosing algorithms, optimizing code, and generally writing programs. Furthermore, if the choice of target instructions is disorganized, many instructions will be included which are minimally useful to the assembly language programmer and useless to the compiler designer, simply because a far-fetched use of the instruction was envisioned by one of the members of the design team. An interesting example of this is the LXB instruction on the META 4A, which reads a 16-bit halfword from storage and loads it into a register after exchanging the two bytes. This instruction was included for no other reason than the META 4 host machine had a byte swapping facility. It should be obvious that LXB has rarely if ever been used. All in all, the ad hoc architecture is due to a random combination of features useful to the application, features easily adaptable to the host, and features useful to the programmer.

How can such architectures be eliminated? A look at many of the high-level languages in use today, particularly PL/I, will reveal a general philosophy that deals with the problem. In PL/I, there is a well-defined set of data types and a well-defined set of operators, and any combination of these data types and operators is defined, except where meaningless. If such a formalization is adopted, it becomes easier to choose the statements necessary for each individual step of a program, and reduces the obscurity which results when "unnatural" statements are required. Furthermore, it reduces the tendency toward operators and data types which are added for special cases and hence do not fit into the overall language scheme. There is no reason why this same formalization cannot be applied at the lower level of target machine instruction sets. Since the average program is concerned chiefly with data manipulation, as opposed to I/O or interrupt handling, a formalization of the data manipulation facilities of the machine would have major impact.

The most apparent programming benefit of such formalization would be to the compiler designer. In order for him to allow the aforementioned generality to a programming language on most current machines, he must generate unnatural code which performs the operations indicated by the programmer. This code could be eliminated if the operations were made natural by allowing them to be performed directly by single instructions. The Burroughs family of stack machines has adopted just such a philosophy and the results are well known: ALGOL compilers which can generate efficient code at an incredibly high rate. In addition, the assembly language programmer gets an equivalent benefit in that he can code the individual steps of an algorithm in a more straightforward manner, without regard to such irrelevant considerations as whether an operand is in storage or in a register or whether a character string is longer than 256 characters. Programs coded in such an environment should be shorter, more free of errors, and easier to modify in the future.

The programming benefit gained from formalizing a proposed architecture is not the most important one, however. The purpose of this paper has been to outline a set of considerations which the target computer designer must keep in mind when creating a new machine. If the design is approached in a haphazard fashion, the designer will have perhaps 100 or 150 assorted features and instructions about which he must ask such questions as "are they useful to the application?", "how useable will they be by assembly and high-level language programmers?", and "will the host machine support them efficiently?" Such an overwhelming number of questions may be impossible to answer, particularly when the interrelationships between the operations is unclear. By formalizing this machine, the synthesis of these considerations can be made simpler and more productive. The designer need only answer these questions about perhaps twenty operations and five data types, a much smaller task. If these features are proven to be useful and efficient, then the designer can feel assured that the final

implementations of the instruction to perform the operations will be useful and efficient.

Current research at Brown is attempting to deal with the formalization question. An analysis of the use of the current META 4A instruction set is being performed via modifications to the firmware with the hope of determining instruction and instruction sequence characteristics and applying these characteristics to a determination of an optimal instruction set. A formalized architecture will then be designed, experimentally implemented, and an evaluation made of the improvement in program coding ease, speed, size, etc. Once this is done, a more detailed guide to computer design and implementation can be written.

REFERENCES

1. Anagnostopoulos, P. C., *META 4 Simulator Users' Guide*, Brown University, Center for Computer and Information Sciences Technical Report.
2. Anagnostopoulos, P. C., Sockut, G. H., *META 4A Principles of Operation*, Brown University, Center for Computer and Information Sciences Technical Report.
3. *B6500 Reference Manual*, Burroughs Corp., 1969.
4. Baker, F. T., "System Quality through Structured Programming," *AFIPS Conference Proceedings*, Volume 41, Part I, 1972.
5. Clapp, J. A., "The Application of Microprogramming Technology," *ACM SIGMICRO Newsletter*, April 1972.
6. *META 4 Computer System Reference Manual*, Digital Scientific Corporation, May 1971.
7. Dijkstra, E. W., *Notes on Structured Programming*, Technische Hogeschool, Eindhoven, 1969.
8. *PDP-11 Processor Handbook*, Digital Equipment Corp., 1971.
9. *System/360 Principles of Operation*, International Business Machines Corp., 1968.
10. *CP-67/CMS User's Guide*, International Business Machines Corp., 1971.
11. *Model 4 Micro-instruction Reference Manual*, Interdata Corp., 1968.
12. Kleir and Ramanoorthy, "Optimization Techniques for Microprograms," *IEEE Transactions on Computers*, July 1971.
13. *Mircroprogramming Handbook*, Microdata Corp., 1971.
14. Mandell, R. L., "Hardware/software trade-offs - Reasons and directions," *AFIPS Conference Proceedings*, Volume 41, Part I, 1972.
15. *QM-1 Hardware Level User's Manual*, Nanodata Corp., June 1972.
16. Rosin, R. F., "Contemporary Concepts of Microprogramming and Emulation," *ACM Computing Surveys*, Volume I, December 1969.
17. Stockenberg, J. E., Anagnostopoulos, P. C., Johnson, R. E., Munck, R. G., Stabler, G. M., van Dam, A., "Operating System Design Considerations for Microprogrammed Mini-computer Satellite Systems," *Proceedings of the National Computer Conference and Exposition*, June 1973.
18. Stabler, G. M., *Brown University, Graphics System Overview*, Brown University Center for Computer and Information Sciences Technical Report.
19. Stabler, G. M., *META 4B Principles of Operation*, Brown University, Center for Computer and Information Sciences Technical Report.
20. van Dam, A., "Microprogramming for Computer Graphics," ACM *SIGMICRO Newsletter*, April 1972.
21. van Dam, A., Schiller, W. L., Abraham, R. L., Fox, R. M., "A Microprogrammed Intelligent Graphics Terminal," *IEEE Transactions on Computers*, July 1971.
22. van Dam, A., Bergeron, D., Gannon, J., Shecter, D., Tompa, F., "Systems Programming Language," *Advances in Computer*, Volume 12, Academic Press, Oct. 1972.
23. Wilner, W. T., "Design of the Burroughs B1700," *AFIPS Conference Proceedings*, Volume 41, Part I, December 1972.
24. Young, S., "A Microprogram Simulator," *ACM SIGMICRO Newsletter*, October 1971.

A new minicomputer/multiprocessor for the ARPA network*

by F. E. HEART, S. M. ORNSTEIN, W. R. CROWTHER, and W. B. BARKER

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

Since the early years of the digital computer era, there has been a continuing attempt to gain processing power by organizing hardware processors so as to achieve some form of parallel operation.[1,2] One important thread has been the use of an array of processors to allow a single control stream to operate simultaneously on a multiplicity of data streams; the most ambitious effort in this direction has been the ILLIAC IV project.[3,4] Another important thread has been the partitioning of problems so that several control streams can operate in parallel. Often functions have been unloaded from a central processor onto various specialized processors; examples include data channels, display processors, front-end communication processors, on-line data preprocessors—in fact, I/O processors of all sorts. Similarly, dual processor systems have been used to provide load sharing and increased reliability. Still another thread has been the construction of pipeline systems in which sub-pieces of a single (generally large) processor work in parallel on successive phases of a problem.[5] In some of these pipeline approaches the parallelism is "hidden" and the user considers only a single control stream.

In recent years, as minicomputers have proliferated, groups of identical small machines have been connected together and jobs partitioned quite grossly among them. Most recently, our group and several others have been investigating this avenue further, attempting to reduce the specialization of the processors in order to employ independent processors with independent control streams in a cooperative and "equal" fashion.[6,7,8]

This paper describes a new minicomputer/multiprocessor architecture for which a fourteen-processor prototype is now (February 1973) being constructed. The hardware design and the software organization include many novel features, and the system may offer significant advantages in modularity and cost/performance. The system contains an expandable number of identical processors, each with some "private" memory; an expandable amount of "shared" memory to which all processors have equal access; and an expandable amount of I/O interface equipment, controllable by any processor. The system achieves unusual modularity and reliability by making all processors equivalent, so that any processor may perform any system task; thus systems can be easily configured to meet the throughput requirements of a particular job. The scheme for interconnecting processors, memories, and I/O is also modular, permitting interconnection cost to vary smoothly with system size. There is no "executive" and each processor determines its own task allocation.

A key issue throughout most of the attempts at parallel organization has been the difficulty of partitioning problems in such a way that the resulting computer program(s) can really take advantage of the parallel organization. This issue is raised in its most serious form when the parallel machine is expected to work well on a great diversity of problems as, for example, in a time-sharing system. Our machine design has been developed under the highly favorable circumstances that (1) the initial application, and a prior software implementation in a standard machine, was well understood; (2) the initial application lent itself to fragmentation into parallel structures; and (3) the design would be deemed successful if it handled only that one application in a meritorious fashion. However, we now believe that the design is advantageous for many other important applications as well and that it may herald a broadly useful new way to achieve increased performance and reliability.

The machine has been designed to serve initially as a modular switching node for the ARPA Network[9] and, in the following section, we briefly describe the ARPA Network application and the requirements that the network imposed upon the machine design. In subsequent sections we discuss our choice of minicomputer, describe our system design in some detail, discuss certain of the more interesting characteristics of multiprocessor behavior, and summarize our present status and plans for the near future.

* *This work was sponsored by the Advanced Research Projects Agency under Contracts DAHC15-69-C-0179 and FO8606-73-6-0027.*

ARPA NETWORK REQUIREMENTS

The ARPA Network, a nationwide interconnection of computers and high bandwidth (50 Kb) communication circuits, has grown during the past four years to include over 35 sites, with more than one computer at many sites. The computers at each site, called Hosts, obtain access to the net via a small communications processor known as an Interface Message Processor or IMP.[10] In order to permit groups without their own computer facility to access this powerful set of computer resources, a version of the IMP called a Terminal IMP allows, in addition, attachment of up to 63 local or remote terminals of a wide range of types.[11]

As a considerable simplification, the job to be handled by an IMP is that of a communications processor. Arriving messages must pass through an error control algorithm, be inspected to some degree (e.g., for destination), and generally be directed out onto some other line. Some incoming messages (e.g., routing control messages) must be constructed or digested directly by the IMP. The IMP must also concern itself with flow control, message assembly and sequencing, performance and flow monitoring, Host status, line and interface testing, and many other housekeeping functions. To perform these functions an IMP requires memory both for program and for message buffers, processing power for executing the program, and I/O units of various sorts for connecting to a variety of lines and devices. The original IMP, built around a Honeywell 516 processor with a 1 μs cycle time, could handle approximately three-quarters of a megabit per second of full duplex communications traffic. A later, smaller and cheaper (Honeywell 316) version handles about two-thirds as much traffic.

As the network has grown and as usage has increased, a number of demands for improvement have led to the need for a new "line" of IMP machines. Our intent is to provide a modular arrangement of flexible hardware from which it will be possible to construct both smaller and less expensive IMPs as well as far more powerful IMPs. An important specific objective is to obtain an IMP whose communications bandwidth could be at least an order of magnitude greater than the 516 IMP; such a high speed IMP would permit the direct connection of satellite circuits or land T-carrier circuits operating at approximately 1.3 megabits/second.

It is also desirable to improve the present IMP design in a number of other areas, as follows.

- Expandability of I/O: The present IMPs permit connection to a total of only seven high-speed circuits and/or Host computers. We would like to permit a much greater fanout so that an IMP might be connected to as many as 20 or more Host computers or to hundreds of terminals. This means that the number of interface units should be expandable over a wide range.
- Modularity: A number of groups have wished to make a network connection from a single Host at a considerable distance (miles) from the nearest IMP. We feel that such Hosts should be locally connected to a very small IMP in order to preserve consistency and standardization throughout the network. Therefore, a goal of this new hardware effort is the provision of a small and inexpensive but compatible IMP which could serve to connect a single, distant spur Host.
- Expandability of Memory: The new line of equipment is required for use in connection with satellite links (or longer faster links in general) and must therefore be able to expand its memory easily to provide the much greater buffer storage requirements of such links.
- Reliability: The new line of processors should be more reliable than the existing IMPs and ought to permit better self-diagnosis and simple isolation and replacement of failing units.

Of the requirements posed by the ARPA Network application, the most central was to obtain an order-of-magnitude traffic bandwidth improvement. We first considered meeting this requirement with highly specialized hardware, but the need to allow evolution of the communications algorithms, as well as the "bookkeeping" nature of much of the IMP task, militate against hardwired approaches and require the flexibility of a stored program computer. Thus we need a machine with an effective cycle time of 100 nanoseconds, a factor of ten faster than the present 1 μs IMP. Realizing that a single very fast and powerful machine would be difficult to build and would not give us compatible machines with a wide spectrum of performance, we began to consider the possibility of a minicomputer/multiprocessor in order to achieve the flexibility, reliability, and effective bandwidth required.

With the idea of a multiprocessor in mind we considered the IMP algorithm to determine which parts were inherently serial in nature and which could proceed in parallel. It seemed difficult to process a single message in a parallel fashion: the job was already relatively short and intimately coupled to I/O interfaces. However, there was much less serial coupling between the processing of separate messages from the same phone line and no coupling at all between messages from different phone lines. We thus envisage many processors, each at work on a separate message, with the number of processors carefully matched to the number of messages we expect to encounter in the time it takes one processor to deal with one message. With this simple image there seems to be no inherent limit to the parallelism we can achieve—the ultimate limit would be set by the size of the multiprocessor we can build.

CHOICE OF THE PROCESSOR

In designing a multiprocessor for the IMP application, we found ourselves iteratively exploring two related but distinct issues. First, assuming that the problem of interconnection could be solved, what minicomputer would be

a sensible choice from the price/performance and physical points of view? Second, and much harder: for any specific machine, how did the CPU talk to memory, how would multiple CPUs, memories, and I/O be interconnected to form a system, and how would the program be organized?

Since the program for the existing IMPs was well understood, it was possible to identify key sections of that program which consumed the majority of the processing bandwidth. Then, for each sensible minicomputer choice, we could ask how many CPUs of this type would be needed to provide an effective 100 nanosecond cycle time; and given a price list, physical data, and a modest amount of design effort, we could define the physical structure and the price of the resulting multiprocessor. With this general approach, we examined the internal design of about a dozen machines, and actually wrote the key code in many cases. Using the fastest available minicomputers it was possible to arrive at configurations with only three or four processors; using the slowest choices, systems with 20 CPUs or more were required.

If we defer the interconnection and contention problems for a moment, it is interesting to note that "slow and cheap" may win over "fast and expensive" in this kind of multiprocessor competition to achieve a stated processing bandwidth. This is an especially happy situation if, as in our case, a spectrum of configurations is needed, including a very tiny cheap version.

In considering which minicomputer might be most easily adaptable to a multiprocessor structure, the internal communication between the processor and its memory was of primary concern. Several years ago machines were introduced which combined memory and I/O busses into a single bus. As part of this step, registers within the devices (pointers, status and control registers, and the like) were made to look like memory cells so that they and the memory could be referenced in a homogeneous manner. This structure forms a very clean and attractive architecture in which any unit can bid to become master of the bus in order to communicate with any other desired unit. One of the important features of this structure is that it made memory accessing "public"; the interface to the memory had to become asynchronous, cleanly isolable electrically and mechanically, and well documented and stable. A characteristic of this architecture is that all references between units are time multiplexed onto a single bus. Conflicts for bus usage therefore establish an ultimate upper bound on overall performance, and attempts to speed up the bus eventually run into serious problems in arbitration.[12]

In 1972 a new processor—the Lockheed SUE[13]—was introduced which follows the single bus philosophy but carries it an important step further by removing the bus arbitration logic to a module separate from the processor. This step permits one to consider configurations embodying multiple processors and multiple memories as well as I/O on a single bus. The SUE CPU is a compact, relatively inexpensive (approximately $600 in quantity), quite slow processor with a microcoded inner structure. This slowness can be compensated for by simply doubling or trebling the number of processors on the bus; performance is limited largely by the speed of the bus. With this bus architecture it becomes attractive to visualize multibus systems with a "bus coupling" mechanism to allow devices on one bus to access devices on other busses.

Similar approaches can be implemented with varying degrees of difficulty in systems with other bus structures, and we examined several approaches in some detail for those processors whose cost/performance was attractive. Rather fortuitously, the minicomputer which exhibited the most attractive bus architecture also was extremely attractive in terms of cost/performance and physical characteristics. This machine, the Lockheed SUE, would require fourteen processors to achieve the effective 100 nanosecond cycle time, and we embarked on the detailed design of our multiprocessor on that basis.

SYSTEM DESIGN

Although our design permits systems of widely varying size and performance, in the interest of clarity we will describe that design in terms of the particular prototype now under construction. Our overall design is represented in Figure 1. We require fourteen SUE processors to obtain the necessary processing bandwidth, and we estimate that 32K words of memory will be required for a complete copy of the operational program and the necessary communication buffer storage. The I/O arrangements must allow easy connection of all the communications interfaces, appropriate to the IMP job (modem interfaces, Host interfaces, terminal interfaces) as well as standard peripherals and any special devices appropriate to the multiprocessor nature of the system.

Some of the basic SUE characteristics are listed in Table I. From a physical point of view, the SUE chassis represents the basic construction unit; it incorporates a printed circuit back plane which forms the bus into which 24 cards may be plugged. From a logical point of view this bus simply provides a common connection between all

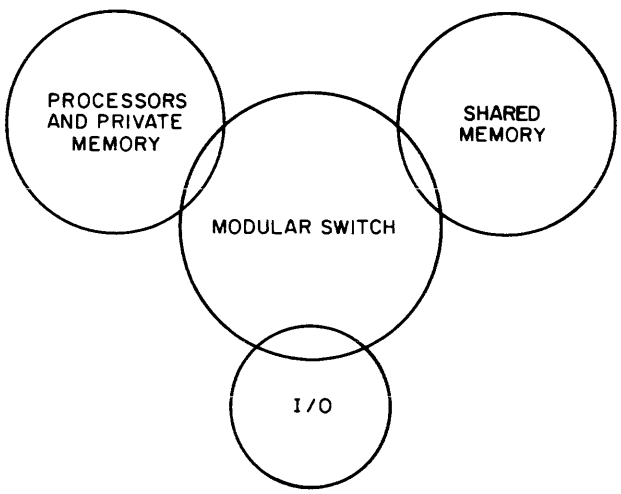


Figure 1—System structure

TABLE I—SUE Characteristics

16-bit word
8 General Registers
△3.7 μs add or load time
Microcoded
Two words/instruction typical
8-½"×19"×18" chassis
64K bytes addressable by a single instruction
~$3K for: 1 CPU+4K Memory+Power, Rack, etc.
200 ns minimum bus cycle time
850 ns memory cycle time
425 ns memory access time

units plugged into the chassis. We are using these chassis for the entire system: processor, memory, and I/O. All specially designed cards as well as all Lockheed-provided modules plug into these bus chassis. With this hardware, the terms "bus" and "chassis" are used somewhat interchangeably, but we will commonly call this standard building unit a "bus." Each bus requires one card which performs arbitration. A bus can be logically extended (via a bus extender unit) to a second bus if additional card space is required; in such a case, a single bus arbiter controls access to the entire extended bus.

We can build a small multiprocessor just by plugging several processors and memories (and I/O) into a single bus. For larger systems we quickly exceed the bandwidth capability of a single bus and we are forced to multi-bus architecture. Then, from a construction viewpoint, our multiprocessor design involves assigning processors, memories and I/O units to busses in a sensible manner and designing a switching arrangement to permit interconnection of all the busses. Of course, the superficial simplicity of this construction viewpoint completely hides the many difficult problems of multiprocessor system design; we will try to deal with some of those issues in the following sections.

Resources

A central notion in a parallel system is the idea of a "resource," which we define to mean a part of the system needed by more than one of the parallel users and therefore a possible source of contention. The three basic hardware resources are the memories, the I/O, and the processors. It is useful to consider the memories, furthermore, as a collection of resources of quite different character: a program, queues and variables of a global nature, local variables, and large areas of buffer storage.

The basic idea of a multiprocessor is to provide multiple copies of the vital resources in the hope that the algorithm can run faster by using them in parallel. The number of copies of the resource which are required to allow concurrent operation is determined by the speed of the resource and the frequency with which it is used. An additional advantage of multiple copies is reliability: if a system contains a few spare copies of all resources, it can continue to operate when one copy breaks.

It may seem peculiar to think of a processor as a resource, but in fact in our system the parallel parts of the algorithm compete with each other for a processor on which to run. We take the view that all processors shall be identical and equal, and we go to some trouble to insure that this is in fact so. As a consequence no single processor is of vital importance, and we can change the number of processors at will. A later section will describe how the processors coordinate to get the job done without a master of some sort.

Processor busses

A SUE bus can physically and logically support up to four processors. As more processors are added to a bus, the contention for the bus increases, and the performance increment per processor drops; but the effective cost per processor also drops, since the cost for the chassis, power supply, bus arbitration, etc., is amortized over the number of processors.

Roughly speaking, using two processors per bus loses almost nothing in processor performance, using three processors per bus loses significant efficiency, and adding a fourth processor gains less than half an "effective processor." After careful examination of the logical, economic and physical aspects of this choice, we decided to use two processors per processor bus, and we thus require seven processor busses in our initial multiprocessor system.

The next question was how the processors should access the program. In our application, some parts of the program are run very frequently and other parts are run far less frequently. This fact allows a significant advantage to be gained by the use of private memory. When a processor makes access to shared memory via the switching arrangement, that access will incur delays due to contention and delays introduced by the intervening switch. We therefore decided to use a 4K local memory with each processor on its bus to allow faster local access to the frequently run code; these local memories all typically contain the same code. With this configuration and in our application, the ratio of accesses to local versus shared memory is better than three to one. This not only reduces contention delays for access to the shared memory but also cuts the number of accesses which suffer the delays.

The final configuration of a processor bus is shown in Figure 2(a). The units marked "Bus Coupler" have to do with our multiprocessor switching arrangement, which will be discussed below.

*Shared memory busses**

The shared memory of our multiprocessor is intended to contain a copy of the program as well as considerable storage space for message buffering, global variables, etc. Application-dependent considerations led us to select a

* *The terms "I/O bus" and "memory bus" as used here and henceforth are not the same as conventional I/O and memory busses.*

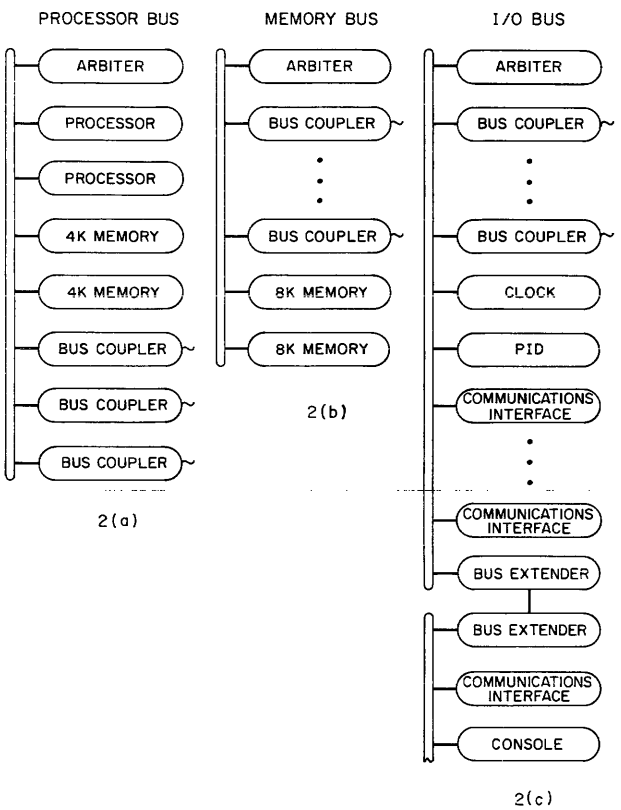


Figure 2—Bus structures

32K memory, but it is possible to configure this memory on a single bus or to divide the memory onto several busses. We first concluded that four logical memory units would be appropriate in order to reduce processor contention to an acceptable level. Then, since the bus is considerably faster than the memories, it is feasible to place two logical memory elements on a single bus with almost no interference. Thus, we are planning two memory busses in the initial multiprocessor; the configuration of a common memory bus is shown in Figure 2(b).

I/O busses

The I/O system of the multiprocessor employs standard SUE busses with standard bus arbitration units on those busses. Into the bus will be plugged cards for each of the various types of I/O interfaces that are required, including interfaces for modems, terminals, Host computers, etc., as well as interfaces for standard peripherals. Our initial system has a single I/O bus and Figure 2(c) shows its configuration; the specialized units shown (a "Clock" and "Pseudo Interrupt Device") are system-wide resources that are used to control the operation of the multiprocessor. The I/O bus will also be the access route for the multiprocessor console; we plan to use a standard alphanumeric display terminal which can be driven by code in any processor, and no conventional consoles will be used.

Interconnection system

Our prototype multiprocessor is now seen to contain seven processor busses, two shared memory busses and an I/O bus. To adhere to our requirement that all processors must be equal and able to perform any system task, these busses must be connected so that all processors can access all shared memory, so that I/O can be fed to and from shared memory, and so that any of the processors may control the operation and sense the status of any I/O unit.

A *distributed* inter-communication scheme was chosen in the interest of expandability, reliability, and design simplicity. The atom of this scheme is called a Bus Coupler, and consists of two cards and an interconnecting cable. In making connections between processors and shared memory, one card plugs into a shared memory bus, where it will request cycles of the memory; the other card plugs into a processor's bus, where it looks like memory. When the processor requests a cycle within the address range which the Bus Coupler recognizes, a request is sent down the cable to the memory end, which then starts contending for the shared memory bus. When selected, it requests the desired cycle of the shared memory. The memory returns the desired information to the Bus Coupler, which then provides it to the requesting processor, which, except for an additional delay, does not know that the memory was not on its own bus. Note that the memory access arbitration inherent in any memory switching arrangement is handled by the SUE Bus Arbiter controlling the shared memory bus, while the Bus Coupler itself is conceptually straightforward.

One additional feature of the Bus Coupler is that it does address mapping. Since a processor can address only 64K bytes (16 bit address), and since we wished to permit multiprocessor configurations with up to 1024K bytes (20 bit address) of shared memory, a mechanism for address expansion is required. The Bus Coupler provides four independent 8K byte windows into shared memory. The processor can load registers in the Bus Coupler which provide the high-order bits of the shared memory address for each of the four windows.

Given a Bus Coupler connecting each processor bus to each shared-memory bus, all processors can access all shared memory. I/O devices which do direct memory transfers must also access these shared memories. These I/O devices are plugged into as many I/O busses as are required to handle the bandwidth involved, and bus couplers then connect each I/O bus to each memory bus. Similarly, I/O devices also need to respond to processor requests for action or information; in this regard, the I/O devices act like memories and Bus Couplers are again used to connect each processor bus to each I/O bus. The path between processor busses and I/O busses is also used in a more sophisticated fashion to allow processors to examine and control other processors; this subject is described in a later section.

The resulting system is shown in Figure 3. One is struck by the number of bus couplers: P*I+I*M+P*M bus couplers are required for a system with P processor bus-

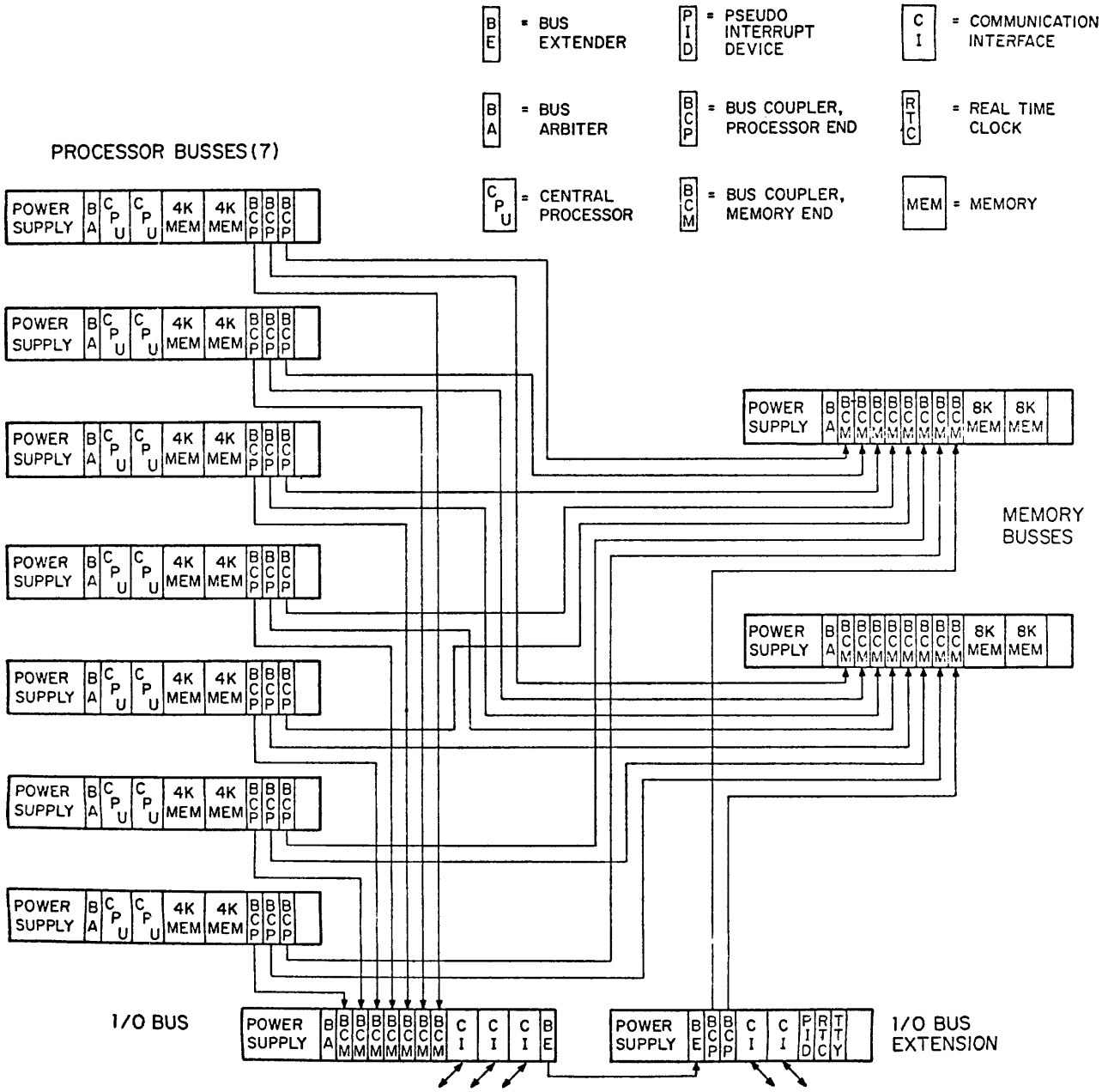


Figure 3—Prototype system

ses, I I/O busses, and M memory busses. In the case of our initial multiprocessor, 23 are needed.

This modular interconnection approach clearly permits great flexibility in the number and configuration of busses, and allows interconnection cost to vary smoothly with system size. We believe that this modular interconnection scheme also permits a complex hierarchical arrangement of busses. Actually the system exhibits a pronounced hierarchical structure already. A processor accesses the local memory when it needs instructions or local variables. Two such processor-memory combinations form a dual processor, which can be regarded as a unit and which needs access to shared resources, such as global variables, free buffers, and I/O interfaces. When one copy of a resource can only support a limited number of users, it seems sensible to provide only the corresponding limited number of connections. If a multiprocessor of this type were to grow larger, the physical number of bus couplers as well as increasing contention problems might not permit the connection of each processor to all of common memory, but might instead require a multi-level structure where groups of processors were connected to an

intermediate level bus which was in turn connected to a centralized common memory. We have not explored this domain but feel it is an interesting area for future work.

MULTIPROCESSOR BEHAVIOR

Until the processors interact, a multiprocessor is a number of independent single processor systems: it is the interaction which poses the conceptual as well as the practical problems. If the various processors spend their time waiting for each other, the system degrades to a single processor equivalent; if they can usefully run concurrently, the processing power is multiplied by the number of processors. If the failure of a single processor takes the system down, the system reliability is only the probability of all processors being up; if working processors can diagnose and heal or amputate faulty processors and proceed with the job, the system reliability approaches the probability of *any* processor being up. We now consider how to keep processors running concurrently, and then how to keep the system running in the case of module failure.

The first problem in making the machines run independently is the allocation of runnable tasks to processors, so that the full requisite power can be quickly brought to bear on high priority tasks. Our scheme for doing this rests on four key ideas: (1) We break the job up into a set of tiny tasks. (2) Our processors are all identical, asynchronous, and capable of doing any task. (3) We keep a queue of pending tasks, ordered by priority, from which each processor at its convenience gets its next task. (4) For speed and efficiency, we use a hardware device to help manage the queue.

By breaking the job up into smaller and smaller tasks until each one runs in under 300 μs, we effectively determine the responsiveness of our system. Once started, a task must run to completion, but there will be a reconsideration of priorities at the beginning of each new task. We have chosen 300 microseconds as the maximum task execution time because this compromise between efficiency and responsiveness is well matched to the execution time of key IMP functions.

By making the processors identical, we can use the same program in systems of widely varying size and throughput capability. Any processor can be added to or removed from a running system with only a slight change in throughput. The power of all processors quickly shifts to that part of the algorithm where it is most needed.

By queuing pending tasks, we keep track of what must be done while focusing on the most important tasks. By using a passive queue in which the processors check for a new task when they are ready, we avoid some nasty timing problems. Tasks may be entered into the queue at any time, either by a processor or by the hardware I/O devices. This approach is an extremely important departure which avoids the use of conventional interrupts and the associated costs of saving and restoring machine state. Further, this approach neatly sidesteps the problem of routing interrupts to the proper processor.

We could not afford a software queue both because it was slow to use and because processors would have been waiting for each other to get access to the queue. Instead we use a special hardware device called a Pseudo Interrupt Device (PID), which keeps in hardware a list of what to do next. A number can be written to the PID at any time and and it will be remembered. When read, the PID returns (and deletes) the highest number it has stored. By coding the numbers to represent tasks, and keeping the parameters of the tasks in memory, a processor can access the PID at the end of each task and determine very rapidly what it should do next.

Contention

Clearly, the PID must give any task to exactly one processor. This is guaranteed because the PID is on a bus that can be accessed by only one processor at a time and because the PID completes each transaction in a single access. This is an example of the more general problem that whenever two users want access to a single resource there must be an interlock to let them take turns. This is true at many levels, from contention for a bus to processor contention for shared software resources such as a free list. When all the appropriate interlocks have been provided, the performance of the multiprocessor will depend rather critically on the time wasted waiting at these interlocks for a resource to become free. As discussed above, whenever conflicts become a serious problem one provides another copy of the resource. We studied our system behavior carefully, noting areas of conflict, in order to know how many additional copies of heavily accessed resources to provide. Table II provides examples of delays due to various conflicts. Practically speaking, the curve of delay vs. number of resources has a rather sharp knee, so that it is meaningful to make such statements as "a memory bus supports eight processors" or "a free list supports eight processors." Of course, these statements are application related and depend on the frequency and duration of accesses required.

With interlocks, deadlocks become possible (in both hardware and software). For example, a deadlock occurs

TABLE II—Expected System Slowdown Due to Contention Delays

Slowdown	Cause
5.5%	Contention for a Processor Bus.
3%	Contention for the Shared Memory Busses.
5%	Contention for the Shared Memories.
10%	Contention for a single system-wide software resource, assuming each processor wants the resource for 6 instructions out of every 120 instructions executed.
1.7%	Contention for one of two copies of a system-wide software resource, as above.
0.15%	Contention for the parameters of a single 1.3 megabit phone line, assuming the parameters will be used for 160 microseconds every 800 microseconds.

when each of two processors has claimed one of two resources needed by both. Each waits indefinitely for the other's resource to become available.[14] Unless there is a careful systematic approach to interlocks, deadlocks interlock, and require that a processor never compete for a resource when it already owns a higher numbered resource. It is not always practical or possible to do this, although we expect to be able to do so with the IMP algorithms.

An interesting example of a deadlock occurs in our bus coupling. To permit processors to access one another, for mutual turn on, turn off, testing, etc., the path connecting each processor bus with the I/O bus is made bi-directional. Thus processors access one another via the I/O bus. In a bi-directional coupler, a deadlock arises when units obtain control of their busses at each end and then request access via the coupler to the bus on the other end. Because the backward path is infrequently used, we simply detect such deadlocks, abort the backward request and try again.

Reliability

We have taken a rather ambitious stand on reliability. We plan to detect a failing module automatically, amputate it, and keep the system running without human intervention if at all possible. Critical to our approach is the fact that there are several processors each with private memory and thus each able to retreat to local operation in the face of system problems. To reduce our vulnerability further, power and cooling are provided on a modular basis so that loss of a single unit does not jeopardize system operation. We are only mildly concerned with the damage done at the time of a failure, because the IMP system includes many checks and recovery procedures throughout the network.

The first sign of a failure may be a single bit wrong somewhere in shared memory, with all units apparently functioning properly. Alternatively, the failure may strike catastrophically, with shared memory in shambles and the processors running protectively in their local memories. Against this spectrum we cannot hope for a systematic defense; instead we have chosen a few defensive strategies.

So long as a module is failing, recovery is meaningless. We must run diagnostics to identify the bad module, or see if cutting a module out at random helps things. We feel that identifying such a solid failure will be relatively easy. Since a processor without couplers is completely harmless, once we identify a malfunctioning processor, we amputate it by turning off its bus couplers. We considered the possibility of a runaway processor turning good processors off. This is unlikely to begin with but we decided to make it even less likely by requiring a particular 16-bit password to be used in turning off a coupler. A runaway processor storing throughout shared memory would need this password in its accumulator to accidentally amputate. Similarly we require a password for one processor to get at another's local memory.

Against intermittents we use a strategy of dynamic reinitialization. Every data structure is periodically checked; every waiting state is timed out; the code is periodically checksummed; memory transfers are hardware parity checked; memory is periodically tested; processors are periodically given standard tests. Whenever anything is found wrong, the offending structure is initialized. Using this scheme we may not know what caused a failure, but its effects will not persist. In the most extreme cases we will need to reload all the program in main memory. Fortunately we have a communications network handy to load from. This technique of reloading has worked remarkably well in the current ARPA Network. Each processor has a copy of the reload program in its local memory, thus making loss of reload capability unlikely.

We might seem to be vulnerable to memory or I/O failures, particularly those involving the PID and the clock. If these modules fail it does indeed hurt us more, but only because we have fewer modules of these types in our system. If we provide redundant modules, the system can reconfigure itself to substitute a spare module for a failed one. Our design allows multiple I/O busses with multiple PIDs and clocks, and we could even have separate backup interfaces to vital communication lines on separate busses.

To summarize, the mainstay of our reliability scheme is a system continually aware of the state of things and quickly responding to unpleasant changes. The second line of defense consists of drastic actions like amputation and reloading. Assuming we can make all this work, we will have quite a reliable system, perhaps even one in which maintenance consists of periodic replacement of those parts which the system itself has rejected.

STATUS AND NEAR FUTURE

In February 1973, as this paper is submitted, we are very much in the middle of our multiprocessor development. Much progress has been made and we are increasingly confident of the design, but much work remains to be done.

The broad design is complete; all Lockheed-provided units (CPUs, memories, busses, etc.) have been delivered; prototype wire-wrapped versions of the crucial special modules have been completed, including the Bus Couplers, Pseudo Interrupt Device, clock, and modem interfaces; and a multi-bus, multi-processor-per-bus assembly has been successfully tried with a test program. A substantial program design effort has been in progress and coding of the first operational program has been started. We are still doing detailed design of some hardware, and we are still learning about detailed organizational issues as the software effort proceeds. An example of such an

area is: exactly how is it best for processors to watch each other for signs of failure?

We currently anticipate the parts cost of the prototype fourteen-processor system, without communication interfaces, to be under $100K.

Hopefully, by the time this paper is presented in June 1973, we will be able to report an operational prototype multiprocessor system. Beyond that, our schedule calls for the installation of a machine in the ARPA Network by about the end of 1973. We also plan to construct many variant systems out of this kit of building blocks, and to experiment with systems of varying sizes. As part of this work, we plan to concentrate on the very smallest version that may be sensible, in order to provide a minimum cost IMP for spur applications in the ARPA Network.

As the design has proceeded, our attraction to the general approach has increased (perhaps a common malady), and we now believe that the approach is applicable to many other classes of problems. We expect to explore such other applications as time permits, with initial attention to two areas: (1) certain specialized multi-user systems, and (2) high bandwidth signal processing.

With our presently planned building blocks, although we do not yet know what will limit system size, we do not now see any intrinsic problem in constructing systems with fifty or a hundred processors. As improvements in integrated circuit technology occur, and processors and memories become smaller and cheaper, organization and connection become the paramount questions in multiprocessor design. We expect to see many attempts at multiprocessors, and are hopeful that the ideas embodied in this design will help to steer that technology. Perhaps minicomputer/multiprocessors will soon represent real competition for the various brontosaurus machines that now abound.

ACKNOWLEDGMENTS

Our new machine design is a product of many minds. We gratefully acknowledge the specific design contributions of M. Kraley, A. Michel, M. Thrope, and R. Bressler. Helpful criticism and an important idea about the Pseudo Interrupt Device were contributed by D. Walden. Assistance in planning and in the choice of building blocks was contributed by H. Rising. Helpful ideas and criticism were provided by J. McQuillan, B. Cosell, and A. McKenzie. Assistance with support software was provided by J. Levin.

We also wish to express appreciation for the support and encouragement provided by Dr. L. Roberts of the Advanced Research Projects Agency.

REFERENCES

1. Lehman, M., "A Survey of Problems and Preliminary Results Concerning Parallel Processing and Parallel Processors", *Proc. IEEE*, Vol. 54, No. 12, pp. 1889-1901, December, 1966.
2. Lorin, H., *Parallelism in Hardware & Software - Real and Apparent Concurrency* Prentice-Hall, 1971.
3. Slotnick, J. L., Bork, W. C., McReynolds, R. C., "Solomon", *AFIPS Conference Proceedings*, FJCC 1962.
4. Barnes, G. H., et al, "The Illiac IV Computer", *IEEE Trans.* C-17, Vol. 8, pp. 746-757, August 1968.
5. Anderson, D. W., Sparacio, F. J., Tomasulo, R. M., "The IBM System/360 Model 91 - Machine Philosophy and Instruction Handling", *IBM Journal* No. 11, January 1967, pp. 8-24.
6. Cohen, E., "Symmetric Multi-Mini-Processors, A Better Way to Go?" *Computer Decisions*, January 1973.
7. Wulf, W. A., Bell, C. G., "C.mmp - A Multi-Mini Processor", *AFIPS Proceedings*, FJCC, Vol. 41, 1972.
8. Cosserat, D. C., "A Capability Oriented Multi-Processor System for Real-Time Applications", *Computer Communication Proc. ICCC*, pp. 282-289, October 1972.
9. Roberts, L. G., Wessler, B. D., "Computer Network Development to Achieve Resource Sharing" *AFIPS Proceedings*, SJCC, Vol. 36, 1970.
10. Heart, F. E., et al, "The Interface Message Processor for the ARPA Computer Network", *AFIPS Proceedings*, SJCC, Vol. 36, 1970.
11. Ornstein, S. M., et al., "The Terminal IMP for the ARPA Computer Network", *AFIPS Proceedings*, SJCC, Vol. 40, 1972.
12. Chaney, T., Ornstein, S., Littlefield, W., "Beware the Synchronizer", *Proc. COMPCON Conference*, 1972.
13. *SUE Computer Handbook*, Lockheed Electronics Company, Los Angeles, 1972.
14. Holt, R. C., "Some Deadlock Properties of Computer Systems", *ACM Computing Surveys*, Vol. 4, No. 3, pp. 179-196, September 1972.

SUPPLEMENTARY BIBLIOGRAPHY

Amdahl, G. M., *Engineering Aspects of Large High-Speed Computer Design - Part II Logical Organization*, IBM Tech. Report TR00.1227, December 1964.

Baskin, H. B., et al, "A Modular Computer Sharing System," *CACM*, Vol. 12, No. 10, October 1969, p. 551.

Bell and Newell, *Computer Structures*, McGraw-Hill, 1971.

Bell, G., et al, *C.mmp the CMU Multiminiprocessor Computer*, Dept. of Computer Science, Carnegie Mellon Univ., August 1971.

Burnett, G. J., et al., "A Distributed PROCESSING System for General Purpose Computing", *AFIPS Proceedings*, FJCC, Vol. 31, 1967.

Dijkstra, E. W., "Cooperating Sequential Processes", in *Programming Languages*, (Gennys, F., ed.), Academic Press, pp. 43-110, 1968.

Flynn, M. J., "Some Computer Organizations and Their Effectiveness", *IEEE Transactions on Computers*", Vol. C-21, No. 9, September 1972.

Flynn, M. J., "Very High-Speed Computing Systems", *Proc. IEEE*, Vol. 54, No. 12, pp. 1901-1909, December, 1966.

Holland, J. H., "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously," *AFIPS Proceedings*, FJCC, pp. 108-113, 1959.

McQuillan, J. M., et al, "Improvements in the Design and Performance of the ARPA Network", *AFIPS Proceedings*, FJCC, Vol. 41, 1972.

Ornstein, S. M., Stucki, M. J., Clark, W. A., "A Functional Description of Macromodules" *AFIPS Proceedings*, SJCC, Vol. 30, 1967.

Pirtle, M., "Intercommunication of Processors & Memory", *AFIPS Proceedings*, FJCC, Vol. 31, 1967.

Randell, B., "Operating Systems - The Problems of Performance and Reliability", *IFIP Congress 71*, Ljubljana, North Holland Pub. Co., 1972, pp. 281-290.

A Description of the Advanced Scientific Computer System, Texas Instruments, Inc., 1972.

Thornton, J. E., "Parallel Operation in the Control Data 6600", *AFIPS Proceedings*, FJCC, Vol. 26, 1964.

Wulf, W., et al, *Hydra— A Kernel Operating System for C.mmp*, Dept. of Computer Science, Carnegie Mellon Univ., 1971.

System integrity in small real-time computer systems

by THOMAS J. HARRISON and THOMAS J. PIERCE

IBM Corporation
Boca Raton, Florida

INTRODUCTION

Reliability has been a long-standing concern to the user of small real-time computer systems. Traditionally, reliability has been measured by parameters such as mean-time-between-failure (MTBF) and is usually associated with the failure of a hardware component in the system. But there are many other causes of errors in a computer system: a minute particle of dirt may become momentarily lodged on a magnetic recording media, a transient pulse may be induced in a logic circuit by a line transient or a lightning strike, or an algorithm for a calculation may result in an invalid address. The errors caused by these phenomena can be just as serious, and can have similarly grave consequences, as those caused by actual component failure.

The thesis of this paper is that the user of small computers must be concerned with a concept much broader than that of simple reliability. This concept is called system integrity. It encompasses all aspects of preventing errors, detecting them if they occur, reporting them to the processor, the operating system, and the user, and recovering from them in an acceptable manner.

In contrast to reliability, which is largely a matter of component failure rates, contributions to system integrity involve many diverse factors. These include such things as the functions available on the console which can affect servicing and debug time, the characteristics of the power supply which can determine whether a power dip will cause the system to go down, the extent and quality of the documentation which can affect service time, the degree and thoroughness of the testing during the system development, and the error detection features incorporated into the hardware and software.

In this paper, the focus is on two aspects of system integrity. The first is the detection and reporting of errors to the operating system by the hardware and their subsequent handling by the software. The second is the features provided by the software alone which contribute to system integrity. Frequently, tradeoffs must be made when deciding whether a particular integrity feature should be provided by the hardware, the software, or a combination of hardware and software.

INTEGRITY CONSIDERATIONS IN SMALL SYSTEMS

Considerations of system integrity in small real-time systems differ in some respects from those in larger general-purpose computers. In large systems, for example, features such as automatic retry of I/O devices and the use of error correcting codes are often implemented in hardware. In the small machine, however, the cost of additional integrity hardware represents a greater fraction of the total system cost than is the case in larger machines. Due to the highly competitive nature of most small computer purchase decisions, these added costs must be carefully evaluated by the designer in terms of the improved performance that they provide. Unfortunately, there is no single simple parameter by which system integrity can be judged so this evaluation is very difficult. Add to this the fact that many buyers of small computers place significant emphasis on price alone, without realizing the importance and value of system integrity features, and the designer is faced with a difficult decision.

Real-time computers are often concerned with speed and time responsiveness to a much greater degree than is the case with larger, general-purpose machines. It is often more important in the small machine to detect, report and recover from errors quickly and accurately because of the role of the real-time computer in the particular application. The system integrity features of both the hardware and the software must be carefully designed not only to provide this response but to do so without significantly degrading the response of the system under error-free conditions. For example, requiring that an additional I/O instruction be executed (to read a status word which indicates if an I/O device responded without error) after each command essentially doubles the effective I/O instruction time. On the other hand, hardware which automatically provides status information in the processor which is accessible to the program without execution of a lengthy I/O instruction greatly improves the time response to error conditions without excessively penalizing normal operation.

Another important consideration is the maximum 64 K word addressing space of the typical 16-bit small com-

puter. The system software must be written to leave the largest possible storage for user-written application programs. This consideration, which says that more system integrity should be built into the system hardware, must be weighed against the cost of both storage, I/O devices, and hardware integrity features in the highly competitive small computer marketplace. A further consideration is the uniqueness of almost every real-time application which means that the system software must be tailorable to each installation, allowing the customer to take special action and to eliminate as much unused code as possible.

A consideration that may be easily overlooked by system designers is that most users want the system to resolve or bypass as many errors as possible and to notify them only when the system cannot do so. Also, many customers are controlling critical processes and require that the system go down only for the most serious problems.

During the development of a small computer system, the above considerations must be combined with others to define a total approach to system integrity. The important point is that the system emerging from the design must provide a consistent approach to the detection, reporting, and recovery of all errors. System integrity cannot be sacrificed because many small real-time system applications are dependent on full and fast error handling.

HARDWARE/SOFTWARE INTERACTION

Illustrations of how considerations of integrity are reflected in the interaction of hardware and software are numerous. Take, for example, the occurrence of a power failure on a System/7. The approach taken accounts for the fact that many real-time processes cannot be restarted at the point where they were when the power failed. In many applications, the prime consideration is to know that a power failure is about to take place so that special action can be taken before the power failure occurs.

As the power dips to 85 percent of nominal value in a System/7, an interrupt is generated at the highest priority level. The DC energy storage designed into the power supply allows for a minimum of eight milliseconds of operation from the time the interrupt occurs until the system shuts down at the 60 percent level. The actual operating time available depends on the power supply loading and the nature of the power dip. Thus, the hardware detects and reports the power failure condition to the system software via the interrupt mechanism. The system software is designed to provide maximum flexibility in this situation because of the uniqueness of different applications. It will first queue a message containing the Instruction Address Register (IAR) contents and the Processor Status Word (PSW) for printing on the error logging device and will then queue a user-specified routine. Obviously, the message will not be printed if the system actually shuts down after eight milliseconds but, because of the hardware priority structure, the user can program his system so that the power failure routine receives control as soon as it is queued. Thus, if necessary, the user can shut his process down, initiate a backup system, or perform some other action tailored to his application. When power is restored and again reaches 85 percent of nominal value, an automatic Initial Program Load (IPL) sequence is initiated by the hardware. This results in an IPL program being loaded from an IPL device (e.g., a disk) which, in turn, loads and executes a specified user program. This provides a way for an unattended system to be reinitialized after a power failure. This illustrates the philosophy of automatically resolving error conditions whenever possible.

Another case in which system integrity is threatened is if the temperature of the system increases to the point where the operation of the semiconductor circuits becomes erratic or actual damage occurs. For this reason, System/7 provides automatic over-temperature indication to the processor and operating system. Temperature detectors are located at strategic locations throughout the machine to indicate when a high temperature condition exists. When activated, an interrupt is generated on the highest priority level to notify the programming system which can then queue the appropriate user-specified program. After a minimum delay of one second, automatic circuits power down the system before erratic behavior or damage can result. By providing an early warning of a temperature problem, the operating system and the user programs can take the appropriate action of notifying the operator or providing for an orderly shutdown of the system.

I/O error processing is perhaps the most important area of system integrity. An examination of I/O error processing in a System/7 again shows how full integrity can be provided by designing the hardware and software to work together to address the particular characteristics of a small real-time system.

First of all, I/O errors are categorized as those occurring when the I/O instruction is executed and those occurring during the I/O operation (data transfer). The former are called immediate errors and, after detecting the error, the hardware sets both a summary status indicator and a 16-bit status word. The system software can check the summary status indicator without having to execute the I/O instruction to read the status word. The system software need read the status word only when an error has occurred. Furthermore, the system knows immediately that an error occurred and does not have to wait and service an I/O interrupt. The advantage of this approach is that reading the summary status indicator takes less than one-fifth the time of reading the status word. The impact on processing speed and time responsiveness is thereby minimized.

If the summary status indicator shows that an error has occurred, the system software reads the status word which, in all cases, resets the device. (Incidently, this "read with reset" operation is very useful in troubleshooting.) The status word further defines the exact source of

error. The system software logs the error for later analysis by servicing personnel and then attempts to recover by retrying the operation, re-preparing the device, or some other procedure. Only when these attempts fail is the user notified that an error has occurred. Notification consists of a printed message containing the contents of the registers and the status word and an error code being returned to the user program. Once the user program receives control and checks the error code, many actions can be taken; the one chosen is user defined based on his application requirements. For example, if the operation was not important, the user can continue. On the other hand, if the failing I/O was critical, the user program could request an entire system reload.

The second type of I/O error is indicated at the time of the I/O interrupt by the interrupt status word (ISW). If any of the possible 32 bits is on in the ISW, a summary status indicator is set. Thus, the system software checks for a valid operation by merely checking the summary status rather than executing the much longer I/O instruction to read the ISW. This again is a time-saving feature allowing faster processing of I/O interrupts. Only when an error occurs (summary status bit set) does the software read the ISW words. The error is logged and retries are performed. If the error persists (a hard error), the system software notifies the user. A message is printed indicating the device address and the ISW words. Each bit in the ISW refers to a particular error so this message accurately describes the location and condition of the error. While costs can be reduced by associating more than one error with a particular bit, the ambiguity resulting from a printout of the status word significantly hinders trouble-shooting. As with the I/O command errors described above, the user program is notified of the error by setting the Error Status Word (ESW) in the I/O request. User action again has the same flexibility as that described above for I/O command errors.

Storage parity is a common and well-known hardware feature that contributes to system integrity. When available in a minicomputer, it is generally interconnected with the interrupt structure so that an interrupt is generated when incorrect parity is detected. The parity bit(s) is normally generated whenever a word is read into storage and is checked whenever a word is read from storage. Parity may be provided on a word basis with one parity bit for the typical 16-bit word or it may be provided with a parity bit for each 8-bit byte.

The concept of parity checking is applicable, however, to more than the storage unit in the computer. It can be utilized on all major data paths in the processor and channel to verify the accurate transmission of data and addresses. Figure 1 illustrates how this is implemented in the IBM System/7. A major portion of the data flow is represented by the storage, the local store which provides the index register stack, the channel, and the attached devices. Parity generators and checkers are provided for each of the major portions of the system. The failure to observe proper parity generates an interrupt which is serviced by the system software. As an aid to further

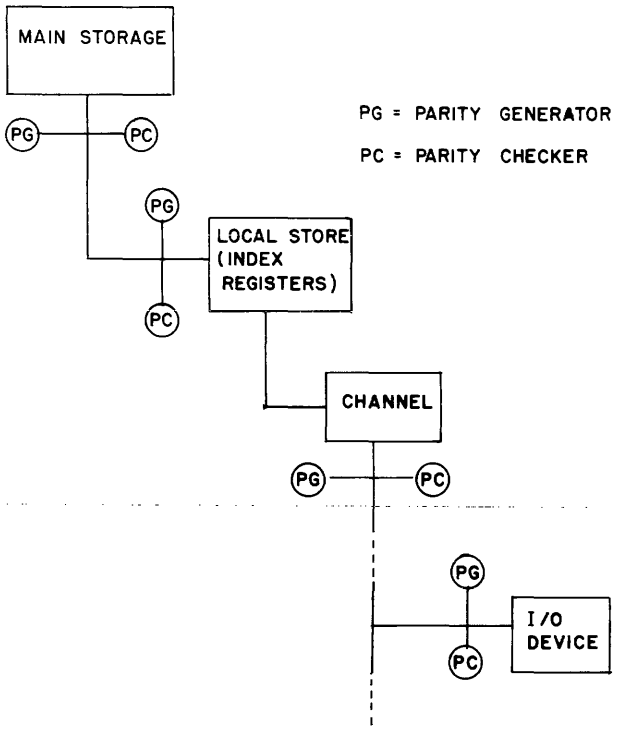


Figure 1—Parity generation and checking

defining the cause of the error, each parity checker generates a unique bit in a status word which is available to the software.

The presence of a unique identifier in the status word for the various sources of parity error is an important demonstration of the philosophy that system integrity features should provide as complete data as possible concerning the source of error. This is important since it affects the manner in which recovery can be effected. For example, a hard storage parity error might result in data and programs being relocated to avoid certain storage locations. A consistent storage parity error which is independent of storage address, however, might indicate a failing driver circuit and it would be necessary to shut down the system for repair. Similarly, a parity check in an I/O device may be relatively minor since the system software may be able to logically reconfigure the system, thereby avoiding the necessity of taking the system off-line. As a secondary effect relating to availability, providing maximum information as to the source of error assists the serviceman in locating the problem. This minimizes the repair time and, therefore, increases the availability of the system to the application.

When and where to generate and check parity is a design tradeoff which is based on the probability of error and cost. This, in turn, is a function of the logical and physical complexity of the data path. For example, in Figure 1, parity generation and checking is not provided between the channel and the local store since this is a logically simple data path which involves only one cable and two connectors in System/7. Thus, the probability of error is much less than in the case of the transfer between

the channel and a device, a quite complicated logical operation involving many cables and connectors.

The interface between the channel and the I/O devices in the System/7 demonstrates another type of hardware checking that can be incorporated in a small computer. The channel is asynchronous with a demand-response or "handshaking" discipline employed on all data transfers. In addition to the parity checking of data and addresses, this type of interface provides the possibility of timing checks and sequence checks. As an illustration, Figure 2 shows a portion of the timing sequence for the execution of the I/O instruction (PIO) in the System/7. At the initiation of the PIO instruction involving the outputting of data (e.g., a write operation), the Address Out (AO) tag line is raised to signal all I/O devices that a device address has been placed on Data Bus Out (DBO). The addressed device raises its Service In (SVC IN) line to indicate to the channel that it has recognized its address and is ready to receive data. The channel then places the necessary data on DBO and raises a Service Out (SVC OUT) tag line to signal the I/O device that the data is available on the DBO. In essence, this sequence is a two-way conversation between the channel and the I/O device in which each acknowledges the action of the other before proceeding with a subsequent action.

Since the signal propagation time through the channel and the device delays are known, it is possible to place bounds on the response time for each of the exchanges of information in the sequence. For example, based on worst case conditions, it is known in System/7 that the device should respond with SVC IN within 2 microseconds after AO is active. Timers have been provided in the channel to check this time. If the SVC IN response has not been received within 2 microseconds, a channel timing error is signaled to the processor and made available to the programming system. Similarly, the interval between SVC OUT becoming active and the response of SVC IN becoming inactive is timed and an error indicated if this time exceeds 2 microseconds. In addition to timing the individual sequences of signals during the execution of the PIO instruction, an overall timer is provided which signals an error if the execution time for the entire operation exceeds 10 microseconds. As in the case of parity, indication is provided for each of these timing errors, thereby making available the maximum amount of information to the programming system to be used in error recovery.

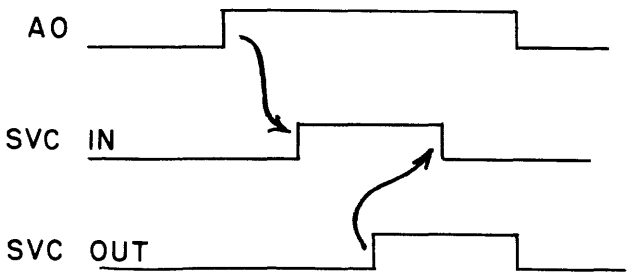


Figure 2—Channel I/O signal sequence

In addition to timing checks, sequence checks can be implemented to test for improper sequences of signals. For example, if SVC IN is active at the time AO becomes active, an error is signaled. Similarly, an error is recognized if SVC IN becomes inactive before SVC OUT is raised. This type of checking is indicative of the logical checks which can be applied to many other parts of the system. For example, in a system such as System/7 which provides multiple priority interrupt levels, a test can be provided to insure that two levels are not active simultaneously. The processor operation also can be checked to indicate if it is executing processor cycles when it should be in a dormant wait state.

The above discussion illustrates some of the checks that can be provided through hardware in the small computer. This philosophy can be extended throughout the processor, channel, and I/O devices. Other specific error indications that are provided in the System/7 hardware include:

1. A storage address has been requested which exceeds the amount of installed storage. (Generates interrupt.)
2. An invalid device address has been requested on the channel.
3. A device which is not installed has been addressed.
4. An invalid operation code has been detected. (Generates interrupt.)
5. The analog-to-digital converter is inoperable.
6. Multiple (or no) relays in the analog input multiplexer have been selected.
7. The fuse in the digital input circuit is open.
8. The communication attachment has an overrun condition.
9. The selected I/O device cannot execute the requested command.
10. The communication modem has detected an error condition.

These are but a few examples of how the hardware and system software must be designed together to furnish the system integrity required for real-time processes.

SYSTEM SOFTWARE CONSIDERATIONS

In small real-time computers, it is important that the system software be extended beyond just the handling of errors detected by the hardware. It too should detect errors before they can cause hardware and system failure. There are two major reasons for this. First, by detecting errors early, time is saved which could be beneficial to the user's process. Secondly, the system software must protect data (especially data stored on disk) which could possibly be destroyed without causing a hardware error. As with hardware/software interaction, it is necessary to design the support for minimum storage and to cause minimum delay in processing valid operations.

Early detection of user programming errors begins with a good level of diagnostics in the language processors (assemblers, compilers, etc.) used to prepare those programs. These diagnostics should not only detect, but should accurately describe, all syntactical errors. If a macro assembler is provided, it should thoroughly check the validity of all parameters and the relationships between macros. A good set of program preparation diagnostics greatly aids debugging and speeds installation. Since time is much more critical during program execution than during program preparation, as many programming errors as possible should be detected during preparation. Host preparation (that is, using a large general purpose computer to prepare programs for the small real-time computer) is often an advantage here. The significantly greater resources available with the large machine allows extensive diagnostics without unrealistically increasing program preparation time.

During program execution, more time can be saved with the proper software integrity design by detecting errors early. For example, most systems that support multiple requests to serial I/O devices queue the request (either first-in/first-out or by priority) until the device is available. When the request list reaches the top of the device queue, the proper I/O commands are executed. I/O request lists are frequently constructed during program execution which means they cannot be validated during program preparation. As a result, all validation in these systems must take place at execution time. Time can be saved by validating the request before placing it on the queue.

Consider, for example, the case of a program that makes an invalid I/O request to the System/7 software. Before the request is queued, the error is detected and the user notified immediately. If no validation was done, the request would remain queued until it reached the top of the queue and, perhaps, an I/O command had been executed. The error would be detected only when the I/O interrupt was received. Not only would the user not be immediately notified, but time would have been wasted.

Since most small computer applications are unique, the system software frequently returns to the user program after processing an error. Thus, the user must be provided with a broad range of recovery facilities. When an error is reported back to a user program, system software facilities should be available to that user program to recover from that error. For example, on an I/O error, the user program may want to retry the operation on an alternate device, interactively communicate with the console operator or, in the case of a serious error, sound an alarm or reload main storage from disk. Or, in the case when a programming error is detected, the user may want to load a new copy or version of all or part of his program from disk. The system software should be designed to provide these functions easily.

Many small computer applications rely on a high speed secondary storage device such as a disk or drum for programs which are loaded into main storage when required. Data stored on disk must also be protected. For example, if data collected and stored on a disk during an experiment is later destroyed because of some error, the experiment may have to be run again. This protection of programs and data stored on disk must be provided by the system software.

One method of protecting programs is by the software providing a "read only" data set facility. Programs are stored in such data sets and the system software prevents user programs from doing disk write operations to such data sets. Of course, the system disk utility that initially loads the program to disk is permitted to do the write operation.

A frequently used method of providing disk data protection is an OPEN function. Data sets are named on disk and all accesses to the data set are made using the data set name. Programs must first OPEN the data set and this operation provides the system software with the disk address boundaries of the data set. Subsequent accesses to the data set are checked by the system software to verify that the access remains within the data set boundaries.

Additional protection can be provided by the system software by checking a volume label on each disk pack. Thus, if the wrong disk pack (or the right disk pack at the wrong time) is loaded, good data will not be destroyed.

The features just mentioned are all part of the data management portion of the system software. In designing data management systems, careful consideration must be given to the protection required on the one hand and the overhead in storage sizes on the other.

SUMMARY AND CONCLUSIONS

This paper has described the general concepts of system integrity and has given some particular examples of how it can be provided in a small real-time computer. It is important to realize that *total* integrity is necessary—any lapses in the support reduces the integrity provided by the entire system.

Many factors must be considered before purchasing a small computer. Some of the factors are unique to small computer applications and may not be considered, for example, when purchasing a larger data processing system. This paper has attempted to focus in on one of these to emphasize the requirement for good system integrity and to indicate how it might be furnished by both the hardware and the system software.

The design and implementation of a small scale stack processor system*

by MICHAEL J. LUTZ

State University of New York at Buffalo
Amherst, New York

INTRODUCTION

A striking phenomenon in the current state of the art in computer technology is the rapidly growing power of mini-computers. One reason for this power is the ability of small computer systems to adapt to specific uses, making them an attractive and economical alternative to large- or medium-scale general purpose systems for many applications. The provision of micro-programming on many of these systems has much to do with this adaptability, since it permits the efficient design and implementation of a virtual machine suited to the needs of the particular application or intended use of the system. In this way the bare hardware can be molded to support the necessary (and often sophisticated) data and control structures desired.

In this paper one such application is presented—the implementation of a machine designed to support block-structured languages effectively in its "hardware". The resulting architecture is quite complex, and would most probably not be economical for such a small scale system (approximately 8K words of memory) were it to be built directly in hardware. Microprogramming made the implementation practical, however, and it is presented in this paper to provide an example of the potential which mini-computers possess in the current era of computer design.

The remainder of this paper is divided into 4 major sections. The following section gives the rationale for designing a machine around block-structured languages, and those data and control constructs which are considered important enough to be explicitly included in the design. It is followed by a section on the way these abstract constructs are actually supported in the architecture, as well as some examples of how this leads to conceptually clean implementations of various features in the system. A quick presentation of the micro-machine and the emulator which creates the virtual machine on top of it is then presented, with some emphasis on the resource mapping used and the microcoding techniques employed. Finally, a brief discussion of the advantages of microprogramming is given, both from the economic standpoint and in relation to technical benefits.

* This work was done as part of Project Mu in the Department of Computer Science and is supported in part by NSF Grant GJ-993.

DESIGN CRITERIA AND CONCEPTS

The design of this processor was motivated by the desire to create a computer whose organization was explicitly language-oriented in nature. To us, (and to others, see References 6, 7 and 11) the convolutions compilers must go through to produce decent object code for the majority of computers indicate a great disparity between the desires of the users of a language and what is provided for in the machine. The added overhead of run-time software to create the illusion of the required environment further attests to the fact that computers in general do not behave as we would like them to. Thus we desired a processor whose operation included features normally supported by these run-time routines, and whose organization was tailored to the compilation of efficient object code in a straightforward manner.

A critical parameter in the design was the class of languages around which to model our machine. It is not at all obvious that any one architecture could be found to provide effective support for every higher level language, since there are a multitude of different and conflicting conventions in use. For example, the definition of accessing environment in ALGOL-60 and APL/360 are quite different, and it may well be that any construct general enough to support both definitions may not enhance the implementation of either. The class settled upon was the block-structured languages, especially those semantically close to ALGOL-60.

This choice was not made capriciously; rather it was based on observations about the current state of programming practice. One strong motivation for the choice is that there is quite a bit of experience in the design of compilers for such languages, and the data structures most convenient for their support have become fairly well known. Also, many algorithms have been coded in these languages, giving us two additional benefits: (1) insight into the most used features of such languages and (2) a large base of programs with which to compare our design with other systems.

These features, attractive as they might be, are insufficient to justify the choice. More convincing are arguments which arise based on the desire to implement an entire system with a multitude of tasks, some of which are co-operating and others of which are independent. It was a major design goal to create a machine which was "complete" in the sense that there were few (if any) operations which necessitated an escape from the architectural framework. In short, a design was desired which allowed the operating system and all of its related tasks to be coded in a higher-level language of the type the computer was designed to support.

In this light, block-structured languages appear to be by far the most attractive alternative. The multi-tasking features necessary in any reasonable contemporary operating system can be incorporated quite elegantly when one considers nested tasks to be generalizations of nested procedures; we shall see that user jobs can all be considered sub-tasks of the operating system in this fashion. The fact that languages modelled on ALGOL-60 [NAU63] permit recursion provides impetus for an object code structure which operates in a re-entrant fashion, and the well-defined boundaries of procedures and blocks provide a natural division of a program into self-contained logical units. This in turn provides the opportunity for one copy of a procedure to be shared by many different tasks, decreasing the average amount of memory required by each. Finally, we will see that the rules of scope in block-structured languages provide a consistent and well defined manner in which to share code and data among several processes. Thus, in the context of an entire system, the choice of block-structured languages appears to be more than justified.

Having decided to design a computer targeted for this class of languages, what should be provided in "hardware" (quoted, since it is irrelevant at this point whether it is hardwired or microcoded) in the way of support? What internal data structures, and what instruction set is best suited to the needs of these languages? And given that the ultimate aim is an entire, cohesive system, can these be generalized or extended to provide such desirable features as protection and relocatability of code and data? The rest of this section is devoted to the answers chosen to these questions, with particular emphasis on the addressing scheme employed, the inclusion of explicit stack structures and the effects this has, the usefulness of information tagging in the design, and the provision of semaphore constructs for task co-ordination and access control in a fashion consistent with the rest of the architecture.

One common feature of block-structured languages is the static definition of directly accessible variables and procedures. The items which can be directly referenced at a given point within a procedure or block can be recursively defined as: (1) those variables, parameters and procedures declared local to the code body in question and (2) the variables, parameters and procedures accessible to the statically enclosing code body (i.e., the surrounding block or procedure in a listing), except those possessing names (identifiers) which are the same as some local item. This is combined with a definition of dynamic environment at any point in the execution of the program: since recursion results in many different allocations of variables, it is the most recent occurrence of the identifier which applies at any time during execution.

Based on the above observations, we note that any item which is accessible at a given point in a program can be identified by a pair of integers called an address couple. The first component is the static nesting depth of the variable or procedure declaration, the second is its ordinal in the declarations for the particular code body in which it occurs; for instance, the first declaration in the outermost block would be designated by the couple (0,1). While it is not true that a unique address couple is assigned to each identifier occurring in a program, it is true that at any point in the listing of a program, the scope definitions above insure that an address couple identifies a unique item.

Assuming the environment is set correctly, so that the first component of the couple can be used to locate the base of the procedure data area active at the indicated static level, this provides a convenient way of locating items in the machine. Since the object code is not directly concerned with setting the environment, this method automatically provides re-entrant code. These observations led us to adopt this as the normal addressing mode in our design.

As mentioned above critical to the success of the addressing scheme is the proper setting of the environment, and this motivates the introduction of the mechanism which is the central unifying concept of the design—the "hardware" recognition, maintenance, and use of stacks (or LIFO lists). In our opinion, though a stack-oriented system is not the most general support for the information structures that arise in the execution of block-structured language programs,[8] it is the best trade-off between generality and practicality, especially for the majority of applications.

The reason the stack is so useful relates to the method in which procedure calls and returns are defined in block-structured languages. A dynamic sequence of called procedures are exited in the reverse order of the invocations, which is modelled exactly by the operation of a stack. Also, the facts that local variables need not be allocated until the procedure is called, and that they become inaccessible upon procedure exit means that the stack can be used to provide storage for the variables at each procedure invocation. Finally, if the machine provides a method of establishing the base of a procedure's data in the stack at each call, then the stack also provides support for the addressing scheme we desire.

The uses of the stack are extended to include the manipulation of data. For example, this provides the base for efficient evaluation of expressions in Polish postfix form; not only is this a convenient type of code for compilers to generate, but considerable code compaction can be accomplished by the elimination of explicitly addressed operands for such operations as ADD—they are implic-

itly located by the current top of the stack. Thus the stack serves as the center of the entire execution of a program. In the section on the actual implementation of these concepts in the architecture, it will be shown how stacks can be used to generalize the concept of procedure call to interrupt processing and multi-tasking.

Since the "hardware" of the machine is charged with much of the responsibility for maintaining the stack and insuring the proper environment during execution, and since the integrity of the system depends heavily on this, it is crucial that mechanisms be supplied to make this relatively easy to accomplish. For this reason, tagging of information in the machine was instituted. Tagging means that the homogeneous memory common to most computers is replaced by a memory whose items have distinct "hardware" semantics associated with them, based on the tag they possess.

A more complete description of the uses tagging has in the design will be given in the following section, but as an indication of the value of this construct, we note that besides "normal" data items, unique tags are used to specify: (1) words that mark the beginning of the accessing environment for each procedure activation in the stack, (2) indirect references to other in-stack items for passing parameters by name, (3) descriptors for code and data residing in memory separate from the stack, with automatic bounds checking and (4) link words to aid the operating system in memory allocation.

Any design which claims to support an entire system must provide some means for co-ordination among the tasks active in the system. In particular, there must be some mechanism for controlling access to critical portions of code and data. We have already mentioned that constructs exist to describe code and data outside the stack, and these were extended to provide "semaphore" constructs, based on the P and V operations of Dijkstra.[4] In essence, the first task referencing the descriptor gains exclusive access to the resource described; an interrupt is generated when other tasks attempt to claim it, allowing the operating system to queue further requests for the resource. When the controlling task releases the resource, another interrupt is generated, informing the system that one of suspended tasks can be re-activated. In this way, primitive task co-ordination constructs are introduced in a consistent manner with the rest of the architecture.

We note in passing that many of the constructs and concepts we have employed parallel the features found in the Burroughs B6700/7700 Information Processing Systems.[1,2,5,14] Though the choice of the stack model for block-structured processes was chosen with explicit knowledge that this was also the mode of operation of these systems, most other similarities were discovered by us only after inclusion in our design. For this reason, we feel that these constructs are a function of the general architectural framework employed. We believe our design is unique in many respects, however, not the least of which is the one most important to this paper: that to our knowledge this is the first time such a complex design has been provided on a small scale system.

REALIZATION OF THE CONCEPTS IN THE ARCHITECTURE

We now turn to the subject of how these constructs that were considered desirable are physically realized in the "hardware" of the machine, which we call the Buffalo Stack Machine, or BSM. First we give a more detailed description of the tag bit concept and how it is used. We then turn to a brief description of how the stack for a process is maintained in memory and the manner in which the address couples are translated to absolute addresses. Next it is demonstrated how this concept is extended to link the code associated with the program and the operating system into the active stack. Finally, we briefly discuss the generalization to multi-tasking which is possible, and the manner in which interrupts are fielded.

The BSM memory is composed of 36-bit words; however, only the low order 32 bits can be considered data in the traditional connotation of that term. The upper 4 bits serve to place the word into one of 16 "hardware" recognizable classes, which is the tagging described previously. Tag type 0, for example, is used for "normal" data words and is the type most frequently manipulated by user processes.

Other tag types are reserved for the code and data descriptors used to point to areas (segments) apart from the memory for the stack, but containing task related information (data or code). For example, the address couple of an array variable would locate a data descriptor for a separate segment allocated to the array. This is trivially extended to multi-dimensional arrays by allowing the segment itself to contain descriptors. This memory allocation scheme has two major benefits: (1) the average amount of contiguous area assigned to a segment (the stack itself is just a special segment) is lowered and (2) the arrays and code segments not currently being referenced can be removed from main memory, which aids the implementation of virtual memory. This second feature is made possible by a presence bit in each descriptor, so that attempts to access a segment through a descriptor whose presence bit is off causes an interrupt indicating which segment must be made present in memory. This shields the task from the fact that not all of its code and data is immediately accessible, and permits it to run in an effective memory larger than that physically available.

How the stack concept is implemented and the manner in which address translation is performed are interconnected to such a degree that it is easiest to describe both together. The following explanation uses the small program in Figure 1 and the corresponding stack structure at a particular point in its execution in Figure 2 as an example. The language used is PL-BSM, the implementation language used on BSM to write supervisor routines, including the entire operating system. Its syntax and semantics are very similar to XPL,[12] and the functioning of the short program, whose main feature is a procedure to recursively compute the factorial of an integer, should be self-explanatory.

```
1.        begin
2.            declare A fixed, B fixed;
3.  FACT:     procedure ( N ) : fixed;
4.                declare N fixed;
5.                if N=1 then return 1;
6.                return N * FACT (N-1);
7.            end /* FACT */;
8.            A = 5 ;
9.            B = FACT(A);
10.           PRINT(A,B);
11.       end
```

Figure 1—Example program in PL-BSM

First, we note in Figure 2 the set of registers collectively called the Display, which are used to implement the address couple translation. Each task in the system is identified with a stack segment, and when the stack is assigned to a processor, each Display register is set to the base of the data area in the stack for the procedure activation whose environment is accessible at the corresponding level. The location addressed by the couple $(i,0)$, for any level i, contains an Environment Control Word (ECW), which is a distinct tag type used to insure that the proper accessing environment is set. An ECW has links in it to both its dynamic (calling) and static ancestors' ECWs, so that when a procedure is exited, the ECW for the caller can be found using the dynamic link, and the Display can be reset from the static link chain beginning there.

It should be quite obvious now how the address couple scheme is implemented. The first component indexes into the Display and reads the address in the proper register; the ordinal is then added to this to get the address of the desired quantity, or of a descriptor or pointer to it. The

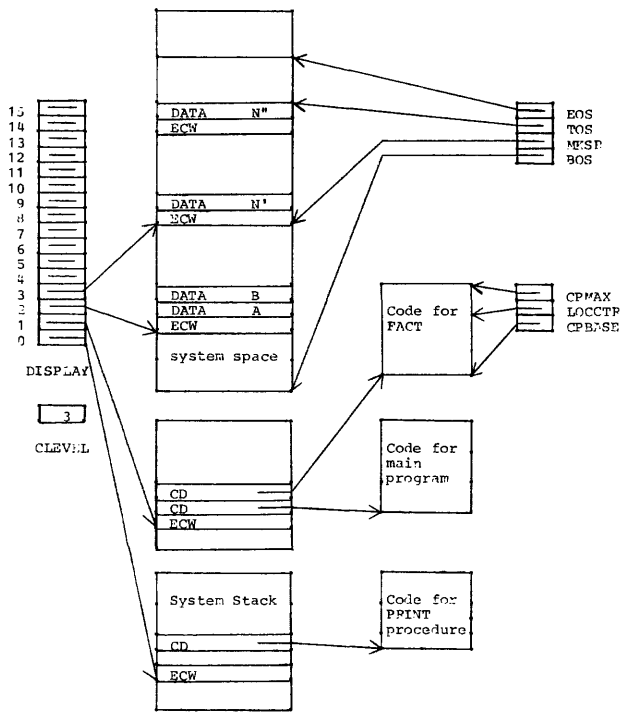


Figure 2—Stack configuration at line 6 in program

CLEVEL register is used to hold the static nesting depth of the currently executing code, and determines the highest numbered Display register which contains a valid address; in this way, address couples can be checked for errors before any damage is done.

While we are discussing the addressing, the extension of the address couple method to locate code as well as data will be presented. At present, the first two Display levels are used to access the descriptors for the code segments. DR[0] points to an ECW in the "system stack", and address couples of the form $(0,i)$ locate descriptors for system procedures. In a completely analogous manner, DR[1] is used to access the code descriptors for the particular user program being executed, which also reside in a separate segment. Technically, both of the segments are stacks, though they represent permanently suspended tasks and are never activated. The reason they occupy separate segments is to allow the sharing of the system stack by all tasks in the system, and in the same fashion, to allow several tasks executing the same program (e.g., a compiler) to share the same code stack. In this way the number of copies of procedures is kept down, with a consequent rise in memory utilization.

This also serves to introduce the fact that the conceptual stack for a task may include portions of other physical stacks. In this case, the other stacks are for tasks that are never active on a processor. It should be obvious that this need not be the only way to interlink stacks, and indeed, it is not. A procedure in PL-BSM may be called, in which case further processing by the calling routine halts until it is returned to, or it may be started as a separate task, in which case it is assigned a distinct stack, and both the created and invoking tasks can proceed in parallel. For either method of invoking the procedure's code, the accessing environment must be identical for consistency, and this may well mean that the Display settings for the new task refer to the creating task's stack, allowing the two to share information.

At this time we turn to a discussion of the constructs used to check the validity of code addresses generated. As shown in Figure 2, the code segment for the procedure currently being executed has associated with it 3 processor registers which keep track of the location of the next instruction (byte) (LOCCTR), and check to make sure the code addresses are in bounds (CPBASE, CPMAX). CPBASE is also useful on branch instructions, which are always relative to the start of the segment. Finally, we discuss the registers used to maintain and manipulate the stack structure. The BOS and EOS registers are used to check for stack underflow and overflow respectively; however, EOS, as can be seen in Figure 2, contains an address which leaves some space at the end of the stack when overflow is detected. This is used to provide room for handling the resulting interrupt; the method of interrupt processing in general will be explained later, at which time the necessity for this extra space should become evident. The MKSR register points to the start of the currently active environment and is used in setting the dynamic and static links at procedure call or return.

TOS contains the address of the next available word in the stack, and is used to locate the implicit operands for the arithmetic and logical operators, as noted previously. The TOS register also gets incremented and decremented by the code for procedure call and return, and insures the stack space being used by each active procedure is preserved.

We conclude this section with two miscellaneous remarks, the first in respect to the way stacks (and thus tasks) are identified, and the second on how a consistent and rather elegant implementation of interrupt processing is possible with this design. In this system, each stack segment has an associated descriptor pointing to it, and all of these are collected into one segment, known as the stack vector. This segment in turn has a descriptor pointing to it at a predefined location (address couple (0,2)). Thus a stack (and its associated task) can be identified by the index of its descriptor in the stack vector. Looking at Figure 2 once again, note that a portion of the user stack below the first ECW is marked "system space"; it is here that task-related control information, such as priority, is kept. The stack vector concept allows the system to access these items via a doubly subscripted reference to the stack vector descriptor.

The fact that the "hardware" has knowledge about the way stacks are used in procedure calls allows a very consistent implementation of the interrupt code. The primary interrupt procedure has its descriptor at the predefined address couple (0,1); when an interrupt is to be signalled to BSM, the stack of the current task is used by the "hardware" to force a procedure entry identical to a software call. In this way, the interrupt procedure is coded *exactly* as any other procedure, and uses the same exit mechanisms to return control to the interrupted process. This is all made possible by the fact that the "hardware" is knowledgeable of stack structures in memory, and that there is a well-defined sequence of operations to invoke any procedure. It should now be apparent why EOS is set to cause overflow while there is still some stack space remaining; it gives the interrupt procedure space to work in. Further stack overflow interrupts will not be generated after the original one, since this exception is detected only in the user mode, and invoking the interrupt procedure automatically sets the processor in system mode.

FIRMWARE IMPLEMENTATION OF THE BSM

The quoting of the term "hardware" in the previous section when referring to some feature of BSM was done intentionally; since the design of a conceptual machine was being explained, it did not matter whether or not its features were actually hardwired or not. Of course, in our implementation the structures outlined above are realized by a microprogram which forms an emulator for BSM. In this section we turn to the design of this emulator, which we call Kielbasa II, a name which derives from a variety of Polish sausage and is thus a reminder of the Polish instruction set on BSM. It is the direct descendent of the original Kielbasa emulator (described in Reference 9), and though the two emulators were designed for two quite different micro-computers, many of the techniques used in the original have carried over to the current version. Thus we feel a discussion of the emulator is in order to present our observations.

In order to understand the design decisions made in the emulator, a short presentation of the facilities available in the Burroughs B1700,[3] which is the computer used by Project Mu, will be given. Though most of the features to be described are present in all models, the particular model described here is the B1726.

Though the internal data paths in the machine are generally 24 bits wide, the memory of the machine is bit addressable, and can be read and written in units of 1-24 bits at a time in either direction from the address specified. The combinatorial section to do the arithmetic and logical functions can also be scaled to work on operands from 1-24 bits in length, and includes residual control to allow for multiple length operations. Also provided is the capability to shift both single and double length quantities, and the ability to extract an arbitrary field of contiguous bits from one of the 24 bit registers.

Hardware condition bits of the B1700 are held in a set of 4-bit registers (external interrupt pending, clock interval, arithmetic conditions, etc.), and certain 24-bit registers are re-mapped into 4-bit registers for added flexibility. Various micro operations can test and manipulate the bits in these registers, and the non-dedicated ones are especially useful for recording status of the machine being emulated. Finally, there is a scratchpad of registers for general purpose use; these can be thought of as either 32 24-bit registers or 16 double length registers, and are most useful for holding the contents of virtual machine registers.

In addition to this, the B1726 has a separate control memory for fast execution of microinstructions, though execution can also proceed from main memory at a reduced rate. The control memory can be overlayed from main memory, but its operation is effectively read-only (there are no operations for reading the contents of control store words). This added memory permits the emulator to run without conflicting with main memory operations, and micro-instruction execution can overlap reads and writes to main memory.

We now turn to the manner in which Kielbasa II uses these physical constructs to emulate the features of the BSM. First we describe the mapping of the resources in BSM onto those of the B1700, including resources used by the emulator which are hidden from the explicit view of the conceptual BSM. Then we turn to some examples from the emulator of its operation and the microcoding techniques employed. In this way we hope to demonstrate the viability of the use of microcode to implement complex constructs and systems.

Most of the registers used in BSM are kept in the scratchpad, where we use it as 32 24-bit quantities. Sixteen of the registers are used as the Display, the other

half being used to implement the remaining processor registers of the BSM. All of the stack and code page control registers are kept here, as well as some registers to hold state information about BSM used only by the emulator. For example, one register (NEXTINSTR) is used to hold the address of the instruction following the one currently being executed, which is addressed by LOCCTR. It is necessitated by the complexity of location counter updating, as will be shown later on in this section. Another pair of registers is used to hold internal interrupt parameters when such a condition is detected. The design of the system insures that only one internal interrupt can be set in the interval between successive checks in the emulator for interrupts to be generated to BSM, and since these conditions cannot be masked out, there is no danger in "losing" an interrupt by this method.

Unfortunately, the same is not true for external interrupts. Several such interrupts may be pending at any one time, and in addition, the BSM operating system can disable them. A small portion of main memory is set aside to queue these interrupts, and when the external interrupt pending flag is set and the interrupts are enabled, the first entry in the queue is "passed up" as an interrupt to the BSM processor. The external interrupt pending flag is reset if and only if the queue is emptied.

Another portion of main memory is reserved to hold the routines for Kielbasa II which will not fit in control store. This is all that will be said on this subject at this point, but we will see later that a major decision in the emulator is what to keep in mainstore, and how to execute it.

Two of the non-dedicated 4-bit registers are used to hold processor status across all operations in the system. One is used as the CLEVEL register; the ability to increment and decrement these registers and test for overflow and underflow is quite helpful in detecting display overflow and underflow. The other 4-bit register is composed of 4 separate emulator state bits: internal interrupt pending, external interrupt pending, external interrupts enabled and a bit to decide whether or not LOCCTR is to be updated to the value in NEXTINSTR.

We now turn to a discussion of the structure of the emulator which manipulates these resources. First, two techniques that have been used extensively (and quite successfully) in the design of Kielbasa are discussed: the structuring of all the code for BSM operations into subroutines and the decision to overlay certain routines from the main memory into control store; both of these techniques were used in the earlier version of Kielbasa and found to be quite advantageous. The fact that the B1726 provides hardware support for both of these techniques (an address stack for subroutines and an OVERLAY microinstruction) has made their use even more attractive.

Subroutining allows the use of code from several parts of the emulator, with many of the same benefits which accrue from this practice in programs for any machine. For example, the operations which compare the 2 top-of-stack items for various relations use the SUBtract routine, which is also a separate operation in the machine. This method of organization is extremely valuable in the emulator, since many of the complex operations are implementable as calls on more primitive ones. The interrupt procedure, as has been noted, can have a call forced by the emulator; this is accomplished very easily in Kielbasa by having the routine which generates interrupts call the various opcode routines for procedure entry in the same sequence as a software invocation. This insures the criterion is met that a hardware or software call on a procedure be indistinguishable in its effects on the stack.

The provision of control store on the B1726 leads to consideration of the best allocation of routines to this fast memory. The indications are that the entire emulator will not fit into the control store, and thus some functions must be kept in main store. The questions that arise in this case are what routines should reside in control store, and of those consigned to main store, which (if any) should be overlaid into control memory when needed. At present, the routines for opcodes which are legal only in the system mode of BSM are kept in main memory, and they are overlaid only when the operation they perform involves considerable looping, it being felt that the overhead in overlaying other routines is greater than the loss in speed inherent in executing from main store. These are preliminary (and rather intuitive) decisions, and we are waiting until the system is in use to evaluate this allocation more thoroughly.

At this point, examples of the way in which the emulator operates are in order; it is hoped that providing these will give insight into convenient ways of structuring emulators, and also demonstrate the way in which complex virtual machines features can be effectively realized through microprogramming. The examples that have been chosen are instruction counter maintenance, address resolution, and the microprogrammed "intelligent" I/O channels.

Unlike conventional machine designs in which the location counter is advanced after each instruction, the operation of BSM causes the location counter maintenance to be more complex. For example, the location counter should not be advanced if a code or data segment is not present; after the segment is brought into memory, the operation must be re-started. On the other hand, detecting an interrupt for arithmetic overflow should cause the location counter to be advanced before the interrupt procedure is called, since if the system decides to resume the task in spite of overflow, it should continue with the next instruction. There are many other instances of this dilemma in the machine, and thus a mechanism was needed to update the location counter correctly. A bit in the emulator status register indicates whether or not the location counter should be advanced; it is set just prior to each instruction fetch for BSM, and if the routines detect conditions which imply the location counter should not advance, they reset the bit. At the end of the instruction cycle, this bit is tested to determine whether or not LOCCTR is to be set to NEXTINSTR.

With a variety of constructs in the machine to cause indirection, (stack relative pointers, indexed descriptors,

etc.) the problem of address resolution becomes quite complex. In order to insure consistent interpretation of indirect chains, the addressing logic of the BSM is embodied in one subroutine, GEA (for Get Effective Address). The purpose of this routine is to follow the chains of pointers and descriptors (doing indexing as necessary) until the desired item is found. Since the requirements of various calling routines are quite different (the store operator expects to find a data item address; the enter procedure operator expects a code descriptor), GEA is passed a set of parameters telling which types of items are expected. On return to the caller, the address of the last item in the chain is given, as well as an indication of any exceptional conditions which caused the exit. The calling routine may resolve any conflicts or cause an interrupt to be set for the BSM operating system to handle. In this manner, the complex address resolution is handled in a straightforward and consistent fashion.

The I/O channels BSM communicates with serve to interface the physical I/O of the B1700 with the logical I/O incorporated in BSM. For one thing, this permits the I/O channels to be "intelligent" in the sense that they obey the same tag-implied semantics as the BSM processor itself. An attempt to overwrite a memory link word, for instance, will cause an I/O error termination and interrupt the processor. This gives us an example where microprogramming not only extended the power of the machine, but allowed this complexity to carry over into its communication with the external world.

This concludes the section on the emulator which supports the BSM architecture on top of the B1700 hardware host. It is hoped that this presentation increases the understanding of the power given to systems through a microprogrammable processor. In addition, it should provide some insight into the techniques of microprogramming found useful in creating an integrated, consistent emulator for the desired virtual machine.

ADVANTAGES OF A MICROPROGRAMMED IMPLEMENTATION

This section is intended to detail some of the advantages microprogramming provides. Though the examples are drawn from the BSM implementation on the B1700, we believe these advantages apply in most cases where this technique is used to realize complex or special purpose designs.

All of these features can apply to any computer endowed with microprogramming capabilities, but they are especially important in small scale system and minicomputers, since they are a source of great power and adaptability. The two broad areas to be touched upon are those of the economics of these small systems, and the technical benefits.

A concise statement of the economic situation is that without a microprogrammable computer, BSM would be nothing more than a paper machine. Though it is economical to incorporate such complex constructs in the hardware of a system the size of the Burroughs B6700, it would not be competitive in the small machine market. However, as demonstrated in this paper, an efficient realization can be microcoded on relatively inexpensively, and sophisticated constructs become cost effective on small machines when they are microprogrammed.

Another indication of the economic attractiveness of such a system, especially one equipped with writable control memory, is its adaptability, which has been alluded to above. The same machine which has been so effectively in the support of BSM is also being used to emulate such diverse designs as an APL machine, a string processing machine (for SNOBOL4), as well as a wide variety of conventional designs, and there appears to be no reason why it could not also be tailored for such special purpose uses as a text editing machine or a data base management machine. We emphasize that this is not a feature of the particular machine used, but to a greater or lesser degree a characteristic of the entire genre of microprogrammable minis. This adaptability via microcode is one of the main reasons for the growing power of small systems.

There are also technical benefits which accrue to the use of microcode in systems implementation. One of the most striking benefits which was discovered during the design of BSM is the ability for primitives to migrate from one level of implementation to another. This means that a construct which is frequently used at the virtual machine level can be directly implemented in microcode, making it truly primitive to the emulated machine; conversely, a little-used feature in the virtual machine, or one whose complexity in microcode outweighs the advantages of its inclusion, can be broken up into less complicated operations, placing the burden of implementation on virtual machine software. Examples of both directions of migration will be given next to demonstrate the value of this feature inherent in the use of microcode.

It was mentioned in the section on the constructs included in the design of BSM that semaphore constructs were added to aid in task co-ordination and access control. In the original design of BSM there were no such constructs at all. Since the design was of a single processor system, it was felt that such controls could be implemented by explicitly called gate-keeper procedures (in BSM code) which would perform the same services as semaphores.

When the writing of the operating system was begun, it became evident that this mode of operation was unsatisfactory. The amount of code necessary in the procedures would cause high system overhead, but even worse, any coding flaws which allowed some task to ignore the conventions could have disastrous consequences for the entire system's integrity.

Thus, the decision was made to provide more immediate support in microcode, and it was discovered that the semaphore operations could be easily added in a consistent manner, as described previously. In this the first implementation of the BSM, these provide a sufficient means of control; we are awaiting the results of the analy-

sis of the system's performance (described later in this section) to see if the choice was optimal as well.

An example of reverse migration is demonstrated by the memory management operations. In the original Kielbasa, memory management was incorporated into 3 system opcodes—GETSEG, RETSEG, and COMPACT. Using the first two, the operating system for BSM allocated and deallocated segments in memory, and the third was used for garbage collection in case no contiguous space was large enough for a segment which was to be brought into memory. However, the code for these three operations was considerably more complex than that for any of the other operations in BSM, and in addition, the BSM operating system did not have as much control over the allocation of memory resources as it did over the other resources of the system. These two observations together convinced us that memory management was not balanced with respect to the rest of the system.

To alleviate this situation, the current design has added more but less complicated operations. A better balance seems to have been struck by this approach; the coding in the emulator is proportionately less complicated, and supports memory management by the operating system without forcing any particular discipline on it (apart from that already dictated by the variable size segment organization implied by the architecture). This, then, is a case where an operation was considered to be unbalanced to the side of the emulator, and was broken up into simpler, lower-level primitives.

Another benefit in the use of microcode is the ability to design both the machine and the software that will run on it in parallel, allowing close co-operation. Instead of the usual process whereby the software has to be designed around an existing machine, the considerations of both software and the "hardware" can interact to produce a (hopefully) better design. Both the inclusion of semaphores in BSM and the dissolution of the original memory management opcodes described above were a direct result of this process.

Another example of the value of this co-operation arose when in the design of the compiler for PL-BSM it became apparent that there was a grave flaw in the environment setting mechanisms as originally defined. Since the correct environment for each call is essential to the proper operation of the entire BSM system, had the error been frozen in hardware, it would have been an embarrassing (and expensive) mistake. However, the changes that were necessary in the emulator were easily made with little cost.

These three examples, plus many other minor changes to improve the operation, were possible only because the BSM was being emulated. We believe this is one of the greatest benefits of microprogramming, and can be of even greater importance when trying to optimize a machine for a specialized set of tasks.

The final comment we have to make on the advantages offered by microprogrammed implementation of systems is in relation to the evaluation of the result and modifications based on this. Once a basic design outline has been chosen, the particular operations to include still remain to be specified. All too often the decisions are based upon limited previous experience and intuition as to what is appropriate, and little evaluation is done later of the result. Of course, if the machine is frozen in hardware, the evaluation would probably be of little use except as a guide for future designs.

When emulation is the implementation technique used, however, the evaluation can be very useful. Recent experiments have shown the power of microprogrammed measurement techniques[15] and how these can indicate characteristics of a machine's operations which are not intuitively obvious. Thus, Kielbasa will have microcode embedded in it to gather statistics on the frequency of use of operations, length of indirect address chains and other relevant measures of BSM's performance.

Based on this, we plan to re-evaluate the decisions made in this, the first iteration of the design. It could well be that more constructs will be directly implemented in microcode, or that little used facilities will be even further subdivided. In the two instances cited above, where operations were changed in their level of realization, it is not inconceivable that they might be changed back to their former implementations. We are waiting for experience with the system to either confirm our original decisions or indicate alternate strategies.

CONCLUSIONS

Those of us in Project Mu who have participated in the BSM design and implementation have become increasingly impressed with the power small systems equipped with microprogramming capabilities possess. As technology continues to lower the cost of these systems we feel they will become even more attractive due to the sophistication they are capable of supporting and the ability to adapt to the needs of an application. It is these features of today's mini-computers which make them very attractive and contribute heavily to their growing power and potential.

ACKNOWLEDGMENTS

The author is pleased to acknowledge the help of all the faculty and students at SUNY/Buffalo who have participated in the BSM project and have helped make it a successful venture. Special thanks are due to Michael J. Manthey, who was the original driving force behind the effort and who contributed much of the original design, to Gideon Frieder and Robert Rosin, whose criticism, encouragement and guidance helped greatly in transforming rather abstract concepts into a coherent machine design, and to the National Science Foundation which helped sponsor this project through NSF Grant GJ-993.

REFERENCES

1. *B6500 Characteristics Manual*, Burroughs Corp., Detroit, Michigan, 1968.
2. *B6500 Master Control Program Reference Manual*, Burroughs Corp., Detroit, Michigan, 1969.

3. *B1700 Systems Reference Manual*, Burroughs Corp., Detroit, Michigan, 1972.
4. Dijkstra, E. W. "Cooperating Sequential Processes," *Programming Languages*, edited by F. Genuys, Academic Press, London, 1968.
5. Hauck, E. A., Dent, B. A., "The Burroughs B6500/7500 Stack Mechanism," *Proceedings of the Spring Joint Computer Conference*, 1968.
6. Iliffe, J. K., *Basic Machine Principles*, American Elsevier, New York, 1968.
7. Iliffe, J. K., "Elements of BLM," *The Computer Journal*, August, 1969.
8. Johnston, J. B., "The Contour Model of Block Structured Processes." *Proceedings of the SIGPLAN Symposium on Data Structures in Programming Languages*, 1971.
9. Lutz, M. J. and Manthey, M. J., "A Microprogrammed Implementation of a Block Structured Architecture." *Fifth Annual Workshop on Microprogramming*, (Preprints), 1972.
10. Manthey, M. J. and Lutz, M. J., *An Overview of the Buffalo Stack Machine Architecture*," Department of Computer Science, State University of New York at Buffalo. Report # 31-72-MU.
11. McKeeman, W. M., "Language Directed Computer Design," *Proceedings of the Fall Joint Computer Conference*, 1967.
12. McKeeman, W. M. et al., *A Compiler Generator*. Prentice-Hall, Englewood Cliffs, N.J., 1970.
13. Naur, P. et al., "Revised Report on the Algorithmic Language ALGOL 60," *Communications of the ACM*, January, 1963.
14. Organick, E. I., Cleary, J. G., "A Data Structure Model of the B6700 Computer System," *Proceedings of the SIGPLAN Symposium on Data Structures in Programming Languages*, 1971.
15. Saal, H. J., Shustek, L. J., "Microprogrammed Implementation of Computer Measurement Techniques," *Fifth Annual Workshop on Microprogramming*, (Preprints), 1972.

Operating system design considerations for microprogrammed mini-computer satellite systems*

by J. E. STOCKENBERG, P. C. ANAGNOSTOPOULOS, R. E. JOHNSON, R. G. MUNCK, G. M. STABLER, and A. VAN DAM

Brown University
Providence, Rhode Island

INTRODUCTION

The operating system described in this paper was developed as part of research sponsored by The National Science Foundation and the Office of Naval Research on satellite processing and symbolic debugging of data structures. This system runs on a small (32K bytes), dual processor, microprogrammable computer equipped with a high-speed graphic display unit and attached in satellite mode to the multiplexor channel of an IBM System/360-67.

In addition to providing the support for these major research projects, the system is intended for use by a variety of applications, both stand-alone and satellite. To support these varied uses an operating system has been designed which attempts to tailor many features introduced in other, usually larger, systems to the particular needs of these applications, and at the same time introduce new features that better meet the research requirements.

The system, which has been in use since August 1972, is implemented in "levels of abstraction" as defined by Dijkstra.[5] Firmware has been included to support the implementation of these levels and to provide automatic storage allocation and status-saving upon program entry. The lowest level of the operating system creates an extended machine which provides an environment that supports multiprogramming. This environment is similar in concept to that described by Hansen.[6] Communication between the levels of the system is provided by the introduction of a data structure called the event table.

CONFIGURATION

The Brown University Graphics System (BUGS)[10] configuration consists of dual microprogrammable Digital Scientific Corporation META 4 processors, sharing 32K bytes of memory. The 'A' processor, which has been microprogrammed as a general purpose computer,[2] is attached to the multiplexor channel of an S/360-67 and has attached to it a card reader, a keyboard/console, and a megabyte removable cartridge disk. The 'B' processor is being microprogrammed with a subset of the general purpose instructions of the 'A' processor and a set of high level graphic display instructions.[11] The 'B' has attached to it a high-speed, high-resolution Vector General CRT. The Vector General is equipped with a joystick, lightpen, function keys, control dials, and a keyboard. A small, very fast (less than 30 nanosecond instruction time) special purpose microprogrammable processor will also be attached to the 'B'. This processor, which uses a writeable control store, can be dynamically set up to perform a variety of matrix and vector calculations including rotation, translation, perspective, windowing, scaling and clipping. The operating system being described runs primarily on the 'A' processor.

SPECIFIC GOALS

A simple general purpose operating system for use with predominately graphical applications

BUGS is intended to be a small general-purpose computing system to be used predominately by graphics applications. In general, graphics applications do not make any unusual demands on the system under which they are running. Graphics applications probably do more data structure manipulation and accessing, as well as more character manipulation, than a "typical" non-graphics application. However, in this respect they are not very different from a compiler. Since graphics applications tend to be interrupt-driven, they will run better and be easier to develop if the system has an efficient and easily interfaced interrupt mechanism. In this respect they are similar to heavily I/O-oriented applications. So while it is true that graphics applications are not really unusual, they do have a combination of characteristics that influenced the design of BUGS.

Investigation of the advantages offered by microprogramming

A major goal of this project is to investigate what advantages the flexibility of microprogramming offers to

* This work is sponsored in part by The National Science Foundation, Grant GJ-28401X, The Office of Naval Research, Contract N000-14-67-A-0191-0023, and the Brown University Division of Applied Mathematics.

both system and application programmers. The effects being examined concern program speed and size as well as certain harder to evaluate aspects such as programming and debugging ease. To facilitate experimentation in this area the operating system has been designed to allow operating system functions or parts of these functions to be moved from the software to the firmware with a minimum of change to the operating system and with no impact on user programs.

HARDWARE (FIRMWARE) CONSIDERATIONS

S/360-Like instruction set

The lowest level of BUGS is that presented by the firmware. A detailed description of the firmware implementation can be found in the META 4A Principles of Operation.[1] Basically, the firmware-defined architecture is similar to that of the IBM System/360. Information is stored in main memory in individually-addressable eight-bit units called "bytes". Bytes may be accessed separately or grouped together. The most common grouping consists of two bytes and is called a "halfword". The sixteen registers and the fixed-point numbers operated upon by arithmetic instructions are halfwords.

Of the sixteen registers, three are special purpose, the rest general purpose. Register 0, called the Machine Status Register (MSR), contains information required for proper program control and execution. Register 1 is the Program Counter (PC). Register 15 is the Stack Frame Pointer (SFP). Its use is described in a later section. The system has no floating-point or decimal support.

Improvements over 360 instruction set

The system has S/360-like storage-to-storage instructions for handling variable length character strings. In addition to the standard S/360 version in which the string length is included explicitly in the instruction, each storage-to-storage instruction has a form in which the length is placed in a register and the register is specified in the instruction. This allows the handling of character strings up to 32K bytes in length.

To provide enhanced data structure and character-string manipulation facilities several instructions have been added to the standard S/360 set. These include a Search instruction for processing linked lists or tables for a key which holds some relation to a search argument. Enqueue and Dequeue instructions for manipulating the elements of linked lists are included for use in conjunction with the Search instruction.

The standard S/360 Translate and Translate-and-Test instructions are provided, as well as a Translate-and-Test which scans from right to left. Several additional scan instructions are included for scanning left or right for equality or inequality with a specified character. For manipulating pushdown stacks, Push and Pop instructions are provided which can be used for stacking registers or storage.

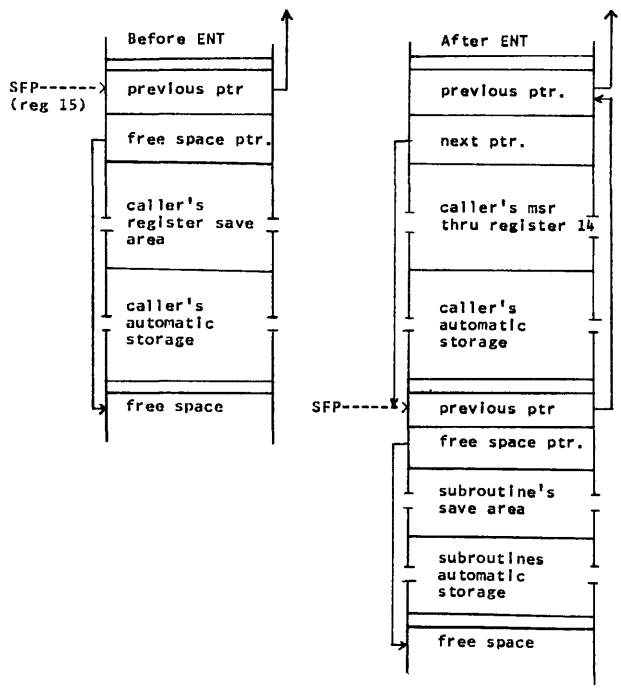
The machine includes other significant non-S/360 features. A group of instructions has been added which includes 16 bits of immediate data. Since this is the size of the registers and the fixed-point data items, these instructions are very useful and allow a four-byte instruction to perform the same function that would take six or eight bytes on an S/360.

360-Like PSW swap interrupts

The BUGS system has an S/360-like interrupt mechanism with the equivalent PSW swaps. There are four types of interrupts on the 'A' processor: SuperVisor Call (SVC), Program, Local I/O (which includes the 'B' processor), and those generated by the S/360.

Stack frames

Probably the most significant departure from standard S/360 architecture in BUGS is the use of stacks for subroutine calls. These stacks are similar in function to the Multics stack segment.[4] Programs which require the saving of registers and optional dynamic work storage can have this service performed by executing an ENT (ENTer routine) as the first instruction. Upon execution of this ENT instruction the stack is updated in the following manner:



The first figure shows only the caller's stack frame. The second shows both the caller's and subroutine's stack frames. During the execution of his program, the user must not modify the SFP. The amount of dynamic storage needed by the routine is specified as an immediate field in the ENT instruction. To return to the calling rou-

tine, a program issues a RET (RETurn from routine) instruction. This instruction restores the stack to its previous state and returns control to the calling program with all registers restored. The firmware controls only the allocation of storage within the stack. If, during the execution of the ENT instruction, the firmware determines that insufficient space remains in the stack to perform the required allocation, a program interrupt is generated and the operating system allocates a stack extension and completes the ENT processing.

I/O COMMANDS

Firmware support for the local I/O devices and the S/360 is very low level. A single command can be sent to a specified device by use of the Start I/O (SIO) instruction. Typical commands allow for the writing of a single character to the console, moving the disk arm a specified distance and direction, or the reading of a sector of information from the disk. The execution of most commands is performed asynchronously by the specified device. The completion of these commands causes an I/O interrupt.

BASIC CONCEPTS

Multi-level operating system

The specific goals outlined in the third section of this paper have led to a number of basic design conceptions. The first concept is motivated by a desire to produce a system in which the interfaces between the various parts of the operating system and between the user and the operating system are as precise and well-defined as possible. The need for such well-defined interfaces arises from the fact that the operating system is basically a research tool and will be continually changing. The system has been designed and implemented as a hierarchy of levels to minimize the effects of these anticipated changes.

The immediate effect of this design is to blur the boundary between firmware and software. The program on LEVEL1 "sees" only hardware below, with the characteristics of that hardware being simulated by LEVEL0 and the actual firmware. It is therefore possible to test the usefulness of a function by implementing it in LEVEL0, and later moving it into the firmware if that is found appropriate. The LEVEL1 program will not notice the change except in terms of the speed of the function. Thus the "extended machine" concept and microprogramming together provide an ideal environment for research into exactly what a "useful" machine for satellite graphics (or many other things) should look like. This is a significantly different approach from that taken in the Venus Operating System,[8] in which the initial (non-modifiable) firmware machine included approximately the same level of support that the BUGS operating system has incorporated initially in LEVEL0. Rather than viewing the development of the firmware defined architecture as a step that must completely precede the development of an operating system, the BUGS project is attempting to overlap these two steps as much as possible.

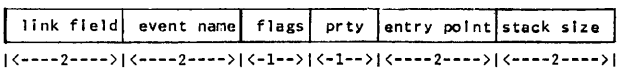
The operating system consists of three levels. Lower levels are completely independent of the data structures and facilities of higher levels. Higher levels access the data structures of lower levels only through the use of the lower level routines.

To facilitate the implementation of the lowest level (LEVEL0) of the operating system, the firmware does instruction parse and effective address computation prior to recognition of an illegal op-code. For illegal instructions, the firmware dumps the parsed fields and effective addresses into core, and initiates a program check interrupt. To utilize this feature, each of the functions of LEVEL0 is assigned an invalid instruction of the proper format. This instruction is coded and executed by higher levels of the system as if it were a valid machine instruction. The program check handler, part of LEVEL0, recognizes these instructions and invokes the proper LEVEL0 routine. Because the operands have already been computed (an activity that could take as long as the rest of the requested LEVEL0 function) these functions execute very efficiently. In addition, these functions can be placed in the firmware in the future with no effect on higher levels of the system. These LEVEL0 functions are called extended instructions.

In addition to making it easier to introduce changes into the system, it was expected that a multi-level system would be easier to implement and debug since the complexity of each separate level of the complete system is kept reasonably low. For the same reason it is easier to document the internals of such a system.

Event table

To meet the goal of providing good interrupt handling for graphics applications and also to provide a consistent interface between the various components of the system, a second concept, that of a special data structure called the event table, has been introduced. This structure is not really a table but rather sixteen separate unidirectional linked lists, many of which will be empty. The heads of these lists reside in fixed low-core locations. A typical element in the list will have the following format:



where

link field will point to the next element in the list
event name specifies an event or class of events
flags specifies the mode of execution of the event routine
prty specifies the priority of the event routine
entry point specifies the address of the event routine
stack size specifies the size of the initial stack for this event routine.

LEVEL0 uses the event table to provide a logical extension of the firmware interrupt mechanism to higher levels of the system. A more detailed description of this process is included in the next section of this paper.

All firmware interrupts cause LEVEL0 to gain control. Some interrupts, such as those caused by extended instructions, are handled completely by LEVEL0. Most interrupts, however, are reflected to higher level routines. This reflecting is done through the event table. In addition, the completion of asynchronous LEVEL0 functions is reflected through the event table. In this manner events represent interrupts from the extended machine (LEVEL0 plus firmware) to higher levels of the operating system. The event name can identify a specific "extended interrupt" or a class of extended interrupts.

Using facilities provided by LEVEL0, higher levels of the operating system and user programs build the event table. The parameter passing conventions and the environment created are identical for all events, creating one consistent interface between the system and the user. To aid in program checkout LEVEL0 can be made to invoke a specified event using parameters passed to it. This allows the creation of the exact environment that would exist if the actual extended interrupt had occurred. Since the actual scheduling of events occurs at only one point in LEVEL0 the tracing of system and user activity can be efficiently and easily accomplished.

Multiprogramming

The final basic design concept was that LEVEL0 should provide a flexible form of multiprogramming which would be fully utilized by higher levels. Because the system is designed for use by a single user or at most a few users, only very basic task coordination and communication facilities are provided by LEVEL0. Applications which require more sophisticated facilities can add this support to higher levels of the operating system, building on the functions of LEVEL0.

LEVEL0

LEVEL0 consists of an interrupt handler, a priority dispatcher, a group of extended instruction routines which provide storage management, a Wait/Post mechanism and a simplified I/O facility.

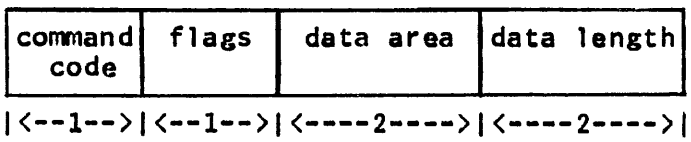
Storage management

Extended instructions are provided to obtain and release specified amounts of storage. In addition, an extended instruction is provided to obtain the largest block of free storage. Besides this explicit storage management, LEVEL0 allocates stacks whenever certain types of events are initiated and allocates a stack extension whenever ENT determines that there is insufficient space in the current stack. When RET determines that an entire stack extension is now free, LEVEL0 is notified and this storage is freed.

Extended I/O

In terms of the amount of storage occupied by code, the largest function in LEVEL0 is the extended I/O support for the S/360 and the local devices: disk, keyboard/typewriter, card reader, the programmer's panel, and the META 4B. This is due mainly to the very low level of the I/O commands accepted by the firmware.

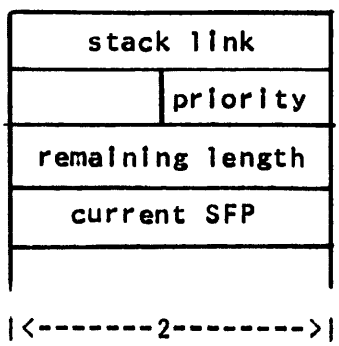
The LEVEL0 extended I/O support simulates an intelligent channel. With this support higher levels of the system can accomplish a logical unit of I/O work with a single request to LEVEL0. A LEVEL0 I/O request is initiated by use of an EXecute Channel Program (EXCP) extended instruction. The operands of the EXCP instruction are the device number and the address of the start of a channel program. The channel program consists of one or more Channel Program Commands (CPC). CPC's have the following format:



CPC's are logically equivalent to S/360 CCW's. The allowable command codes and their meanings vary from device to device. Upon completion of all processing required for a particular channel program, an extended interrupt is generated. The particular device and whether or not an error has occurred is reflected in the event name. Event naming conventions are described in a following section.

Priority dispatcher

The priority dispatcher allows for the execution of an essentially unlimited number of parallel tasks with priorities specified by the higher levels of the system. The only data structure that the dispatcher accesses is the stack. Each task in the system has its own stack. When the stack is created a header is attached with the following format:



This stack is then linked into the stack queue according to its priority. The head of this queue is at a fixed low-core location. The priority dispatcher searches this queue for the first stack that is not in wait state (e.g., one which contains an MSR without the wait bit set). The user's registers, MSR and PC, are loaded from this stack starting or restarting execution of the task associated with the selected stack. If all users are in wait state, the dispatcher loads the MSR, PC and registers from the last stack on the queue, placing the machine in wait state.

Extended interrupt generation/task creation

When LEVEL0 determines that an event has occurred which it does not handle internally, such as an SVC interrupt, completion of an I/O request, or an interrupt from the graphics processor, it uses information about the event to generate an event name. The event name consists of four hex digits. The first digit generally describes the type of event, e.g., 0 for a timer interrupt, 2 for a local I/O device completion, 'C' for an SVC interrupt. This digit also defines which of the sixteen lists is searched. The meaning of the three remaining digits varies depending on the first digit. For a local I/O completion the second digit is the device number, a third digit of 1 indicates successful completion, a 2 indicates an error, and for an error the last digit describes the type of error. For an SVC interrupt, the second digit is always zero; the third and fourth digits are the SVC number. After LEVEL0 has determined the proper event name, an extended interrupt is generated. This is done by searching the appropriate list in the event table for an entry with the specified event name. If an event entry is found for the specified name, the flag field in the event table element is checked. If this flag indicates that the event is to be processed in parallel with the other tasks running in the system, a stack is allocated for the new task. The size of this stack and its priority are obtained from the event table element. The stack is linked into the stack queue and initialized so that when it is dispatched, the PC will point at the entry point for the routine to handle the event as specified in the event list element.

Certain high priority LEVEL1 event routines do not wait for completion of I/O. (System performance would be degraded by allocating these routines their own stack since nothing is gained by executing them as parallel tasks.) Instead, they are allowed to execute immediately. They run using the stack of the routine that was executing when the event or interrupt occurred. These routines are run disabled and are allowed to use only a restricted set of system facilities.

A range of event names is reserved for applications programs. These programs are allowed to link in event list elements describing events that they wish to handle. These can be events that the system normally supports, such as the interrupt button on the panel; or they can be new events such as user SVC's. Also, by adding an event table element with a user-specified name (with the entry point specifying the proper routine and with the event flag set to indicate parallel execution) an application program can initiate the routine as a parallel task by executing a 'signal event' extended instruction.

The event names are defined, and the event table searched, in a manner that allows a single event table entry to specify a class of events to be handled. If LEVEL0 fails to find an entry in the event table for a particular event that has occurred, the rightmost hex digit of the event name is zeroed temporarily, and the event table is searched again. This process is repeated up to four times, or until a match is found. It is because of this search technique that the event names are defined in the previously described fashion. In addition, no event names are defined with 'trailing' zeroes. The fourth search of the event table, if necessary, is always performed with an event name of X'0000'. LEVEL1 provides an event table entry for this event name. The entry point specifies the LEVEL1 supplied DEBUG package.

The result of all this is that the user can have both specific and general interrupt handlers, such as specific routines for function keys 5, 9, and 12 (entries X'nnn5', X'nnn9', and X'nnnC' in the event table) and a general routine for all other function keys (X'nnn0').

In discussing the creation of parallel tasks, the use of the term sub-tasks was specifically avoided. In this system all tasks are independent and equal. No information is retained concerning which tasks initiated which other tasks. In this way, a task's life is not bound by that of its initiator, nor is its allowable priority range affected by that of its initiator. This does allow for more flexibility, but also requires that the programmer who utilizes multitasking provide his own means of removing unwanted tasks should the need arise.

Wait/Post

In order to coordinate the execution of tasks within this system, Wait and Post task synchronization primitives have been included in LEVEL0 as extended instructions. Both Wait and Post refer to a Wait Control Halfword (WCH). When a Wait is issued, LEVEL0 inspects the specified WCH; if the Post bit (the high order bit) is a '1', control is returned to the user and execution continues. If the Post bit is '0', then LEVEL0 stores the address of the stack header of the user in the WCH and turns on the Wait bit in the MSR in his stack frame. The dispatcher is given control to find another task to execute. When a Post is issued, LEVEL0 examines the addressed WCH to see if it contains a pointer to a stack header. If it does, LEVEL0 finds the MSR in the current stack frame and turns off the Wait bit. LEVEL0 then turns on the Post bit in the WCH and gives the dispatcher control. If LEVEL0 does not find a stack header address in the WCH, it just turns on the Post bit and returns.

LEVEL1

Introduction

The LEVEL1 routines, in combination with LEVEL0 and through the use of LEVEL0, are intended to provide an environment for execution of application programs. Most of the application programmers also use the Cambridge Monitor System (CMS)[7] which runs on Brown's IBM System/360-67. These users are therefore accustomed to the facilities offered by CMS. Because of this, LEVEL1 provides 'CMS-like' support for the local I/O devices. LEVEL1 performs file management and space management for the local disk. LEVEL1 provides a linking loader and a command processing routine to initiate the execution of user programs and LEVEL2 utility routines. An automatically invoked debugging routine is also provided.

The LEVEL1 routines are invoked by issuing an SVC. All of these routines are passed a parameter list in which the first halfword is a WCH. The WCH is posted by the invoked routine when it completes the requested function. This Posting is necessary since these LEVEL1 routines run as parallel tasks after being invoked. This parallel execution allows the maximum overlap between the LEVEL1 routines, which normally perform I/O, and the user program. If the user wants to suspend execution until the LEVEL1 request is complete he must issue a WAIT on the specified WCH.

The LEVEL1 being described represents the standard or full LEVEL1. At IPL-time the user can specify that an alternate LEVEL1 should be loaded. This LEVEL1 may consist of a subset of the standard LEVEL1 or a completely different, user written, program.

"CMS-Like" I/O support

For the card reader the RDCARD routine provides the user with up to 80 characters from the next card in the card reader. For the keyboard/console, the user can read a line (TYPEIN), type out a line (TYPEOUT), or type a line and read a reply (TYPEOUT, with reply option specified). Optional editing facilities are available.

RDBUF provides the user with a logical record from a specified file. Deblocking is provided automatically. The user specifies the logical record number so RDBUF can be used to read sequentially or randomly. WRBUF allows the user to write a logical record to an existing or new file. Again, blocking is performed automatically and the records can be written sequentially or randomly. A new file is allocated automatically when WRBUF is issued for a file whose file name is not found in the file directory. A file is extended automatically when a WRBUF specifies a record number beyond the current end of a file.

Disk management

LEVEL1 provides space management on the local disk through the use of an allocation map and a file directory. The allocation map contains a bit for each sector. This bit is on if the sector is allocated or defective. The file directory contains the file name and type and the starting sector number of each file on the disk. Additional information, including the logical record length and file length, is contained in the file header which occupies the first logical record of each file. LEVEL1 uses these data structures to allocate, delete, and extend files. The STATE routine can be used to obtain information from the file directory for a specified file.

Linking loader

The largest part of LEVEL1 is a linking loader. The input to the loader, called a module, is in the form of text decks stored on disk. These text decks are a compacted version of the standard OS/360 text deck. A text deck may represent the output of one or more compilations. The loader relocates the module into any available free storage. Cross-references between the CSECT's within a module are resolved.

Debugs

DEBUGS is invoked automatically when an event occurs for which there is no entry in the event table. The most common instance of this is a program check in the user program. When such an event occurs, DEBUGS informs the user of this fact through the keyboard/console. DEBUGS then provides the user with facilities to display and modify storage and registers, to set a breakpoint and/or origin, and to restart execution at an arbitrary point.

Command processing/parsing

The final function provided by LEVEL1 is command processing. User programs and LEVEL2 utility programs are stored on the disk in the module form. After the system has been IPL'ed and initialization is complete, the LEVEL1 command processing routine accepts a line from the console. The first field in the line typed-in specifies the file name of the user program or LEVEL2 program. This program is loaded and executed. The rest of the line is parsed and passed to this routine as parameters. When this program finishes, it returns to the command processor, which reads another line from the console and executes the next command.

LEVEL2 COMMANDS

The portion of BUGS that exists on LEVEL2 consists of a group of utility commands for file and directory manipulation and inspection. These commands are similar to a subset of the CMS command language. The

STATISTICS and LISTF commands can be used to inspect the allocation map, file directory, and file headers. The PRINTF and PRINTFX commands can be used to display files. The MODMAP command displays the addresses of CSECT beginnings and entry points in modules on the disk. ALTER and DELETE can be used for file directory management. MODZAP can be used to patch a module. LOADMOD can be used to load a module into storage without executing it. OFFLINE is used to read a file from the card reader and store it on the disk. PACK is used to reclaim non-contiguous free space on the disk.

EXTENSIONS

The final section of this paper deals with the major extensions to the operating system currently being designed and implemented. These extensions are designed to provide storage management facilities for programs which require more core storage than is actually available and to reduce the amount of storage permanently required by the operating system. The approach being taken is similar to that taken in the SYSEX[9] system.

Two types of storage are supported by these extensions. "Permanent" storage will be provided as it is currently done by LEVEL0. "Permanent" does not mean that the storage is not allocated and freed dynamically, but rather that, once allocated, the storage remains in a fixed location until freed by the user. The second type of storage is termed "dynamically relocatable". Such storage is allocated and managed by LEVEL0 when a higher-level routine requests the use of a record from a file or temporary work space which meets the conventions established for this type of storage. The file record may be a program or a data item. The program or data that occupies this storage may not contain any absolute pointers to itself or other dynamic storage. Also, these programs will not be allowed to modify themselves. At the time the higher level or user requests this storage, a base register is specified. LEVEL0 initializes the register for the user and records the assignment in its internal tables. Only limited modifications may be made to these registers.

With the user following these conventions it is possible for LEVEL0 to dynamically relocate programs and data either to reclaim fragmented free space or when bumped items are reloaded. This capability, which was pointed out in the early sixties by Corbato[C1], allows LEVEL0 to provide a form of virtual memory to higher levels.

Additional LEVEL0 facilities will be available to allow the user to provide additional information that LEVEL0 can use to determine what data items can be bumped when the need arises.

LEVEL1 disk support will be modified to use the new LEVEL0 support where applicable. This approach allows the programs that meet these conventions to be loaded with a simple disk read. The current LEVEL1 routines, including the loader, meet these requirements and can now be dynamically relocated, reducing the fixed storage requirements of the operating system.

CONCLUSIONS

The operating system has been in use by applications programmers since August 1972. Production usage was preceded by a 12 man/month design phase and an 18 man/month implementation phase. The implementation proceeded so smoothly that the operating system was useable a month ahead of schedule. It is felt that this is directly attributable to two basic design concepts, the more important being the division of the operating system into a hierarchy of levels and, to a lesser extent, the use of the Event Table as a canonical means of passing control between certain levels of the system.

Specifically, the division of the system into levels allowed each of these levels to be developed and debugged prior to its use by higher levels. This greatly reduced the overall debugging task. The Event Table, which was expected to ease the users programming effort, was felt to have provided much the same benefit to the systems programmers. In addition, it provided a convenient means of allowing access to debugging routines during the actual development of LEVEL1 and LEVEL2.

Much experimentation with the system is currently under way. This includes some of the projected experimentation in firmware implementation of operating system functions. So far, the work in this area has been limited to the addition or modification of instructions to improve certain common coding sequences.

The resident portion of the system, when a full LEVEL1 is included, occupies 8K bytes. Although, this is considered acceptable, a desire to reduce this figure is part of the motivation for the extensions outlined earlier.

The Event Table has proven very successful as a means of allowing the user to interface with the operating system. The extended interrupt mechanism using the Event Table meets the requirement of efficient interrupt handling. Less than .25 msec is required to process an interrupt and initiate the execution of the routine specified in the Event Table as a parallel task.

REFERENCES

1. Anagnostopoulos, P. C., Sockut, G. H., *Meta 4A Principles of Operation,* Brown University, Center for Computer and Information Sciences Technical Report.
2. Anagnostopoulos, P. C., Michel, M. J., Sockut, G. H., Stabler, G. M., van Dam, A., "Computer Architecture and Instruction Set Design", *Proceedings of the National Computer Conference and Exposition,* June 1973.
3. Corbato, F. J., *System Requirements for Multiple Access, Time-Shared Computers,* Project MAC Document MAC-TR-3.

4. Daley, R. C., Dennis, J. B., "Virtual Memory Processes, and Sharing in MULTICS", *Comm. ACM*, Vol. 11, No. 5, p. 306, May, 1968.
5. Dijkstra, E. W., "The Structure of the 'THE' Multiprogramming System", *Comm. ACM*, Vol. 11, No. 5, p. 341, May 1968.
6. Hansen, P. B., "The Nucleus of a Multi-Programming System", *Comm. ACM*, Vol. 13, No. 4, p. 238, Apr. 1970.
7. *IBM Control Program-67/Cambridge Monitor System, Version 3.1 User's Guide GH20-0859*, IBM Corp., 1971.
8. Liskov, B. H., "The Design of the Venus Operating System", *Comm. ACM*, Vol. 15, No. 3, p. 144, March 1972.
9. Reiter, A., "A Resource-Oriented Time-Sharing Monitor", *Software-Practice and Experience*, Vol. 2, No. 1, p. 55, January 1972.
10. Stabler, G. M., *Brown University Graphics System Overview*, Brown University, Center for Computer and Information Sciences Technical Report.
11. Stabler, G. M., *META 4B Principles of Operation*, Brown University, Center for Computer and Information Sciences Technical Report.

Another attempt to define computer science

by MICHEL A. MELKANOFF

Computer Science Department
University of California
Los Angeles, California

ABSTRACT

There are equally important scientific and engineering aspects to computer science; they may be described as follows:

SCIENTIFIC ASPECTS OF COMPUTER SCIENCE

The unique aim of the inductive sciences is to predict the future. This has been accomplished in the physical sciences by means of mathematico-symbolic models; the method has been much less successful in the behavioral and life sciences due to the difficulties of constructing soluble models which are valid over a sufficiently large domain. The advent of the computer has made it possible to construct and resolve models sufficiently complex to be interesting although their domain of validity is still limited: The program is here the model. To a large extent, the utility of the mathematical models (or theories) developed in the physical sciences is predicated on the possibility of developing mathematically generalized analytic solutions which permit predictions of very general types of events. This becomes proportionally more difficult as the model becomes more complicated. Thus even though the computer permits resolution of individual cases which otherwise would not be practically feasible, it cannot provide the power of an analytic solution. Thus the scientific aspects of computer science are similar to those of mathematics: they deal with all facets of the construction and especially the resolution of models embodied in this case by programs. The most fundamental problem is still to obtain (if at all possible) general analytic solutions to classes of algorithmic interactions described by programs and this may be considered as one of the most important long-range goals of computer science. In view of the enormous difficulties expected of such an endeavor, secondary goals must also be pursued; these include theoretical studies of program's schema, development and formal analysis of programming languages, development of more powerful computer systems, etc.

ENGINEERING ASPECTS OF COMPUTER SCIENCE

The engineering aspects of computer science are directly related to system design. Defining recursively a system as a collection of interacting objects or systems, we are particularly concerned with systems whose components interact through procedures embodied in computer programs. The aim of engineering is to design a system in such a fashion as to optimize a given criterion function subject to certain given constraints. Thus the design of complex systems where the computer plays a major part requires both extensive knowledge of computer science and of the discipline where the system is utilized. Since a computer system is itself a system of hardware and/or software, computer science design is doubly related to computer science: as the discipline wherein the system is utilized and as the discipline necessary to carry out the design.

The masters degree program in computer science

by M. A. MELKANOFF

University of California
Los Angeles, California

and

B. H. BARNES and G. L. ENGEL

Pennsylvania State University
University Park, Pennsylvania

ABSTRACT

The Curriculum Committee on Computer Science (C^3S) of the Association for Computing Machinery (ACM) is involved in a study of the masters degree curriculum. As visualized by the working group, such programs have two basic objectives:

(a) Providing the student with training in Computer Science preparing him for major positions in the industry.
(b) Testing and preparing the student for more advanced work leading to a doctorate.

Though it is not the intention of the Committee to provide a guide to the formation of a graduate program in Computer Science, it is recognized that many new programs are being developed, and thus it is hoped that the final guidelines will serve as a standard on which the various programs can be measured. To this time efforts have been concentrated on programs directed to the above objectives. It should be recognized that the question of special masters programs such as those for teachers, are not now under consideration.

The structure of the proposed program will be presented, as well as the underlying philosophy of the program. The proposal has been reviewed in several meetings of C^3S, and by several departments now offering masters degrees in Computer Science.

The comments and suggestions of the various Committee members and interested professionals will be summarized, and the program will be compared with existing masters degree programs.

This presentation will be of a working document. One of the motivations for presentation of the proposal is to elicit further comments and suggestions from those attending the National Computer Conference.

A homophonic cipher for computational cryptography*

by FRED A. STAHL

University of Illinois at Urbana-Champaign
Urbana, Illinois

INTRODUCTION

Computational cryptography deals with the storage and processing of sensitive information in computer systems by enciphering. Sensitive information is information that for one reason or another must be protected from being disclosed to individuals without proper authorization. The need for systems to be secure from unauthorized access to sensitive information has been well documented.[1-7] Cyrptographic techniques appear to be one of the most simple and secure methods of providing this much needed protection.

In the next section the requirements for computational cryptography will be discussed, especially with regard to the differences from communication cryptography; second, a short review of the cryptographic techniques that have been suggested for computational cryptography; third, a review of some standard cryptanalysis techniques; fourth, a homophonic cipher will be described; and finally, some additional problems associated with computational cryptography will be discussed.

REQUIREMENTS FOR COMPUTATIONAL CRYPTOGRAPHY

Cryptography has been used for ages as a technique for concealing the informational contents of messages. The use of cryptanalytic techniques combined with the availability of high speed electronic equipment have made both encipherment and cryptanalysis very sophisticated endeavors indeed. For this reason, simple ciphers can no longer be used to provide a great deal of security. This carries over to computational cryptography also. Moreover, there are a number of desirable features that any ciphers to be used in computer systems should have. These include:

1. Encipherment and decipherment should be simple. That is, the computational complexity of enciphering and deciphering information should be minimal.
2. The effective ability to not be broken should be high. As the sensitivity of the information increases a cipher that is more difficult to break should be available for use.
3. The cipher should be independent of message length. Messages can be even one character in length.
4. Small errors should not cause large losses of information. If a small amount is ciphered incorrectly it should not make a large block of information undecipherable.
5. Information should remain integral through editing of the ciphered text. The cipher should be such that the normal editing functions of deleting, inserting and moving strings of information (such as would occur in a dynamic data-base) can be performed without destroying the information contained in the ciphered text.
6. The cipher should not increase the length of the message excessively.
7. The cipher should be of such a nature that security can be maintained even though the cryptanalyst knows the ciphering system but not the key.

There are in addition to these desirable features a number of problems peculiar to computational cryptography. These will be discussed in the last section.

CRYPTOGRAPHIC TECHNIQUES THAT HAVE BEEN SUGGESTED FOR COMPUTATIONAL CRYPTOGRAPHY

The major thrust, so far, in computational cryptography has been to provide a cipher key to the computer and let it encipher information that enters the system, and also decipher and reencipher information as it is processed by the machine. The ciphers suggested include: simple substitution schemes, arithmetic schemes (i.e., adding or multiplying by a constant, or base conversion), logical schemes (e.g., exclusive-or), and transposition schemes. These have been described quite thoroughly by Krishnamurthy,[8] and Van Tassel.[9,10]

Some of these ciphers are too easily broken by the most elementary cryptanalytic techniques and others are too computationally complex to be implemented for use in computers. All fail to have some of the desirable features listed above.

* The work presented here was performed at the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign. It was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy, U.S. Air Force) under Contract No. DAAB-07-72-C-0259.

STANDARD CRYPTANALYTIC TECHNIQUES

It is not the purpose here to describe in great detail the cryptanalytic techniques that can be used to break a cipher. The reader interested in this topic is referred to Gaines.[11] It is only intended to suggest what types of information remain visible after the original text has been enciphered.

For the most part cryptanalytic clues are gained from the information inherent in the structure of the language such as frequency information which is extremely hard to remove. Simple substitution, for example, leaves for easy analysis all single letter, and multiple letter frequencies, doublet and reversal frequencies, as well as contact variety information. Figure 1 is representative of this information for a standard text.

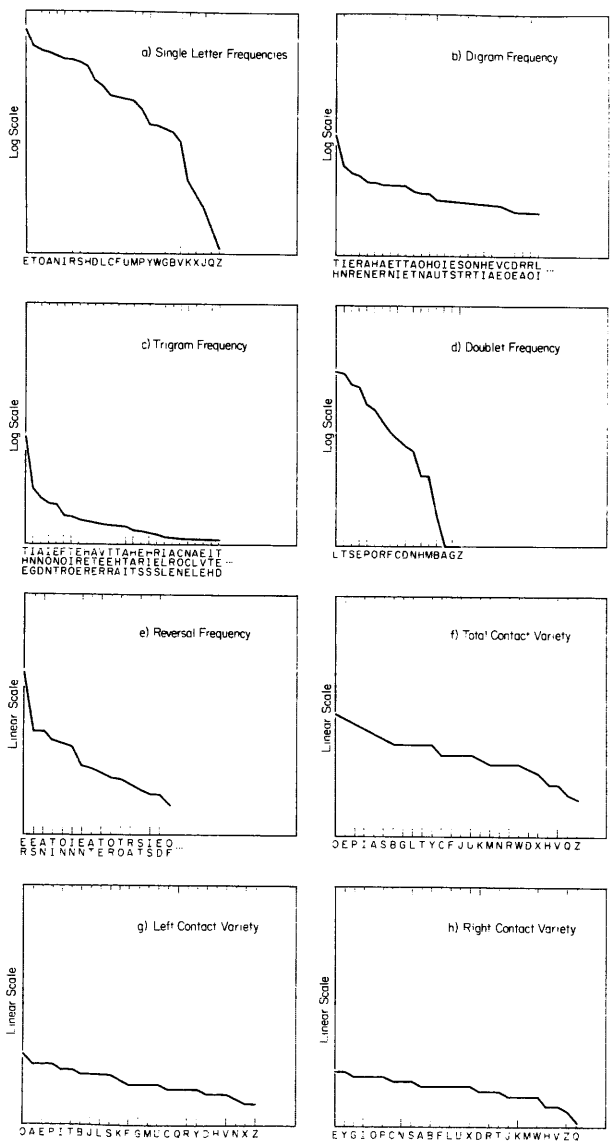


Figure 1—Various frequency distributions for English

HOMOPHONIC CIPHERS

A homophonic cipher is a substitution cipher in which a given character may have any of a number of different representations. Figure 2 gives one such cipher and a sample message using it. Note that the cipher-text for E, for instance, varies from substitution to substitution. Kahn[12] notes that the first known Western instance occurs in a cipher that the Duchy of Mantua prepared in 1401 for correspondence with one Simone de Crema. Each of the plaintext vowels has several possible equivalents.... "That the homophones were applied to vowels, and not just indiscriminately, indicates a knowledge of at least the outlines of frequency analysis."

Obviously, the more ciphertext symbols relative to plaintext symbols the easier it is to disguise the structural properties of the plaintext through enciphering. Each plaintext symbol could have many ciphertext encipherings. In order to illustrate this consider a plaintext alphabet of 26 symbols and a ciphertext alphabet of 1024 symbols (10 bits). Initially each plaintext symbol would have as many ciphers directly proportional to its frequency in the language (see Figure 3). Notice that the single letter frequency of the ciphertext is nearly constant (compare Figure 4 to Figure 1a) relative to its frequency in the plaintext thereby not providing any single letter frequency information for the cryptanalyst.

However, as noted above, the cryptanalyst uses other techniques for breaking codes. The cryptanalyst looks for the most deviant structural features first. Consider, for example, the abnormal reversal frequency of ER in Figure 1e. If this reversal can be located in the ciphertext the cryptanalyst is much closer to breaking the code. In contrast, he would not look for the reversal OF since there are many other reversals with nearly the same frequency occurrence. The crucial point to remember is that it is only the abnormal or deviant frequencies that can be used for clues. Clearly, if all measurements of the ciphertext yielded nearly flat distributions as in Figure 4 there would be no information gained from those measurements.

Key

Plaintext:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Ciphertext:	C R Y P T O G R A M 5 6 7 8 9 B D E F H I J K L N Q
	1 2 3 4 S
	U V W X
	Z

Message

Plaintext:	THIS IS A SECRET MESSAGE
Ciphertext:	HRAF AF C FTYE2H 7VFF1GZ

FP-3445

Figure 2—A simple homophonic substitution cipher

Plaintext Symbol:	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of Cipher Symbols:	81	13	31	43	133	29	14	61	71	2	5	38	27

	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	15	85	22	2	70	65	93	28	10	15	3	15	1

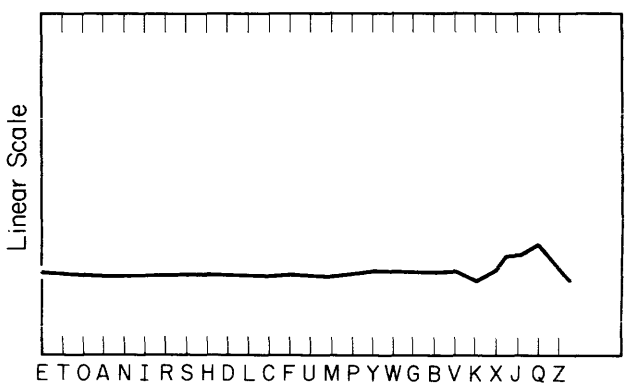
FP-3448

Figure 3—A homophonic substitution cipher generated in direct proportion to frequency in plaintext

Now let us generalize the homophonic technique to flatten the other curves illustrated in Figure 1. This is easier said than accomplished, since adjusting one curve to be flatter will generally result in making another one more curved.

Let us return to the earlier generalized homophonic enciphering hypothesis and modify it somewhat to allow for more flexibility. Before we wanted the curve resulting from single letter enciphering to be nearly flat. If we lessen this restriction somewhat and allow for some distribution but not nearly as much we can avoid the abnormal or deviant frequencies that are normally used for clues on all the measurements found in Figure 1. Figure 5 gives one such homophonic cipher.

The technique used to generate this code is fairly simple. The frequencies are adjusted as for the code in Figure 3, and the corresponding frequency charts can be generated for the other measurement of the ciphertext. Next, the most deviant frequency in any measurement is examined. If, for example, it is a reversal frequency the corresponding number of cipher symbols used for each of the constituent symbols is raised accordingly. The process is repeated until the frequency curves are satisfactory. If there is no convergence the process can be started over again taking care to choose different measurements first. If after a number of attempts are made appropriate curves cannot be gotten, it might be necessary to increase the number of ciphertext symbols by increasing the number of bits used to represent each symbol.



FP-3449

Figure 4—Single cipher frequency for homophonic cipher

Plaintext Symbol:	A	B	C	D	E	F	G	H	I	J	K	L	M
Number of Cipher Symbols:	73	10	24	34	129	23	10	52	77	2	4	43	22

	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	27	89	19	12	80	72	87	45	8	13	2	13	2

FP-3450

Figure 5—A generalized homophonic cipher

A cipher successfully generated in the manner described has all the desirable features for computational cryptography set forth above. An extremely simple enciphering and deciphering scheme can be used since this is a substitution type cipher. A sample scheme will be given. With regard to the ability to adjust the cipher to meet security needs only increase the number of bits used to represent each ciphertext symbol (effectively, increasing the ciphertext alphabet). Since there is one ciphertext symbol for each plaintext symbol the messages can be of arbitrary length. An error in one symbol does not extend errors even to adjacent symbols of the message; thereby keeping losses of information to a minimum. Note in particular that since it is a substitution cipher the 'integrity through editing' condition is met. That is, strings (including individual characters) of enciphered message may be moved with respect to each other without going through a deciphering and reenciphering process. This property makes it invaluable for large dynamic databases.

The length of the message only increases with the security needed. For a typical low security cipher 8-bits should be sufficient for a 64 symbol plaintext alphabet. A homophonic cipher can effectively destroy all standard language frequency information as shown in Figure 1. In addition, information in ciphered form may be received by the computer from a terminal and be edited without it ever being deciphered at the central facility. As mentioned earlier the device to encipher the plaintext message need not be very complex. Consider a key of 256 characters; each of the 64 characters appears in the key the number of times desired for the particular application (see Figure 6). The key is loaded into a 256 word memory. Deciphering consists of returning the contents of the address specified by the 8-bit cipher. Enciphering involves generating an address randomly and then searching sequentially until a matching character is found and then transmitting its address (see Figure 7).

The amount of secrecy needed can be controlled by the number of bits. So, for instance with 9 bits there would be 418 remaining bit patterns; with 10 bits it would be 984, etc. Each additional bit increases the security.

ADDITIONAL PROBLEMS ASSOCIATED WITH COMPUTATIONAL CRYPTOGRAPHY

There are a number of additional problems associated with keeping sensitive information secure in computer systems. These include the following:

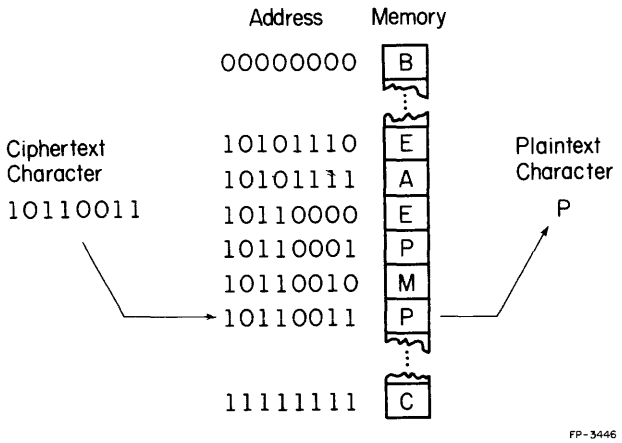


Figure 6—Deciphering the homophonic cipher

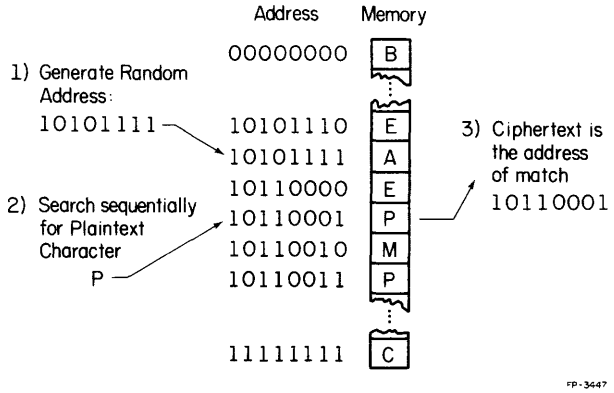


Figure 7—Enciphering the homophonic cipher

1. *The decoded message problem.* This comes about when a block of decoded or unenciphered message is known by the analyst. With this type of information available very few cipher systems are safe.
2. *The limited syntax problem.* When dealing with limited languages such as programming languages the analyst can break the cipher by knowing the restrictive properties of the language involved.
3. *The arithmetic problem.* If the ciphered text is not to be decoded inside the computer arithmetic operations cannot be performed.
4. *The overlapping access problem.* When different individuals have access to the same ciphered data and yet do not want common access to other enciphered data.

It is hoped that this paper will influence in some way the designers of future computer systems by showing that simple techniques can be used for effectively limiting the access of information to only those who should have access to it. A justification of the effectiveness of this homophonic cipher system on a mathematical basis using techniques developed by Shannon[13] will be described in a subsequent paper.

ACKNOWLEDGMENTS

The author wishes to thank Chung Laung Liu and James Studier for their suggestions and criticisms of the manuscript during its preparation and Greg Michael for his program to compute the various frequencies for the figures that appear.

REFERENCES

1. Ware, W. H., "Security and privacy in computer systems" *Proceedings Spring Joint Computer Conference*, pp. 279-282, 1967.
2. Ware, W. H., "Security and Privacy - Similarities and Differences", *Proceedings Spring Joint Computer Conference*, pp. 287-290, 1967.
3. Peters, B., "Security Considerations in a Multi-Programmed Computer System", *Proceedings Spring Joint Computer Conference*, pp. 283-286, 1967.
4. Petersen, H. E., Turn, R., "Systems Implications of Information Privacy", *Proceedings Spring Joint Computer Conference*, pp. 291-300, 1967.
5. *The Computer and Invasion of Privacy*, hearings before a Subcommittee on Government Operations, House of Representatives, July 26-28, 1966, Government Printing Office.
6. Harrison, A., *The Problem of Privacy in the Computer Age - An Annotated Bibliography*, Rand Corporation Memorandum RM=5495-PR/RC, December 1967.
7. Harrison, A., *The Problem of Privacy in the Computer Age - An Annotated Bibliography Vol. 2*, Rand Corporation Memorandum RM=5491/1-PR/RC, December 1969.
8. Krishnamurthy, E. V., "Computer Cryptographic Techniques for Processing and Storage of Confidential Information", *International Journal of Control*, Vol. 12, No. 5, pp. 753-761.
9. Van Tassel, D. L., "Cryptographic Techniques for Computers", *Proceedings Spring Joint Computer Conference*, pp. 367-372, 1969.
10. Van Tassel, D. L., "Advanced Cryptographic Techniques for Computers", *Communications of the ACM*, Vol. 12, No. 12, December 1969, pp. 664-665.
11. Gaines, H. F., *Cryptanalysis*, Dover, 1956.
12. Kahn, D., *The Codebreakers*, Macmillan, 1967.
13. Shannon, C. E., "Communication Theory of Secrecy Systems", *Bell System Technical Journal*, Vol. 28, No. 4, October 1949, pp. 656-715.

Cryptology, computers, and common sense

by G. E. MELLEN

Sperry Univac
St. Paul, Minnesota

INTRODUCTION

With that as title, the writer ought give at once the meaning of the final term. Here, "common sense" is used with double intent.

First, it is a caveat to the reader that a discourse on computers and cryptology in the open literature is like a "layman's guide to worldwide espionage." It simply cannot be done. Too much is unknown. Too much (because cryptographic matters are exempt from automatic declassification) will never be known.

To the extent, then, that the author of such a paper requires a degree of *chutzpah* to attempt it, to the same extent he may ask of his readers an apprehension of the difficulties involved, and the forebearance not to make harsh judgment of sometimes unavoidable shortcomings.

The second meaning in which "common sense" is used alludes to the opinion that cryptography, as it pertains to the needs of many commercial users, is perhaps becoming "oversold," resulting occasionally in needless expense, operational difficulties, and a false sense of security. The defense of this thesis is deferred to later parts of the paper.

SOME DEFINITIONS

Data security is that technology, the objective of which is to prevent the interception of data, whether by wiretapping, masquerading, trap-doors, or any of the many clandestine tools of the "enemy" (see below). Data security is a technology in itself, and is not dealt with in this paper.

Cryptology and *cryptography* are near-synonyms, the former being somewhat wider in scope. *Cryptography* pertains to the means used by the originator of data to prevent the message, once intercepted, from being understood by an enemy. This is the process of *encryption* or *encryptment*, its cryptographic complement, *decryption* or *decryptment*, being the process used by the intended addressee to decipher and read the message.

The over-all algorithm for encryption and decryption is the *cryptosystem*, while the *key* refers to those unique parameters employed in a specific application of the algorithm.

The *enemy* is any person or organization who takes positive action to intercept and decrypt data to which he is not legitimately entitled. "Enemy" thus has a different meaning here than in a military context, the connotation of violence being absent.

Having intercepted encrypted data, the enemy's principal tool is *cryptanalysis*. "Enemy," "cryptanalyst," or simply "analyst" are used synonymously throughout the paper.

A *code* is distinguished from a *cipher* in that the former employs a compact group of five letters (a "pentagram"—a legacy of the Morse telegraph era) to represent a message of any desired length. For example, ALOHA might mean "Shipment will be made Wednesday night." In a *cipher*, each element of the original message *(plaintext; pt)* has a counterpart in the encrypted version *(ciphertext; ct)*. CPPLLFFQFS might represent b o o k k e e p e r. Computer-oriented readers may appreciate the analogy that a code is to a cipher what a FORTRAN statement is to the corresponding function in assembly language. The discussion in this paper is limited to ciphers.

The paper uses the following conventions: Plaintext is represented by l e t t e r s p a c i n g. Ciphertext appears in CAPITALS. The key, if litteral and not numerical, is in *italics* (underscored in figures). The conventions are useful when p l a i n t e x t is added to p l a i n t e x t to produce *key*.

Another convention is the use of the English alphabet in the examples. The reader will understand that the underlying principles apply, *mutatis mutandis*, to any alphabet whatsoever, from the 12-letter Hawaiian alphabet, to the 256-character ASCII set, to the *n*-letter alphabet of the reader's own invention.

TRADITIONAL CRYPTOGRAPHY

Selected bibliography

For security reasons the bibliography of cryptography is predictably meager. There is only one comprehensive tutorial work in English for the pre-computer period, Gaines' *Elementary Cryptanalysis*[1] Occasional tutorial articles appear in *The Cryptogram*,[2] the periodical of the

American Cryptogram Association. Specialized technical discussions of cryptographic methods are contained in *The Broken Seal*[3] and in *The Shakespearean Ciphers Examined.*[4]

For the post-computer age, an outstanding tutorial work has been published by Sinkov,[5] a master of the craft. In 1967, a remarkable book, Kahn's *The Codebreakers,*[6] appeared. It is difficult to imagine a work on a subject as esoteric as cryptology being definitive, but within the confines of security, Kahn has succeeded. In addition to extensive historical coverage, he describes in sometimes surprising detail both past and current techniques of cryptography. Of special interest are the sections devoted to the cryptographic agencies and practices of the major powers, including the United States.

Basic techniques and some functional observations

The introduction of computers into cryptotechnology has affected the practice but not the underlying principles of the craft. It appears useful, therefore, to review certain of these principles in order later to show how they have evolved into the age of automation, and how, to an extent, they have carried some of their weaknesses with them.

In kernel, there are just two ways of converting plaintext to cipher-text, by substitution and by transposition. Regardless of its complexity, any cryptosystem can be shown to be either an elaboration of one of these methods or a combination of both.

Substitution types

Of the substitution types, the special-case Julius Caesar is the most familiar.* Here, each letter is replaced by the letter j places further along in the normal alphabet, where j is a constant. The substitution is performed modulo n, n being the number of letters in the alphabet. In the general case of simple substitution, any letter may replace any other letter, the substitution being invariant and usually with no letter representing itself.

Simple substitution is trivial. On occasion, though, even the trivial can breed insight. Under the rules of simple substitution, there are 25! possible "keys," or roughly 1.5×10^{25} possibilities. The magnitude of this number can be illustrated by a computer programmed to try one key each microsecond. At that rate the machine would take 1.7×10^{14} years to run through the list. Even the number needs explication. It is some 50,000 times longer than the estimated age of the Earth.

Yet most people can solve this sort of newspaper puzzle in a few minutes. Some can "sight-read" them, much as an accomplished pianist plays a piece of unfamiliar music.

The moral to be gained here is that one ought not be awed solely by large numbers. When a data-encryption device is promoted as having 10^{xx} different keys, the cautious system designer will repress the proclivity to regard this huge number as an unchallengeable figure of merit. It is not.

Simple substitution can demonstrate still another aspect of language. Consider the cipher word ABCDE. This word may be resolved into good English in more than 6000 ways (*e. g.*, b l a c k, g h o s t, etc.). The cipher word AABCA, however, may be resolved into English in one and only one way. There now arises an interesting question: What is the longest simple-substitution cipher which can be constructed, which can be resolved into English in one and only one way? The answer applies, in modified form, to the security of many kinds of ciphers.

Another form of pencil-and-paper substitution cipher is shown in Figure 1. Figure 1-A is an abbreviated version of the classic Vigenere tableau. Figure 1-B is the partial tableau which would be used for encrypting a message using the key *rogue* (resulting in a period of 5, since the same cipher alphabet comes back into play at every fifth pt letter). Figure 1-C shows the encryption, using *rogue* of the (specially chosen) plaintext;

```
pt:  a b c d e f g h i j k l m n o p q r s t u v w x y z

     A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
     B C D E F G H I J K L M N O P Q R S T U V W X Y Z A
     C D E F G H I J K L M N O P Q R S T U V W X Y Z A B
     . . . . . .
     Y Z A B C D E F G H I J K L M N O P Q R S T U V W X
     Z A B C D E F G H I J K L M N O P Q R S T U V W X Y
```

A. The Vignere Tableau. The "key letter" is the cipher letter under the plaintext a.

```
pt:  a b c d e f g h i j k l m n o p q r s t u v w x y z

     R S T U V W X Y Z A B C D E F G H I J K L M N O P Q
     O P Q R S T U V W X Y Z A B C D E F G H I J K L M N
     G H I J K L M N O P Q R S T U V W X Y Z A B C D E F
     U V W X Y Z A B C D E F G H I J K L M N O P Q R S T
     E F G H I J K L M N O P Q R S T U V W X Y Z A B C D
```

B. The partial tableau for the key rogue.

```
pt:   p e t e r  p i p e r  p i c k e d  a  p e c k  o f
key:  r o g u e  r o g u e  r o g u e r  o  g u e r  o g
ct:   G S Z Y V  G W V Y V  G W I E I U  O  V Y G B  C L
pt:   p i c k l e d  p e p p e r s
key:  u e r o g u e  r o g u e r o
ct:   J M T Y R Y H  G S V J I G
```

C. Encryption using the key rogue.

```
G S Z Y V
G W V Y V
G W I E I
U O V Y G
B C L J M
T Y R Y H
G S V J I
I G
```

D. Ciphertext set up by period for cryptanalysis.

Figure 1—Facets of Vigenere cryptography

* An interesting but unobtainable measure of the familiarity of the Caesar cipher would be the percentage of viewers who appreciated why, in Arthur C. Clarke's script for the motion picture, "2001: A Space Odyssey," the computer was named HAL.

peter piper picked a peck of pickled peppers. Figure 1-D indicates how the cryptanalyst, by determining the period (and also having sufficient ciphertext—a matter to be discussed later), can arrange the ciphertext so as to permit the recovery of the plaintext.*

The Vigenere and its many variants (of which there are 24, some having names such as the Beaufort and St. Cyr) are susceptible to analysis both by this method, which relies on frequency counts of individual letters, bigrams, etc., and by differential methods particularly well suited for computer implementation.[7]

Another kind of polyalphabetic substitution is shown in Figure 2. In place of the predictable alphabets of the Vigenere kind, the cipher alphabets are randomly generated and unrelated to one another. A total of 26!, or about 4×10^{26}, alphabets are available,** of which just ten are used here.

Instead of enciphering the plaintext by means of a periodic key, a nonrepetitive key (in this example, *pi*) is used. The resulting ciphertext is secure until a persistent analyst distributes the cipher letters into ten groups, using the digits of *pi* as a guide. If there is sufficient ciphertext, each of the ten groups will exhibit the frequency distribution of normal English, except the set will have undergone a simple-substitution transform. All is lost.

pt:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	K	Y	H	F	G	Z	D	R	N	O	P	J	A	E	I	L	C	Q	M	V	W	B	X	T	U	S
1	T	L	K	A	D	Q	F	H	N	J	X	E	I	C	Y	M	G	U	Z	V	W	S	R	O	F	B
2	L	E	M	F	R	N	C	Y	D	U	H	J	P	W	X	Z	Q	B	V	A	K	T	S	G	O	I
3	J	E	D	Q	B	C	O	I	G	N	H	Y	Z	S	A	M	U	V	R	F	X	W	T	K	P	L
4	J	W	K	B	H	E	C	L	Y	N	D	F	V	M	U	R	P	O	G	X	Z	I	Q	A	T	S
5	Q	L	N	G	X	A	F	R	M	H	U	D	V	P	K	B	E	J	S	T	I	Z	C	O	W	Y
6	G	Z	M	A	F	T	K	Q	V	L	Y	B	N	S	C	J	E	P	H	O	W	X	D	I	R	U
7	T	V	N	Z	U	B	X	S	M	A	Y	D	J	W	R	L	I	H	E	F	C	K	G	P	Q	O
8	H	D	K	O	U	P	V	F	G	R	T	C	I	E	Q	J	L	Y	S	X	M	A	B	N	Z	W
9	P	W	R	H	C	J	O	N	Z	M	K	I	A	B	F	U	S	X	E	T	G	Q	V	L	D	Y

A. Tableau of random alphabets.

pt:	p	e	t	e	r		p	i	p	e	r		p	i	c	k	e	d		a		p	e	c	k		o	f
key:	3	1	4	1	5		9	2	6	5	3		5	8	9	7	9	3		2		3	8	4	6		2	6
ct:	M	D	X	D	J		U	D	J	X	V		B	G	R	Y	C	Q		L		M	U	K	Y		X	T
pt:	p	i	c	k	l	e	d		p	e	p	p	e	r	s													
key:	4	3	3	8	3	2	7		9	5	0	2	8	8	4													
ct:	R	G	D	T	Y	R	Z		U	X	L	Z	U	Y	G													

B. Encryption using the key pi.

MDXDJ UDJXV BGRYC QLMUK YXTRG DTYRZ UXLZU YGXXX

C. Ciphertext in 5-letter groups for transmission.

Figure 2—Substitution using random alphabets

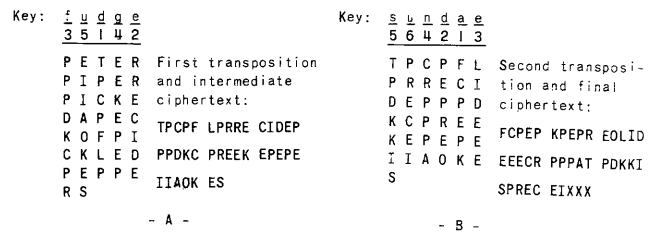


Figure 3—Double columnar transposition

Anticipating this contingency, the astute encryptor will not use 3.1415... as the invariable starting point of his key, but will begin each message at a different place in the *pi* sequence. If the analyst remains convinced that *pi* is the key, he must undertake the laborious task of checking each digit in the sequence as the potential starting point for each message.

This effort is both time-consuming and costly. Both factors work to the advantage of the encryptor. The "time factor" operates to keep the message secure long enough so that it may be out-of-date and useless when the enemy finally reads it. The "cost factor" operates such that the enemy may pay a higher price for the information than it is worth to him. Time and cost factors remain important when the scene shifts to the computer environment.

The random-alphabet example raises an engrossing question: How much text is required in a cipher of this nature so that only one meaningful interpretation is possible? The answer is a function of (1) the amount required for the simple-substitution case, and (2) the number of alphabets used in the cipher. In general, the product of these two numbers is near the minimum amount of ciphertext necessary for cryptanalysis from an information-theoretic viewpoint.† From the viewpoint of the pencil-and-paper analyst, this amount is too low by a factor of about five, depending on the nature of the plaintext.

Transposition types

Traditional forms of the second major encryption algorithm, transposition, are illustrated in Figures 3 and 4. In Figure 3-A, the plaintext has been written out under a keyword. A transposed version is then obtained by taking the letters out by column, the order of the columns being determined by the numerical sequence of the key letters in the normal alphabet.

The ciphertext at this stage is a "simple columnar transposition," which presents little difficulty to the analyst, even though the columns are of two different lengths. In geometrical terms, simple columnar transposition is a 1-dimensional operation since the plaintext is converted in effect into a series of disjointed line segments.

* For reasons of space, this simple example must suffice to support the following premises: (1) Given sufficient ciphertext, if there is a small enough period (say 100 or less for pencil-and-paper work and several decimal orders of magnitude greater for computer analysis), the period can be recovered. (2) Even if the period is not constant, but varies by some definite rule, it can be recovered. Details may be found in references in the selected bibliography.

** More than enough to encipher everything written since the invention of writing (2×10^{15} letters is a conservative estimate of the upper bound) without repeating a single alphabet. Recalling the similar staggering statistics for simple substitution, the reader is not impressed.

† Readers familiar with Shannon's work will recognize the "unicity distance," which is treated later.

```
P O K C E P A
E F R E P P D
T P S X X E E
E I X X X P K
R C K L E D C
P I P E R P I
```

Ciphertext:

POKCE PAEFR EPPDT
PSXXE EEIXX XPKRC
KLEDC PIPER PIXXX

Figure 4—Typical route transposition

In Figure 3-B, the ciphertext of 3-A is subjected to further transposition, using the identical algorithm but a different keyword.† The final cipher has a surprisingly high resistance to analysis. Double columnar transposition is 2-dimensional in that each letter may be equated with a particular cell in an X-Y matrix.

Figure 4 illustrates another form of 2-dimensional transposition known as a "route" cipher. Here, the plaintext has been written into a 7×6 matrix in a counterclockwise spiral. The ciphertext is then taken out by rows. Other routes are also possible, but the example suffices to show the principle.

A cryptanalytic technique called "multiple anagramming" applies to transposition ciphers when one has two or more messages of the same length (or suspected block length). By manipulating the ciphertext of one message to produce plaintext, and carrying out the identical operations on the other message(s), if plaintext results in the other message(s) as well, the decryptment is achieved.

To close the discussion of transposition types for the present, Figure 5 shows a "3-dimensional" technique not described in any of the known literature. The example is trivial but allows elucidation of principles which relate to programming techniques of value when the general case is described for computer application.

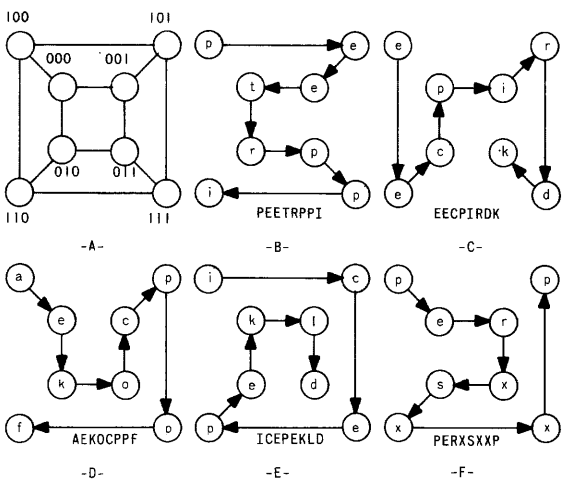


Figure 5—A 3-dimensional route transposition

† If the keyword for the second transposition is the same as the one used for the first, the cryptosystem is known as the "U.S. Army Transposition Cipher," of World War I vintage.

Figure 5-A will be seen to be the topographical equivalent of a unit cube with *X-Y-Z* origin at 0,0,0; with the vertex of the major diagonal at 1,1,1, and with intermediate vertices suitably labelled (commas have been omitted in the diagram). The reader is invited to view Figure 5-A as the "shadow" of a 3-dimensional object cast on the 2-dimensional paper; the concept will be serviceable later.

Viewed as a graph in matrix terminology, there are many routes through the structure. Two kinds of route are of special interest, the "Hamiltonian path" and the "Hamiltonian circuit."* A Hamiltonian path is one which passes through every vertex in the graph once and only once. The Hamiltonian circuit is a special case of the path wherein the last element in the path is so located that the first element is adjacent, allowing the path to be repeated.

In Figure 5, five unit cubes have been used to transpose the plaintext. For clarity, the plaintext has been written into each cube from left to right and from top to bottom.** The ciphertext is then read out of the cubes via a different Hamiltonian path for each cube. Figures 5-B, 5-D, and 5-F are circuits as well as paths.

TABLE I—Hamiltonian Paths of Figure 5

5-B	5-C	5-D	5-E	5-F
100	100	100	100	100
101	110	000	101	000
001	010	010	111	001
000	000	011	110	011
010	001	001	010	010
011	101	101	000	110
111	111	111	001	111
110	011	110	011	101

Using dimensional notation, the paths of Figures 5-B through 5-F are shown in Table I. The path sequences, it will be noted, form Gray codes. Because of the Gray-code property, all may be generated by a relatively simple software routine. Although the example here is for three dimensions, the simplicity holds for the general case of *n*-dimensions.

CRYPTOGRAPHY IN TRANSITION

The Vernam era

For the computer-oriented, the *locus classicus* of modern cryptotechnology is a paper written 47 years ago by Gilbert Vernam of the Bell Telephone Laboratories.[8] The paper describes (what today would be called) a binary cryptosystem suitable for use with the 5-level Baudot Teletype code, a code which still may be found in wide

* Named after Sir William Rowan Hamilton, who first described them in the mid-19th century.

** In practice, the plaintext would be entered using one path and the ciphertext read out using another. The legitimate receiver, knowing the two paths, need merely reverse the process to decrypt the message.

use today.* The two possible truth tables of the cryptosystem are shown in Figure 6, together with samples of the two possible encryptments and decryptments. The system is identical to that used in many of today's computer encryption devices, although the character length of the latter has been increased in most cases to accommodate standard data-processing character sets.

The security of the Vernam system results from the use of an apparently random key of great length. The important word here is "random." Contrast a random nonrepeating key with the predictable, "semi-random," key of Figure 2, *viz., pi*. If the key were truly random, and if it were used only once, the enemy analyst would be impotent, professionally at least.** What constitutes true randomness is a matter best left to mathematicians.

Vernam obtained his lengthy key by using two punched-paper tape loops of character length j and k, where j and k are mutually prime. The encipherment equation is thus: $ct_i = pt_i \oplus j_i \oplus k_i$. For each cycle of the j tape, the k tape advances one character, yielding a total key length of $j \times k$. Typical values for j and k during the '20s were 775 and 776, for an over-all key length of 601,400 characters. In a similar but computer-based system today, using magnetic tape for key storage, $j = (5 \times 10^6)$ and $k = [(5 \times 10^6) + 1]$ are attainable values, for a total key of the order 2.5×10^{13}.

Numbers of this magnitude appear irresistible to advertising managers for computer cryptosystems. The reader, however, may now be more suspicious than stunned, possibly as the result of previous examples. His suspicion is not unfounded.

An example may serve to demonstrate one of the characteristics of this kind of cipher. The example is artificial only in the sense that it compresses into a brief interval data which normally could be acquired only after the interception of considerable ciphertext. The phenomenon itself, owing to the nature of language, is certain to occur.

In Figures 7-A, 7-B, and 7-C, the same plaintext, b a r g e, has coincided with the same j_i key, *1blpt*, but with three different k_i keys, *c, m,* and *y*. Three different ciphertexts result: SHP ≡ ⌊(ct_1); QI#TF (ct_2), and VKFB# (ct_3).

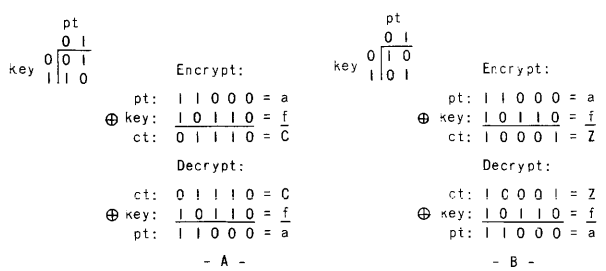


Figure 6—Basics of Baudot cryptography

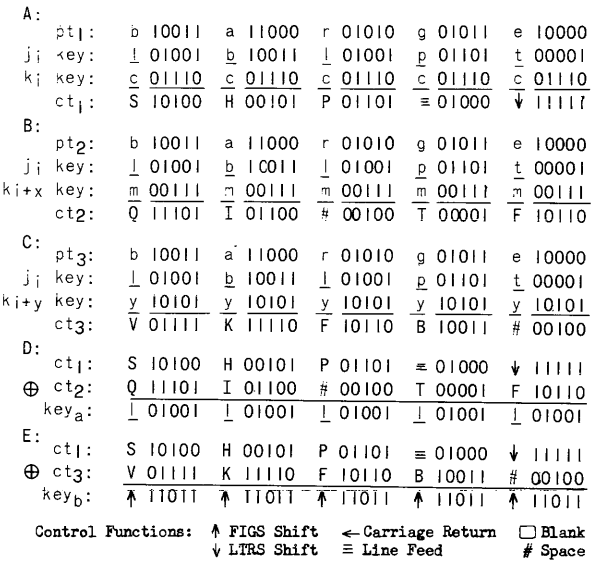


Figure 7—Recovery of minor cycles from Vernam cipher

In Figure 7-D, ct_1 is added (vectorially)* to ct_2; in Figure 7-E, ct_1 is added to ct_3. In each instance, a constant *1* and ⌊ respectively, results. We shall refer to these constants as k_a and k_b.

The k_a and k_b are compound keys. k_a is the sum of k_i and k_{i+x}, yielding *1*. k_b is the sum of k_i and k_{i+1}, yielding ↓.

The appearance of these constant sequences signals the discovery of one of the minor cycles of which the complete ($j \times k$) cycle is comprised. Each minor cycle is initiated by the stepping of the k tape. Having found sufficient minor cycles with the identical compound key, the analyst may now apply the periodic technique of Figure 1, the Vigenere example. Ironically, the rigor of Boolean algebra ensures that the Baudot cipher alphabets are as immutable and predictable as those of the Vigenere.

Boolean rigor leads to a second weakness of this kind of cryptosystem: If any part of the key is used to encipher as few as two plaintext messages, the messages can be "lined up" by a technique known as the index of coincidence.** Figures 8-A and 8-B show just a portion of two such messages. In practice, much more text is needed in order to line the messages up.

The analyst then proceeds to add ct_1 to ct_2 as shown in Figure 8-C, to produce the compound key, k_c. Assume the analyst suspects the word n u m b e r is probably contained in one of the plaintexts. To test the assumption, he tries each k_c letter as the starting point of the word n u m b e r, reading the resultant diagonals to see if a

* In 1925, the then-Captain William F. Friedman invented a bit-transposition device (the equivalent of a plugboard) to increase the security of the system. The transposition was invariant until manually changed.

** The enemy, however, wins a point in that the truly random key must somehow be transmitted to the legitimate receiver (who cannot generate it himself, obviously, it being a one-of-a-kind sort of thing. This transmission gives rise to opportunities for interception, theft, and bribery.

* Throughout the remainder of the paper, the process of addition refers specifically to vector addition unless otherwise stated.

** The index of coincidence was discovered by William F. Friedman and published in "The Index of Coincidence and Its Applications in Cryptographic Analysis," *Riverbank Publications*, No. 22, Geneva, Il: Riverbank Laboratories, 1922. The method is described in Kahn (*op. cit.*, pp. 376-385).

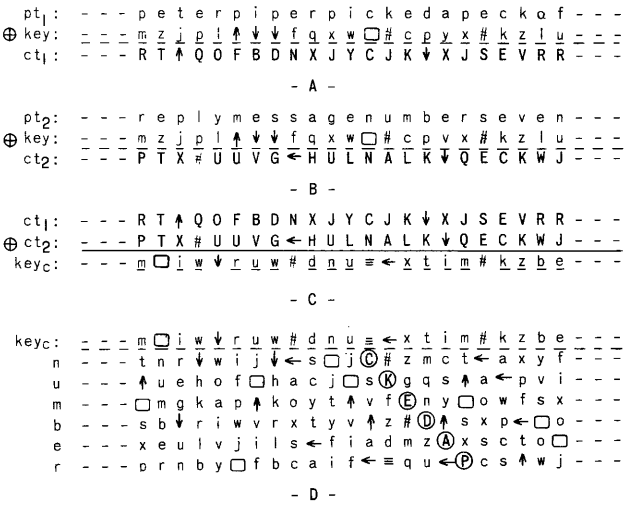


Figure 8—Probable word solution of Baudot cipher

reasonable plaintext sequence emerges (Figure 8-D). With luck and persistence, he finally obtains c k e d a p. The tongue-twister recognized, the analyst tries p e t e r p i p e r p i (c k e d a p) against k_c, and with much gratification obtains r e p l y m e s s a g e (n u m b e r) as the counterpoised plaintext.*

Extending the messages in the other direction, the analyst is aided by another phenomenon. The SEV of ct_1 is the s e v of pt_2 and the ECK of ct_2 is the e c k of pt_1.**

So long as the two ciphertexts continue to share the common key, the messages may be recovered by the near-mechanical process of assuming a letter-by-letter continuation in one plaintext and seeing if it results in an acceptable continuation in the other.

If the two preceding examples have produced further skepticism in the reader regarding those cryptosystems, the security of which is attributable solely to their having keys of length 10^{xx}, they have served their purpose.

The Shannon era

Kahn declares Friedman's discovery of the index of coincidence is "the most important single publication in cryptology."*** The practitioner is likely to agree. The theoretician may justifiably nominate Shannon's analysis of secrecy systems for the honor.[9]

The text, tightly knit in the manner of mathematical exegesis, admits of no easy summary. The writer faced with space limitations has open only a few options if his intent is to induce the reader to consult the original. The strategy selected here is to limit the discussion to just one topic, the "unicity point" or "unicity distance." (Shannon uses the terms interchangeably.)

Earlier in the paper the question was raised: What is the longest simple-substitution cipher that has only one meaningful resolution? In essence, this length is Shannon's unicity distance. But the value depends on the cryptosystem. For simple substitution, it is 27 letters. For the Vigenere example of Figure 1, it is 10 letters (or 2d, where d is the period length). For the random-alphabet example of Figure 2, it is 270 letters (equivalent to ten simple substitutions). For a periodic cipher of random alphabets and unknown key, the unicity distance is 53d, and so on.

In the course of developing his thesis, Shannon proves the unicity distance for a cipher which employs a random key, never repeated, is infinite. The cipher cannot be solved.

If, then, an impregnable cipher does exist, why is it not universally employed? The answer is logistics. To the originator of voluminous plaintext, the generation *and testing* of a truly random key, plus the expense of distributing it (and the dangers accompanying the distribution as mentioned earlier), and the coordination of its use to ensure its one-time-only employment, add substantially to the user's cost. Only the more affluent governments, and then only for the most sensitive texts, can afford it.

CRYPTOGRAPHY IN THE AGE OF AUTOMATION

Those aspects of pre-computer cryptography which have now been covered are essential if the unfamiliar reader is to understand what has occurred now that the computer is a commonplace tool of the cryptologist. Except for algebraic cryptography, a genre not previously described, the reader will see that though the language and the claims have changed, there is at least some justification for maintaining the skeptical attitude of the aphorism, *Plus ca change, plus c'est la meme chose.*

Algebraic cryptography

The seeds of modern algebraic cryptography were planted more than 40 years ago in two papers by L. S. Hill.[10,11] To one not mathematically trained, the procedures are complicated, even arcane. Mathematicians (one is told) perceive an inscape of excellence unrivaled by competing schemes. The discussion has been postponed till now because in the absence of edp equipment, the encryption and decryption tasks entailed high cost and time factors for the legitimate users, even if calculating machines were employed.*

* We have omitted the binary operations involved in this procedure, those in Figure 7 being deemed sufficient to show the principle. The full 5-level Baudot code can be found in many standard electrical engineering references by the painstaking reader who wishes to test the operation for himself.

** Again, the reminder, the examples have been contrived to show various contingencies in a short space.

*** Kahn, *op. cit.*, p. 376.

* Hill patented an unwieldy mechanical device which could operate on up to six letters ("hexagrams") per cycle. Computers permit polygrams of any size to be processed, at least in theory. Programmers familiar with the demands matrix algebra place on machine time will see that a practical limit exists. The limit is set by the acceptable trade-off between cost factor and the degree of security desired, both increasing exponentially with polygram size.

Figure 9 is a simplified example of one algebraic method, adapted from Davis.[12] The example consists of a matrix of order 3, which serves as the key, and a column vector, which is the plaintext. For encryptment, the numerical equivalent in the standard alphabet is substituted for the plaintext letters.

The encryptment algorithm is shown in detail in Figure 9-A. The operations involved in enciphering the plaintext p e t appear in Figure 9-B. For the remainder of the plaintext, only the skeleton of the encipherment is given. Figure 9-G is the resulting ciphertext.

Among others, the chief disadvantage of the method is that the encipherment process involves five steps per letter, or a total of 15 steps per 3-letter *pt* group. The number of steps per *pt* group grows quadratically with polygram size according to the formula, $S=n^2+(n-1)$, where S is the number of steps and n is the number of letters in the polygram.

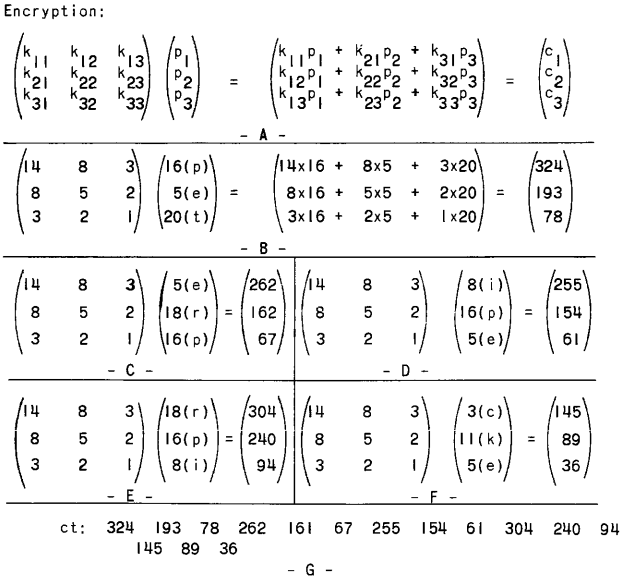


Figure 9—Elementary algebraic cryptography

An advantage which offsets the main disadvantage is also evident. Although the first 3- letter *pt* groups contain the letters *p e*, there is no indication of this in the ciphertext. Similarly, the reversal *i p* and *p i* in groups 3 and 4 is concealed. The principle applies regardless of polygram size. Thus if just one letter in an n-letter polygram differs from another n-letter polygram, the ciphertext will conceal the fact that all but $(n-1)$ letters are identical. The phenomenon denies the analyst the use of one of his more powerful tools, the analysis of repetitions in the ciphertext.

Figure 10 shows the decryption process for the ciphertext of Figure 9-G. Only the first 3-letter group is deciphered; the others follow the same paradigm.

The decryption key matrix, it will be noted, is not the same as the encryption matrix. The disparity may appear to add to the security of the cipher but the inference is misleading. The decryption matrix is merely the inverse

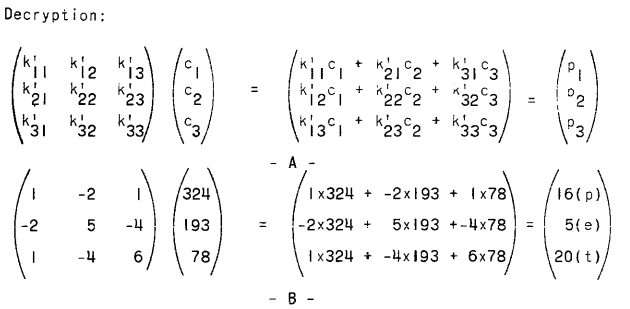


Figure 10—Algebraic cryptography decipherment

of the encryption matrix, a familiar mathematical procedure.

Commercial cipher systems

Algebraic cryptography aside, most commercial cryptosystems depend on means for generating a key which to the casual observer appears random but is in truth only pseudorandom. The commercial systems take the form of both hardware and software. The two kinds may conveniently be discussed together because whatever can be performed by hardware may be emulated by software. Indeed, some systems employ software at the computer site and hardware at the remote terminals.

The first algorithms for generating pseudorandom keys for computer use appeared in the '50s. The resulting key was fully deterministic, derived by a method identical or similar to the binary equivalent of the decimal example shown in Table II. The procedure begins by selecting a number, say 6378, squaring it, and then proceeding as the table indicates.*

The operation yields the sequence 7-8-8-2-5-8-4-5-5-0-1-9-2-1-1-4-5-7. The series is apparently random but wholly determined.

Today, the most commonly encountered commercial cryptosystem is the "shift register." Despite design variations, the principles and more importantly the results are identical: Shift registers are pseudorandom key generators, but of a kind different than that illustrated in Table II.

TABLE II—Generation of Pseudorandom Key

Operation		Product	Key Sequence
6378×6378	=	40678884	788
788×6378	=	5025864	258
258×6378	=	1645524	455
455×6378	=	2901990	019
019×6378	=	121182	211
211×6378	=	1345758	457
...		...	...

* A curious sidelight of the era was the discovery *and promulgation* of a rather small set of numbers, favored because they produced long pseudorandom sequences. Analysts presumably concealed their delight.

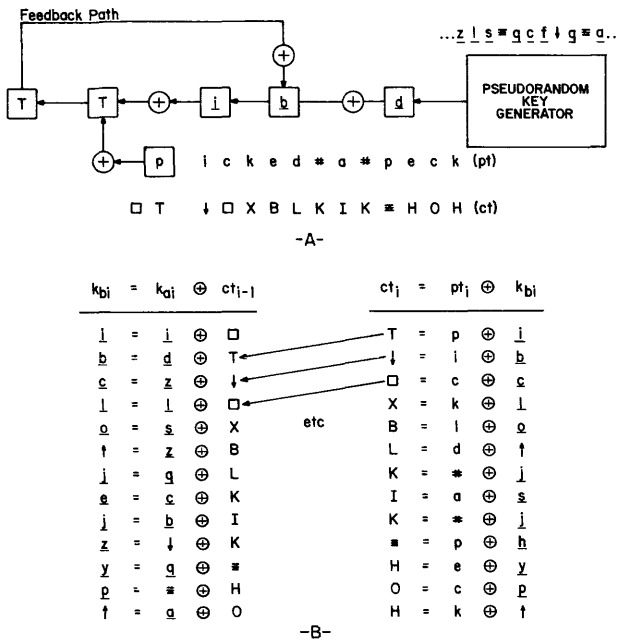


Figure 11—Simplified shift-register operation

The key generated by a shift register is (in all cases worthy of consideration) not deterministic but Markovian. A brief quote from Feller[13] succinctly states the process: "If two independent systems subject to the same transition probabilities happen to be in the same state, then all future probabilities relating to their future developments are identical."*

In the case of the shift register, the two "independent systems" are the pseudorandom key from the key generator and the stream of the plaintext; designers may validly dispute the term "independent" as applied to the key generator. We retain it for the sake of the following example.

Figure 11 shows a simplified shift-register system. For clarity, the operations use Baudot encipherment (truth table of Figure 6-A).

The action is portrayed *in medias res*, since initial start-up conditions are unique and at most occur once per message. A comparison of Figure 11 and Figure 7 will reveal that in both cases a compound key is used to encipher the plaintext. In Figure 7, the final key is the sum of j_i and k_i. In Figure 11, the final key, k_{bi}, is the sum of k_{ai} (the current output of the key generator) and ct_{i-1}, the cipher counterpart of pt_{i-1}, the last-enciphered plaintext letter. The compound key is formed by feedback from the ciphertext output stage. Feedback in one form or another (and it is usually more complex than shown here) is an essential feature of shift registers.

Figure 11-A portrays the case $ct_i = pt_i \oplus k_{bi}$ (that is, $T = p \oplus i$); simultaneously, k_{bi+1} is being formed by the process, $K_{bi+1} = k_{ai} \oplus ct_i$ (that is, $b = d \oplus T$). The equations are rearranged slightly in Figure 11-B to show the formation of the successive k_{bi+n} and at the same time the interrelationship of the two streams. In order to start the sequence in Figure 11-B, we have assumed ct_{i-1} was □ and k_{ai} was i; the assumption also explains why $k_{bi} = i$ in Figure 11-A. As so many things are, the process is hard to explain but relatively easy to implement. The explanation has succeeded if designers who initially objected to the term "independent" are now modified.

Figure 11-C, in turn, shows the formation of the successive ct_{i+n}. Readers still with us will see that in Figure 11-A, the *pt* stream and the *ct* stream appear in proper superposition.

In practice, the commercial shift register is frequently a cascaded series of binary stages. The maximum length of the pseudorandom key cycle is $(2^n - 1)$, where n is the number of stages. A common length for the shift register is 20 stages, yielding a key cycle of 1,048,575 bits.

Some commercial shift registers provide the capability of allowing the user to change the feedback connections, and thus alter the pseudorandom key stream. Different key streams obtained in this way are usually referred to as "codes." An article by Twigg[14] treats the design logic of these devices. Interestingly, a complementary article by Meyer and Tuchman[15] outlines a method of attack on the ciphertext of such systems based on the recovery of just a small part of the key stream.

Another method of attack is that of Figure 8. It is applicable when two messages enciphered with the same key can be lined up. If the user varies the initial setting of a given code for each message, the enemy must intercept considerable traffic in that code before he can achieve this felicitous condition. (It is appropriate to suggest here that the user never let his line go "dead." Meaningless character streams should fill the void between legitimate messages, to prevent the enemy from detecting the start and end points of messages.)

In the absence of a definitive comparison of off-the-shelf commercial cryptosystems, let the writer nominate his own candidates for the top and bottom rungs of the security ladder—both, of course, a matter of personal opinion.

Of the systems examined, the top rung is occupied by the IBM Feiste;/Notz/Smith system.* The design is too complex for explanation here, though on the other hand the user interface is admirably simple.**

The apparently unchallengeable occupant of the bottom rung of the security ladder is the not-inexpensive "XYZ" system. The "black box" is furnished with a two-code "module," although users may purchase additional modules up to a total of more than 8 million codes. The key length is not revealed. However, it is irrelevant. To simplify operation by the user, the system is reset anew for each message to exactly the same place in the keying cycle.

* Feller, *op. cit.*, p. 420.

* Girdansky, *op. cit.*, pp. 6–12.

** As the poet-author of Ecclesiastes asked, "Is there a thing of which it is said 'Lo, this is new?'" The Feistel/Notz/Smith method incorporates a programmable version of Friedman's bit-transposition scheme referred to earlier.

The delighted analyst may now return to the example of Figure 2 and the accompanying text, where the fallacy of beginning each message at an invariant starting point is explained.

A polydimensional transposition cipher

The following cipher is described not for the usual reason (*i.e.*, the amateur cryptologist has devised still another "unbreakable" system) but because it poses—rather, may pose—a challenge to the theoretical mathematician. The cipher is an (unbounded) extension of the 3-dimensional system of Figure 5. The extension will be described first. The challenge follows shortly.

The reader was asked to view Figure 5-A as the shadow of a 3-dimensional cube cast on the 2-dimensional paper. In the same way, without trying to visualize the object itself, the reader may consider the graph in Figure 12-A

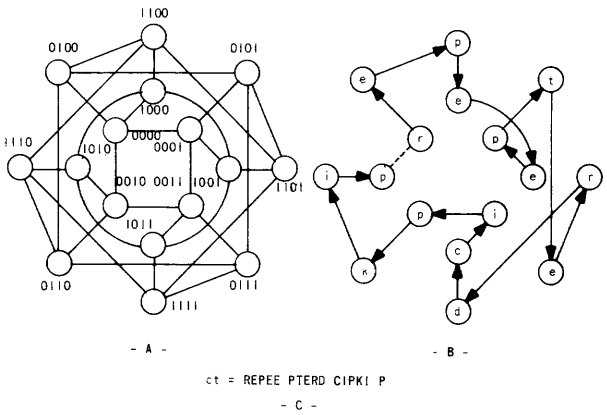


Figure 12—A 4-dimensional route transposition

as the shadow of a 4-dimensional hypercube, or tesseract, as it would appear in 2-dimensional space.

Paralleling the example of Figure 5-B, the plaintext p e t e r p i p e r p i c k e d has been written into Figure 12-B from left to right and from top to bottom (although, as with Figure 5, in practice a Hamiltonian path should be used). The ciphertext has been taken out by a path which is also a circuit.

The tesseract, the vertices of which are identified in vector notation, exhibits the same property as the cube. All Hamiltonian paths and circuits form Gray codes. This characteristic holds for the general case of the n-dimensional hypercube. The reader may test this for himself by tracing paths and circuits in the 6-dimensional hexact (or rather its shadow) in Figure 13, the largest hypercube which may reasonably be drawn in the available space.

The reason for surmising that the polydimensional transposition may be usable as a secure commercial cryptosystem is based chiefly on the evidence in Table III. Note the rate of growth of the number of paths with increasing dimension. The table ends at dimension 4 because it was estimated that three months of continuous

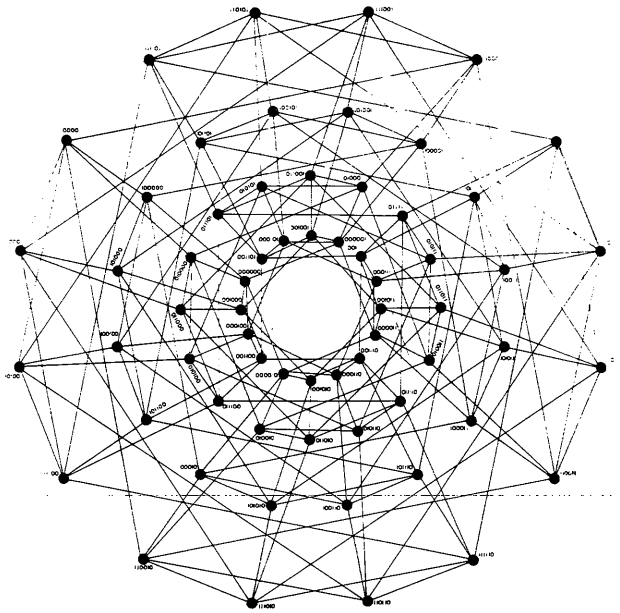


Figure 13—Hypercube of dimension 6 (Hexact)

computation on the UNIVAC 1107 would be required to list exhaustively the paths and circuits for the 5-dimensional pentact if the program prepared specifically to count Hamiltonians for the n-dimensional case were used.

Other grounds for giving the system further consideration include: (1) The routine which generates the Hamiltonian paths is relatively simple and makes but slight demands on high-speed memory. (2) If successive blocks of plaintext are encrypted and decrypted by paths which vary in pseudorandom manner, multiple anagramming as a cryptanalytic tool is defeated. (3) Different sets of paths can be dedicated to individual remote sites, thus preventing sites from reading traffic not intended for them. This capability does not exist in some current commercial systems.

One advantage to the legitimate user is that he need not generate all possible paths for a hypercube of dimension (say) 20.* He need generate only a few thousand or tens of thousands—a relatively simple task. The enemy on the

TABLE III—Hamiltonian Paths and Circuits of the n-dimensional Unit

Unit Dimension	Hamiltonian Paths	Hamiltonian Circuits
0	0	0
1	0	0
2	2	2
3	144	96
4	91,392	43,008
5	—	—

* A 20-dimensional hypercube may seem to imply that the ciphertext must comprise blocks of more than a million characters each. The inference is not true because all vertices need not be filled—a complication easily programmable but which adds substantially to the enemy's work factor.

other hand must try all paths—and the trend of the data for dimension 20 indicates this is impossible.

This is where the challenge to the reader enters: What is the generating function for the number of Hamiltonian paths and circuits for the general case of the n-dimensional hypercube? A respectable amount of effort by qualified mathematicians, supplemented by inquiries by the writer, has failed to unveil it. One is tempted to suspect that for the 20-space hypercube, the number of Hamiltonian paths is of the order 10^{xxx}.*

A CRYPTOGRAPHIC SCENARIO

Till now, the emphasis of the paper has focused on the means available to the enemy to "read the mail" of the cryptouser. This stress may have given the user qualms about the security of his communications, an uneasiness which may have some slight justification in fact. But in practice, the defenses of the user (in the specific area of cryptography) surpass the weapons of the enemy to an overwhelming degree. (We speak here of nongovernmental users and enemies.)

Let the hypothesis be made that MSI is a large international corporation which maintains extensive digital links among many transcommunicating data banks holding information of a most sensitive nature. MSI is continually reminded of its vulnerability by the many vendors of commercial cryptosystems. But MSI's manager of telecommunications has not yielded to the suasions of any one vendor and has adopted a cryptosystem which admits not only of frequent and easy change of key but of the basic system itself.

The enemy is IPF, MSI's largest competitor and rumored to budget a sizable amount each year for industrial espionage. Regard first the probable strategy of IPF in allocating its espionage fund:

- Planting persons on MSI's payroll appears to be the tactic with the greatest potential payoff, and thus may account for the largest share of the funds available.
- Bribing MSI employees and its vendors, the tactic which judgment would seem to rank in second place, consumes another share of the budget.
- Of the money allotted to wire-tapping, bugging, and digital eavesdropping, the two former activities probably receive priority.

What resources are on hand for the manager of the IPF digital wire-tap fund? (We will grant him the knowledge of which data links carry the information of most value.)

An obvious first need is a cryptanalytic staff. The staff must be familiar not only with cryptanalysis and with the protocols of digital communication, but must also be criminally inclined. While there does exist a pool of government-trained analysts with the first two qualifications, it is unlikely they would participate in illegal activities unless they have greatly changed their lifestyles since receiving their clearances.

But let's grant the manager his staff. Next he needs a fairly sophisticated data-processing system which stresses mass storage for recording the intercepted bit streams.

Let's grant these facilities also, though by now the manager has probably exceeded his budget which was severely limited in the first place. What of the time and cost factors? We assume MSI keeps its lines active in the absence of genuine message traffic. Then not only is it perversely difficult to locate the (enciphered) messages themselves, it is often a stupendous task merely to identify the system in use. And by the time the key for a given message has been discovered, the plaintext may refer to a division of MSI which had been sold two weeks ago. MSI's telecommunications system, then, appears reasonably secure from cryptanalytic attack. However, this conclusion is drawn with the emphatic qualification that it pertains to the state of the art as it exists today.

It may be instructive, though, to view the situation from the eyes of MSI's manager of telecommunications. He has wisely initiated cryptographic procedures which offer high theoretical and practical security. But unfortunately he must delegate responsibility for day-to-day operations to an army of programmers, operators, and clerks. All have many admirable qualities. Also, they are variously careless; forgetful; malicious; indifferent; hurried, and possessed of all the usual failings of humanity in general. As a result, MSI's manager is frequently confronted with such situations as:

- A site transmitting in one cryptosystem to a second site currently set up to receive in another system.
- Plaintext somehow evading the cryptoroutine and going out on the line *en clair*.
- The same message transmitted repeatedly in the same system with but slight variation in key.
- Messages (the more important ones) vanishing in a void, never to be seen in plaintext form again.

Sometimes, the manager must yearn for an unlimited budget which would allow him to install the most sophisticated equipment available, and hire and train persons of only the highest caliber and personal integrity. The system would then work perfectly. One fervidly hopes his dream is not shattered, as shattered it might be, by the following quotation:

> *Security Note: I had asked that a cable from Washington to New Delhi summarizing the results of the aid consortium be repeated to me through the Toronto Consulate. It arrived in code; no facilities existed for decoding. They brought it to me at the airport—a mass of numbers. I asked if they assumed I could read it. They said no. I asked how they*

* In light of our previous skepticism concerning large numbers, no guarantee is implied or should be inferred regarding the security of the system based on this number alone.

managed. They said when something arrived in code, they phoned Washington and had the original message read to them.[16]

CONCLUSION

Inevitably, at various places in the preceding discourse, the feisty reader has objected, "Ah, but what the writer alleges is a weakness may be offset easily by adding the X complication." Just so. But the analyst has at hand the X' countermeasure which negates or ameliorates the X factor. To which the reader may reply, "Yes, but a Y-type strategy will nullify the X' remedy, and thus be a countercountermeasure." About this time, the analyst dusts off his Y' technique, the countercountercountermeasure. And so on. The situation is reminiscent of the ECM, ECCM, ECCCM, . . . spiral. While the cryptographic and electronic countermeasure chains may not be infinite, they appear surely to be unbounded. One must cut the cord somewhere. Here.

ACKNOWLEDGMENTS

The writer is grateful to Jon Tempas of Univac who wrote the program for counting the Hamiltonians of the n-dimensional unit, and for preparing an analysis of how, and in how many ways, the n-dimensional hypercube may be dissected into its component $(n-m)$ hyperflats. He is also beholden to those friends in the American Cryptogram Association who unknowingly lent their *noms de chiffre* to some of the examples in the paper. To another group he is obligated for anecdotal material. Nor should one be neglectful of one's teachers, especially those who awakened an interest in language, even if foreign; regrettably, as a whole, these have fallen on hard times of late.

REFERENCES

1. Gaines, Helen Fouche, *Elementary Cryptanalysis*, American Photographic Publishing Company, 1943. Reprinted under the title, *Cryptanalysis*, Dover Publications, New York, 1956.
2. *The Cryptogram*, published Bimonthly by The American Cryptogram Association, Rogot, E. & E. 9504 Forest Road, Bethesda, Md. 20014.
3. Farago, Ladislas, *The Broken Seal: The Story of "Operation Magic" and the Pearl Harbor Disaster*, Random House, New York, 1967.
4. Friedman, William F. and Elizabeth S., *The Shakespearean Ciphers Examined*, Cambridge University Press, London and New York, 1957.
5. Sinkov, Abraham, *Elementary Cryptanalysis—A Mathematical Approach*, Random House, New York, 1968.
6. Kahn, David, *The Codebreakers*, The Macmillan Company, New York, 1967.
7. Girdansky, M. B., *Data Privacy—Cryptology and the Computer at IBM Research*, IBM Research Reports, Vol. 7, No. 4, 1971.
8. Vernam, G. S., "Cipher Printing Telegraph Systems," *Journal of the AIEE*, Vol. XLV, February, 1926.
9. Shannon, C. E., "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, Vol. 28, October, 1949.
10. Hill, Lester S., "Cryptography in an Algebraic Alphabet," *American Mathematical Monthly*, Vol. 36.
11. Hill, Lester S., "Linear Transformation Apparatus," *American Mathematical Monthly*, Vol. 38.
12. Davis, Philip J., *The Mathematics of Matrices*, Balisell Publishing Company, Waltham, Mass., 1965.
13. Feller, William, *An Introduction to Probability Theory and Its Applications*, John Wiley & Sons, Vol. I, 3rd ed., 1968, New York.
14. Twigg, Terry, "Need to Keep Digital Data Secure?," *Electronic Design*, Vol. 23, November 1972.
15. Meyer, C. H., Tuchman, W. L., "Pseudorandom Codes Can Be Cracked," *Electronic Design*, Vol. 23, November 1972.
16. Galbraith, John Kenneth, *Ambassador's Journal*, Houghton Mifflin Company, Boston, Mass., 1969 (p. 115; used bypermission).

Information theory and privacy in data banks*

by I. S. REED

The Rand Corporation
Santa Monica, California

INTRODUCTION

The problem of providing privacy and security in retrieval systems falls into two rather modern disciplines: information theory and computer science. In this paper the concern is primarily with the former, i.e., how to relate security of data records in computerized retrieval systems and data banks with Shannon's information-theoretic treatment of secrecy systems for natural language messages in communication systems.[1] In doing so, it is useful to establish first the analogy between retrieval systems and certain communication channels.

RETRIEVAL SYSTEMS AND COMMUNICATION CHANNELS

Retrieval systems and communication channels have many similarities. The encoding and insertion of records into a file is akin to the process of transmitting messages through a communication channel. The acts of addressing, accessing and retrieval of records from the file are, likewise, similar to the operations that take place at the receiver end of a communication channel. (The analogy is even stronger between a *common carrier* communications system and a retrieval system—each station of the former may be either a transmitter or a retriever, or both, and many of the addressing, accessing and waiting-time problems have counterparts in a computerized information retrieval system.)

The addresses or storage locations of words in the retrieval system's memory may be likened in the communications channel to the "position" of a signal waveform in frequency and time. Synchronizing signals, header information and identity codes of a message in a common carrier communication system are analogous to the key-word data in a record in a retrieval system. Finally, the access time problems associated with the open-addressing[2] or the hash-addressing[3] techniques in modern retrieval systems are statistically similar to some of the waiting-time and queueing problems in common carrier communication networks.[4]

In order to analyze retrieval systems, it is necessary to become more specific and to idealize the nature of such systems. At the "transmitter" or the encoding and insertion end of a retrieval system is the information source of records to be entered into the file (storage medium). The file or storage medium is the "channel". At the "receiver" end of the system is a retrieval mechanism which generates output records. A schematic "flow diagram" for this situation is depicted in Figure 1.

The *record source*, which could be the output of one or many individuals or machines, produces records to be encoded and stored. The encoder generates an address, which can be a function of both the record and the state of the storage medium, encodes the record in symbols from some alphabet (e.g., binary), and inserts the data into the storage.

At the *retrieval* end, an address is generated either from keyword information or a directory, and used to gain access to the desired record. Once accessed, the record is read out. Here, such a readout operation is assumed to be destructive (i.e., no copy of the record remains in the storage medium). To accomplish the equivalent of nondestructive readout, which is usually desirable, the record, as it is being read out, can be reentered into the record source. By this means, some records can be discarded and others kept, thereby providing a means for efficient storage and retrieval management.

Perhaps the greatest physical difference between a communications system and a storage-retrieval system is the dwell time—the time that data "stays" in the channel or storage medium. In the ideal channel, the transit time is assumed to be constant. However, in many "real" channels (e.g., in air-to-ground relay links) the length of the transmission path is variable and, as a consequence, transmission time varies from message to message. The dwell time of a given record, besides being variable in time, also varies from record to record. In fact, the dwell time of a record in a random access storage system is evidently a random variable with some probability density function. For the ideal channel, the dwell time probability density is a Dirac delta function, whereas for a random access storage one would expect a dwell time distribution to have the character of a waiting time distribution. For the present purposes, it is convenient to make the assumption that the dwell time density is truncated. That is, the dwell time of any record R is assumed to be less than some value T, where $T>0$.

* The research reported in this paper was supported by the National Science Foundation Grant # GI-29943. However, any views or conclusions contained in this paper should not be interpreted as representing the official position of the National Science Foundation or The Rand Corporation.

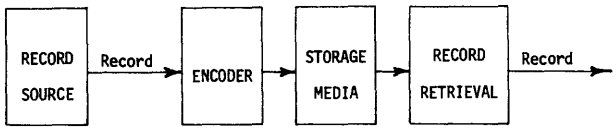


Figure 1—Schematic of a general storage and retrieval system

PRIVACY TRANSFORMATIONS AND MUTUAL INFORMATION

In the above, the analogy between an information retrieval system and a communication system was developed. Here we show that, from the viewpoint of an invader of a data bank, the transformation of messages by either enciphering or noise is similar to the action of noise on a communication channel.

A schematic of a privacy system for an information data bank is shown in Figure 2. At the input to the data bank there are two sources of information. One is the record source and the other is the source of privacy transformations. Privacy transformations are applied to the record source. This results in a perturbed or enciphered record source which is stored in encoded form in the data bank or storage medium.

The dotted line to the decoder at the retrieval end indicates the data transmitted to the decoder needed to correct the perturbations or distortions of the record. Data about the privacy transformations is made available only to those requesters of records who have a "need to know". Such information about the privacy transformation is presumed to be noninterceptible.

The above scheme for a privacy system is very similar to communications over a noisy channel. The source record, which corresponds to the transmitted signal, is distorted by randomly chosen privacy transformations. The perturbed record, residing in the data bank, is analogous to a signal plus noise in a communications channel. However, in the communications channel the simple addition of noise to the message (see Reference 5) is usually a less complex transformation than is used for a secrecy system (see Reference 1, Section 3).

If R is a record and T is a privacy transformation, the perturbed or enciphered record, E, is functionally related to T and R as follows:

$$E = T(R)$$

where T is regarded as an operator on R. Assume that T is an element of a set of such transformations. For simplicity, let this set be finite, i.e., the set if $T_1, T_2, \ldots, T_m$. Associate with each element T_k is a probability P_k. The transformations $T_1, T_2, \ldots, T_m$ together with their probabilities form what is known as a finite scheme (see Reference 6, p. 2):

$$P = \begin{pmatrix} T_1 & T_2 & \ldots & T_m \\ P_1 & P_2 & \ldots & P_m \end{pmatrix}$$

One can similarly assume that there is a finite number of records $R_1, R_2, \ldots, R_n$ at the source with their associated *a priori* probabilities $p(R_1), p(R_2), \ldots, p(R_n)$. The records and associated probabilities form the finite scheme

$$X = \begin{pmatrix} R_1 & R_2 & \ldots & R_n \\ p(R_1) & p(R_2) & \ldots & p(R_n) \end{pmatrix}$$

for the records source.

At the receiving end of a *secrecy system* it is desirable to recover R *uniquely*, given the received perturbed record E and the transformation T. Hence, for a secrecy system each T of a scheme P has a unique inverse T^{-1} such that $TT^{-1} = I$, the identity transformation. Thus, if E is received, the original record is uniquely decoded as $R = T^{-1}E$. So, following Shannon,[1] ". . . a secrecy system is a family of uniquely reversible transformations, T_i, of a set of possible messages into a set of cryptograms, the transformation T_i having an associated probability P_i."

The requirements of a *privacy system* are not as stringent as those of a secrecy system. For example, personal records stored in large data banks, such as census and income tax files, might be needed to study the distribution of incomes among various groups of professionals. In such a case many of the needs of personal privacy are met if an individual's record is merely *distorted* by the privacy transformation T. For satisfactory privacy, the "level" of distortion of a personal record should be sufficiently high to make inferences about personal identification nonunique, yet low enough for the distorted records to be used in, say, statistical analyses.

In order to quantify the degree of nonuniqueness and the level of distortion which is obtained from a privacy transformation, it is necessary to borrow from the measurements of modern information theory and rate distortion theory. For the present purposes it is convenient to use the notation and machinery of some recent references (e.g., see Gallager in Reference 7).

The *average mutual information* is defined by

$$I(X;Y) = \sum_x \sum_y P(x,y) \log_2 P(x,y)/P(x)P(y) \quad (1)$$

where $P(x, y)$ is the joint probability distribution of the record source and the receiver space, and $P(x)$ is the distribution of the source. $I(X; Y)$ is the most generally accepted measure of uncertainty about the source X given the re-

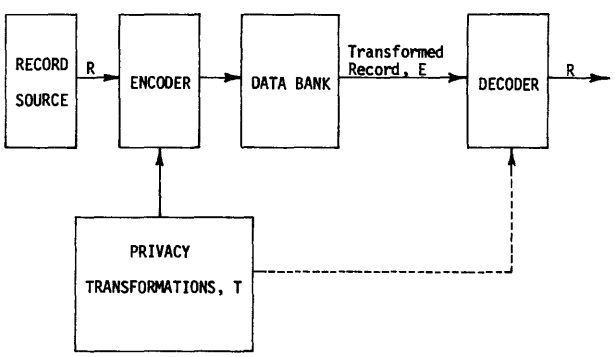


Figure 2—Schematic of information retrieval with privacy transformations

ceived record Y. $I(X; Y)$ is the average of the "information" provided about an event $x \in X$ if an event $y \in Y$ occurs.

The average mutual information is expressed as the difference in the entropy of the source X and the equivocation of the conditional entropy of L given Y,

$$I(X; Y) = H(X) - H(X \mid Y) \tag{2}$$

where $H(X)$ is the entropy, $H(X) = -\sum^x P(x) \log_2 P(x)$, and $H(X \mid Y) = -\sum^x \sum^y P(x, y) \log P(x \mid y)$ is the equivocation of X given Y. From this equation $I(X; Y)$ can be interpreted as the average amount of uncertainty in X, the entropy of X, which is resolved by an observation in the receiver space Y. After the observation in Y, $H(X \mid Y)$ is the average uncertainty still remaining about X.

The mutual information $I(X; Y)$ is measured in bits; it represents the average number of bits of information which can be inferred unambiguously about the X space, given an event in the Y space. The properties of $I(X; Y)$, $H(X)$ and $H(X \mid Y)$ include the following:

1. $I(X; Y)$ is symmetric, i.e., $I(X; Y) = I(Y; X)$.
2. $I(X; Y) \geq 0$, or $H(X) \geq H(X \mid Y)$, where equality holds if and only if X is statistically independent of Y.
3. If X has N states, $H(X) \leq \log_2 N$. $H(X) = \log_2 N$ if and only if the states of X are equally likely.
4. $H(X \mid Y) = 0$ if and only if there is no statistical uncertainty about X, given Y.
5. If X has N states and Y has M states,
   $0 \leq I(X; Y) \leq \text{Min } \{\log_2 N, \log_2 M\}$.
6. If X and Y have the same number N of states, $0 \leq I(X; Y) \leq \log_2 N$, and $I(X; Y)$ is called the *rate*, the average number of bits of information transferred unambiguously from X to Y.

Most of the above properties are easily derived from the definitions of $I(X; Y)$, $H(X)$, and $H(X \mid Y)$. (See, for example, Reference 5 and Reference 7.) Property 5, though not usually provided, is an evident consequence of Eq. (2) and properties 1, 2, and 3. From these properties one can further see that if $I(X; Y)$ is large, the uncertainty of X, given Y, is low and, conversely, if $I(X; Y)$ is small, i.e., near zero, the uncertainty of X, given Y, is great.

Now return to a secrecy system and apply it to the concepts of mutual information. A theorem of Shannon[1] is used to obtain the following theorem:

> *THEOREM.* A necessary and sufficient condition for *perfect* secrecy is that
>
> $$I(R; E) = 0$$
>
> for all R and E. That is, the mutual information conveyed by an enciphered record E about the original record R is zero.

The proof of this theorem is almost immediate by definition (Eq. (2) and Reference 1, Theorem 6, p. 680). Shannon's theorem states, "A necessary and sufficient condition for perfect secrecy is that $P(E \mid R) = P(E)$ for all R and E. That is, $P(E \mid R)$ must be independent of R." By Eq. (2) the mutual information of E given R is

$$I(E; R) = H(E) - H(E \mid R)$$

where

$$H(E) = -\sum_E P(E) \log_2 P(E)$$

and

$$H(E \mid R) = -\sum_E \sum_R P(E, R) \log_2 P(E \mid R).$$

Replacing $P(E \mid R)$ by $P(E)$ in $H(E \mid R)$ yields

$$H(E \mid R) = -\sum_E \sum_R P(E, R) \log_2 P(E)$$

$$= -\sum_E P(E) \log_2 P(E)$$

since

$$\sum_R P(E, R) = P(E).$$

Thus, $H(E) = H(E \mid R)$ and $I(E; R) = I(R; E) = 0$ by symmetry.

The converse follows from property 2 and a reversal of the above steps. Hence, the theorem is proved

Since

$$I(R; E) = H(R) - H(R \mid E),$$

the above theorem implies that perfect security or privacy is obtained if and only if the entropy of the source $H(R)$ is *matched* to the equivocation $H(R \mid E)$. To show how such matching can be accomplished, consider the following example.

Example 1:

The Vernam system for enciphering binary records (Reference 1, p. 662). Let R be encoded into a sequence of binary digits

$$\{x_1, x_2, \ldots, x_N\}$$

where $x_i = 0$ or 1. Let the "running key" be the sequence $n_1, n_2, \ldots, n_N$ of binary digits. Then define the enciphered message E to be the sequence

$$E \equiv \{y_1, y_2, \ldots, y_n\}$$

$$\equiv \{(x_1 + n_1, x_2 + n_2, \ldots, x_N + n_N)\} \bmod 2.$$

This transformation T is reversible since clearly R is obtained uniquely by merely applying the same transformation again, i.e.,

$$R \equiv \{x_1, x_2, \ldots, x_N\}$$

$$\equiv \{(y_1 + n_1, y_2 + n_2, \ldots, y_N + n_N)\} \bmod 2.$$

For simplicity, consider now only the mutual information associated with single letters of the record. Then $X = \{0, 1\}$ and $Y = \{0, 1\}$. Let $P(x = 1) = \alpha$ and $P(x = 0) = 1 - \alpha = \beta$. This is the distribution of the source X. Let $P(n = 1) = p$ and $P(n = 0) = 1 - p = q$. A schematic expressing the transitions from X to Y is shown in Figure 3.

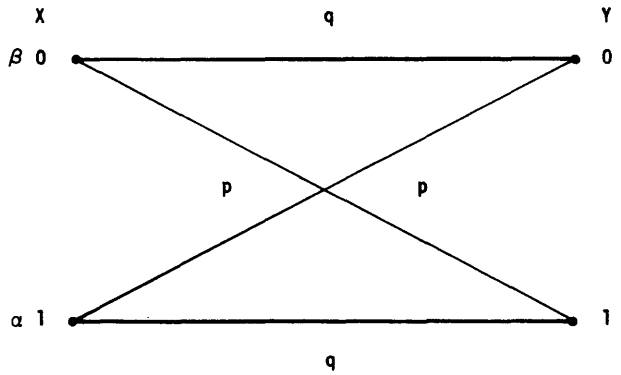


Figure 3—Single letter transition for binary Vernam system

From this

$$P(y=1 \mid x=0)=P(y=0 \mid x=1)=P(n=1)=p$$

$$P(y=1 \mid x=1)=P(y=0 \mid x=1)=P(n=0)=q.$$

Hence,

$$p(y=1)=\alpha q+\beta p$$

$$p(y=0)=\alpha p+\beta q$$

so that

$$H(Y)=-(\alpha q+\beta p)\log_2(\alpha q+\beta p)-(\alpha p+\beta q)\log_2(\alpha q+\beta p)$$

and

$$\begin{aligned}H(Y \mid X) &= -\beta q\log_2 q-\beta p\log_2 p-\alpha p\log_2 p-\alpha q\log_2 q\\ &= -(\alpha+\beta)p\log_2 p-(\alpha+\beta)q\log_2 q\\ &= -p\log_2 p-q\log_2 q.\end{aligned}$$

Thus,

$$I(X;Y)=-(\alpha q+\beta p)\log_2(\alpha q+\beta p)$$
$$-(\alpha p+\beta q)\log_2(\alpha q+\beta p)+p\log_2 p+q\log_2 q \quad (3)$$

is the average mutual information of X, given Y. One can recognize Eq. (3) as the expression for the average rate (bits per symbol) transmitted through a binary symmetric channel (e.g., see Reference 5).

Note that $I(X;Y)$ can be expressed as

$$I(X;Y)=H(\alpha q+\beta p)-H(p)$$

where $H(x)$ is the "entropy" function

$$0\leq -x\log_2 x-(1-x)\log_2(1-x)\leq 1.$$

The point p^* for which $I(X;Y)=0$ obtains by setting

$$p=\alpha q+\beta p.$$

Solving for p, assuming $0<\alpha\leq\frac{1}{2}$, yields $P^*=\frac{1}{2}$. This can also be seen in the graphs of $H(\alpha q+\beta p)$ and $H(p)$, shown in Figure 4.

The above arguments show that the *only* value of p for which the Vernam system yields perfect secrecy is $p=\frac{1}{2}$. This is in agreement with one's intuition: for any other value of p, $I(X,Y)>0$, so that enciphered record still contains information about the original record.

In the next section distortion measures are introduced in order to achieve maximum privacy (although less than perfect) for a given allowable degree of distortion. As mentioned previously, such distortion is designed primarily to hide the identity of the individual that the record is about, yet the distortion is not so great that the record is unusable for statistical purposes.

DISTORTING RECORDS FOR PRIVACY

In this section a study is made of providing privacy transformations to the source of records, given an upper bound to the amount of distortion allowed in the record. The distortion measures defined to quantify the degree of distortion were originally defined by Shannon (Reference 8). Later works include Reference 7 (Chapter 9) and Reference 9 by Gallager and Berger, respectively.

Let i range over the m symbols of the record source, and j range over the n symbols of the received (enciphered) records. Here the symbols of a record source might include letters of the alphabet, digits, given names, surnames, names of geographic locations, religion, etc., and, similarly, for the symbols of the received words. Let the set of such symbols for the record source be denoted by X, and the corresponding set for the received records by Y.

Suppose a symbol i in the original record R is received as j in the received record, E. Assume there is a cost d_{ij} associated with every such input symbol i reproduced as symbol j in the output. If the reproductions are correct, set $d_{ij}=0$. If incorrect, d_{ij} is positive and proportional to the cost of the error. It is convenient to index sets X and Y so that $d_{ij}=0$ if $i=j$, and $d_{ij}>0$ if $i\neq j$. Any cost matrix (d_{ij}) which has these properties is called a *symbol distortion matrix.*

If a record R is distorted by a privacy transformation to a word E, the distortion d which R undergoes is defined to be

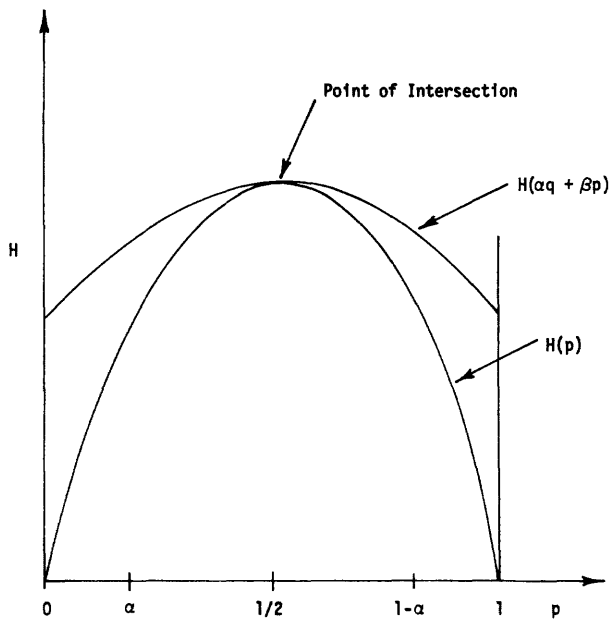


Figure 4 Plots of $H(\alpha q+\beta p)$ and $H(p)$

the average of the distortions of its symbols. That is,

$$D(R, E) = \frac{1}{N} \sum_{k=1}^{N} d_{i_k, j_k} \tag{4}$$

where $i_1, i_2, \ldots, i_N$ are the symbols of record R and $j_1, j_2, \ldots, j_N$ the distorted symbols of E. To obtain the overall system distortion average $d(R, E)$ with respect to the joint probability distribution $P(R, E)$ of R and E,

$$\begin{aligned} D[P(E \mid R)] &= \sum_{R,E} P(R, E) d(R, E) \\ &= \sum_{R,E} P(R)\,(P(E \mid R) d(R, E) \end{aligned} \tag{5}$$

where $d(R, E)$ is defined by Eq. (4) and $P(E \mid R)$ is the conditional probability of E, given record R.

If one minimizes the mutual information $I(R, E)$ over every average distortion $D[P(E \mid R)]$ less than some fixed value, say D^*, the resulting function $R(D^*)$ is called, following Shannon (Reference 7), the *rate distortion function*. In this minimization, only the conditional probabilities $P(E \mid R)$ in Eq. (5) are assumed to vary. Thus,

$$R(D^*) = \min_{D[P(E|R)] \leq D^*} I(R; E) \tag{6}$$

where

$$D[P(E \mid R)] = \sum_{R,E} P(R) P(E \mid R) d(R, E)$$

is the record distortion, defined by Eq. (4) and where $P(R)$, the *a priori* distribution of the records, is held fixed. As defined by Eq. (6), $R(D^*)$ is the *minimum* average mutual information about R conveyed by a knowledge of E, assuming the average distortion is bounded above by D^*. $R(D^*)$ for present purposes is a *measure* of the maximum amount of privacy one might expect if he allowed only a limited amount of distortion of the stored record.

In order to illustrate the utility of the rate distortion function by examples, it is convenient to specialize the record source space R and received record space E to the symbol spaces X and Y, respectively. For the symbol spaces X and Y,

$$R(D^*) = \min_{K[P(j|i)] \leq D^*} I(X; Y) \tag{7}$$

where

$$D[P(j \mid i)] = \sum_{i \in X} \sum_{j \in Y} P(i) P(j \mid i) d_{ij}.$$

Since X and Y are sets of symbols used in records R and E, respectively, $R(D^*)$ can be called the *single symbol* rate distortion function. To see the application of this concept, consider the following example where both X and Y are the two letter set $\{0, 1\}$.

Example 2:

Let X be a binary letter source, the set $\{0, 1\}$ and likewise let $Y = \{0, 1\}$. Suppose $P(X=1) = \alpha$ and $P(x=0) = 1-\alpha = \beta$. This is the source distribution of Example 1 in the preceding section. Let the transition probabilities for distortion be

$$P(y=1 \mid x=0) = P(y=0 \mid x=1) = p$$

$$P(y=1 \mid x=1) = P(y=0 \mid x=0) = q.$$

These transition probabilities and source distribution probabilities are the same as those shown in Fig. 3. The reader can recognize that Figure 3 is the schematic representation of the binary symmetric channel.

The distortion matrix for this example is

$$(d_{ij}) = \begin{pmatrix} d_{00} & d_{01} \\ d_{10} & d_{11} \end{pmatrix} = \begin{pmatrix} 0 & a \\ b & 0 \end{pmatrix}$$

where $a, b > 0$. Then, by Eqs. (5) and (7), the distortion is

$$D = \beta a p + \alpha b p = (\beta a + \alpha b) p = \delta p$$

where

$$\delta = \beta a + \alpha b.$$

Evidently, D is linear in p and ranges from 0 to δ as p changes from 0 to 1. An allowable distortion D^* is chosen from the interval $(0, \delta)$. D^* determines a p^* according to the relation

$$p^* = \frac{1}{\delta} D^*$$

where

$$\delta = \beta a + \alpha b.$$

The mutual information $I(X; Y)$ needed in the present example was computed for Example 1 of the last section. This result, Eq. (3), is

$$I_p(X; Y) = H(\beta p + \alpha q) - H(p)$$

where here $I(X; Y)$ is subscripted to denote its dependence on the transition probability p of distortion.

There are two cases to consider.

Case 1—$0 < p^* < \frac{1}{2}$. This case is shown in Figure 5. From the figure for all p such that $0 \leq p \leq p^*$,

$$I_{p^*}(X; Y) \leq I_p(X; Y).$$

Thus, by the definition of $R(D^*)$ and Eqs. (6) and (7),

$$\begin{aligned} R(D^*) &= I_{D^*/\delta}(X; Y) \\ &= H(\beta D^*/\delta + \alpha[1 - D^*/\delta]) - H(D^*/\delta). \end{aligned}$$

Case 2—$\frac{1}{2} \leq p^* \leq 1$. If $P^* = \frac{1}{2}$,

$$I_{1/2}(X, Y) = R(\tfrac{1}{2}\delta) = 0.$$

By Theorem 1 of the last section, this is the condition of perfect secrecy or privacy. Any further increase in p^*, which is proportional to the allowable distortion D^*, will not lessen the degree of privacy. Thus, $R(D^*) = 0$ for all $p^* \geq \frac{1}{2}$.

Combining Cases 1 and 2, one has

$$\begin{aligned} R(D^*) &= H(\beta D^*/\delta + \alpha[1 - D^*/\delta]) - H(D^*/\delta) && \text{if } 0 \leq D^* \leq \delta/2 \\ &= 0 && \text{if } \delta/2 < D^* \leq \delta \end{aligned} \tag{8}$$

where

$$\delta = \beta a + \alpha b.$$

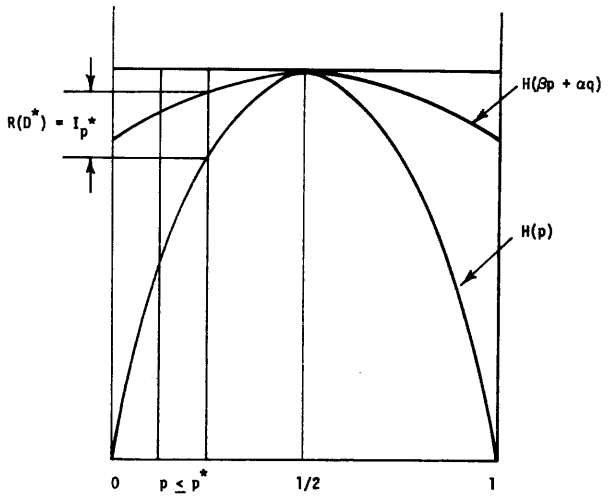


Figure 5—Graphical determination of $R(D^*)$

The privacy transformation to realize the rate distortion function $R(D^*)$, given by Eq. (8), is the Vernam system for enciphering binary records, given in Example 1 of the last section. This follows from the fact that for $0 \leq p \leq \frac{1}{2}$ the equations for $I(X; Y)$ in Eq. (3) and for $R(D^*)$ in Eq. (8) are identical if one sets $p = D^*/\delta$. Thus, if the sequence of "running key" digits $\{n_1, n_2, \ldots, n_N\}$ are chosen to be mutually independent and with probability

$$P(n_k = 1) = D^*/\delta \text{ if } D^*/\delta \leq \tfrac{1}{2}$$
$$= \tfrac{1}{2} \quad \text{if } D^*/\delta \geq \tfrac{1}{2}$$

the amount of mutual information provided by $R(D^*)$ in Eq. (8) will be achieved exactly.

It is of interest to consider briefly here how one might generate "noise-like" sequences such as $\{n_1, n_2, \ldots, n_k\}$ where $q = \text{prob}\ \{n_k = 1\}$ is *less* than one half. If $q = \frac{1}{2}$, a number of feedback shift-register generation methods have been devised. (See, for example, Reference 10 for a study of both linear and nonlinear sequence generators.) Such methods have the advantage over numerical techniques of utilizing a minimum of computer hardware (and also software).

The mathematical bases for the classical linear feedback shift-register is the irreducible polynomial $P_n(x)$ of n-th degree over the Galois field of two elements $GF(2)$. Associated with each such $P_n(x)$ are binary sequences of period no less than $2^n - 1$. Assuming an unknown starting state, it can be shown that $q = \frac{1}{2}$ and that the autocorrelation function of the sequence is such that the digits of the sequence appear to be statistically independent.

Consider the *nonbinary* shift register associated with an irreducible polynomial $P_n(x)$ of n-th degree over the general Galois field $G(p^m)$ or p^m elements where p is a prime. It can be shown that a particular element of the field occurs in the generated sequence with "probability" $q = 1/p^m$ and with period no less than $p^{mn} - 1$. Evidently, these more general sequence generators over Galois fields of p_m elements represent a possible method for generating "running key" sequences $\{n_1, n_2, \ldots, n_k\}$ for privacy distortion where $q = \text{prob}\ \{n_k = 1\} = 1/p^m$.

If sequence generators are used to distort a record and if the initial settings or "key" of the generator is retained for later use, it would be possible for those with the proper need to know to recover the message perfectly (except for possible natural errors). On the other hand, to all others without proper authorization, the retrieved record would appear so distorted or garbled that it would be unfit for any other use, say, than statistical purposes.

Let us now discuss another important method of distortion. This method of distortion is called by some "data aggregation", by others "distortion by coding", and by still others "distortion by data compression". Shannon was motivated by the following type of observation in his development of rate distortion theory (Reference 7): "If an error probability of .3 [the distortion] can be tolerated, a capacity of only .1 bit is necessary and sufficient [a *code replaces* on the average every ten bits by one bit]. If .5 error probability can be tolerated, of course, *no* channel capacity is required. . . ." Clearly, codes which reduce ten bits to one bit accomplish a form of information or data aggregation. Moreover, this is a ten-to-one reduction in the number of bits needed to store the message. Thus, the allowance of distortion yields data compression and vice versa.

Usually data aggregation schemes involve a coarsening of numerical data. The simplest and most widely employed data aggregation scheme for numerical data is quantization. For example, one form of quantization is the mapping of a finite real variable x onto a set of N real values, $y = L_1 < L_2 < \ldots < L_N$, such that

1. For $L_1 \leq x$, $y = L_K$ if and only if $L_K \leq x < L_{K+1}$
2. For $X < L_1$, $y = L_1$.

Evidently quantization is a data aggregation scheme, as well as a data compression scheme, since only $\log_2 N$ bits are required to store the image y of the mapping f. Appropriate distortion matrices for data aggregation methods, such as quantization, and the resulting rate distortion functions will be considered elsewhere. For the present, discussion of data aggregation will be limited to the following simple example of bit reduction by coding.

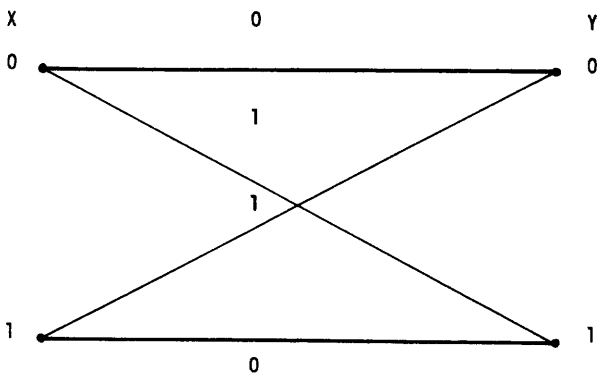


Figure 6—Line diagram of distortion matrix (d_{ij})

Example 3:

Here, binary repetition codes of odd block length n are used to aggregate n bits of record into one bit. As in Examples 1 and 2, let R be encoded into a sequence of binary digits $\{x_1, x_2, \ldots, x_N\}$ where $x_k = 0$ or 1. Again, suppose $P\{X=1\} = \alpha$ and $P(x=0) = 1-\alpha = \beta$ the distribution of the source.

Specialize the distortion matrix, used in Example 2, by letting $a=b=1$. A line diagram of this distortion matrix is shown in Figure 6.

This assumes the costs of both types of errors, transitions from 0→1 or 1→0, are the same. The average letter distortion is, in this special case,

$$D = \beta p + \alpha p = (\alpha+\beta)p = p.$$

Evidently the distortion in this case is *identical* to error probability per binary digit produced by the distortion.

Let $n=2m+1$ and suppose $x_1, x_2, \ldots, x_n$ are n binary digits of the record. A binary repetition code is a mapping of the set of such n-triples, namely $\{0, 1\}^n$, into the two-element set $\{0, 1\}$ according to the following rule:

$$z=1 \text{ if } \sum_{k=1}^{2m+1} x_k \geq m+1$$

$$=0 \text{ if } \sum_{k=1}^{2m+1} x_k < m+1$$

where $\{0, 1\}^n$ denotes the Cartesian product of the two-element set $\{0, 1\}$, n times, i.e.,

$$\{0, 1\}^n = \{0, 1\}x\{0, 1\}x \ldots \{0, 1\} \qquad n \text{ times}$$

A device which performs the mapping $g(x)$ is called a *coder* (Reference 8, p. 9, 108).

To reproduce a distorted facsimile of the original record sequence $\{x_1, x_2, \ldots, x_N\}$ one runs the output of the coder, sequence $z_1, z_2, \ldots, z_T$, into another device called the *reproducer*. The reproducer realizes the following one-to-one mapping $y=f(z)$ of $\{0, 1\}$ into $\{0, 1\}^n$:

$$y = (0, 0 \ldots 0) \text{ if } z=0$$

$$= (1, 1 \ldots 1) \text{ if } z=1.$$

The composition

$$y = f[g(x)]$$

maps the record sequence $\{x_1, x_2, \ldots, x_N\}$ into a distorted version $\{y_1, y_2, \ldots, y_N\}$ of the same sequence at the output of the reproducer. The sequence $\{y_1, y_2, \ldots, y_N\}$ is the n bits of output record E, corresponding to the n-bits $\{x_1, x_2, \ldots, x_N\}$ of the record before encoding.

If one assumes that the letters of the source are equiprobable, that is,

$$\alpha = \beta = \tfrac{1}{2},$$

it is a simple matter to compute the distortion exactly for this example. For this,

$$D = \text{Prob } \{y_1=1 \mid x_1=0\} = \text{Prob} \left\{ \sum_{j=2}^{2m+1} x_j \geq m+1 \right\}$$

$$= \frac{1}{2^{2m}} \sum_{k=m+1}^{2m} \binom{2m}{k}$$

$$= \frac{1}{2}\left[1 - \frac{1}{2^{2m}}\binom{2m}{m}\right]$$

where $\binom{2m}{m}$ is a binomial coefficient.

To illustrate let $n=5$, then

$$D = p = 5/16.$$

The equivalent rate, number of bits used per record letter, is 1/5. It is of interest to note that this rate is somewhat higher than would be required by the Vernam system in Example 2. If one uses $D^* = 5/16$ in Eq. (8)

$$R(5/16) = 1 - H(5/16) \approx 0.1,$$

approximately half the rate achieved by the simple repetition code, above, for $n=5$.

REFERENCES

1. Shannon, C. E., "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, 1949.
2. Peterson, W. W., "Addressing for Random-Access Storage," *IBM Journal*, April 1957.
3. Morris, R., "Scatter Storage Techniques," *Communications of the ACM*, Vol. 2, No. 1, January 1963.
4. Kleinrock, L., *Communications Nets*, McGraw-Hill, New York, 1964.
5. Shannon, C. E., and Weaver W., *Mathematical Theory of Communication*, University of Illinois Press, Urbana, Illinois, 1949.
6. Khinchin, A. I., *Mathematical Foundations of Information Theory*, Dover Publications, New York, 1957.
7. Gallager, R. C., *Information Theory and Reliable Communication*, John Wiley & Sons, New York, 1968.
8. Shannon, C. E., "Coding Theorems for a Discrete Source with a Fidelity Criterion," *Information and Decision Processes*, R. E. Machol (ed.), McGraw-Hill Book Co., New York, 1960, pp. 93-126.
9. Berger, T., *Rate Distortion Theory*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.
10. Reed, I. S., and R. Turn, "A Generalization of Shift-Register Sequence Generators," *J. of the Assoc. for Computing Machinery*, Vol. 16, No. 3, July 1969, pp. 461-473.

Privacy transformations for databank systems*

by REIN TURN

The Rand Corporation
Santa Monica, California

INTRODUCTION

The term *databank* implies a centralized collection of data to which a number of users have access. A computerized *databank system* consists of the data files, the associated computer facility, a management structure, and a user community. Several classes of databank systems can be defined on the basis of the nature of the organization supported by the databank, and its activity; the nature of the data and its uses; and the structure of the associated computer facility. Such classifications have been discussed in detail elsewhere.[1]

The recent years have seen a steady increase in the establishment of databanks in all sectors of our society in the United States,[2] as well as in other countries:[3,4] in the federal, state and local governments for administrative, law enforcement, education, social welfare, health care purposes; in business and industry for supporting management, planning, marketing, manufacturing and research; in universities for administrative purposes and for supporting social research projects; and the like.

The information maintained in such databank systems includes proprietary data on the operations of industrial concerns, sales data of business establishments, and large collections of personal information on individuals. In all databank systems there is a need to control the access to the data, if for no other purpose than, at least, to assure the *integrity* of the data—that they will not be accidentally modified or erased. In many databanks containing proprietary business information, classified defense information, or confidential personal information on individuals, there is a requirement for *data security*—protection against accidental or deliberate destruction, and unauthorized access, modification or dissemination of the data.

In databanks maintaining personal information on individuals, often collected without the consent or knowledge of the persons concerned, the questions of potential violations of an individual's *right of privacy*—his right to determine for himself what personal information to share with others, arise. These, however, relate to what personal information is gathered in the first place, and thus are legal, political and ethical questions, rather than the technical questions of data security which are addressed in this paper.

Privacy transformations** represent one technique for providing data security—the mathematical/logical transformation of the protected data into forms which are unintelligible to all but the holders of the "keys" to the transformations, i.e., those who know what *inverse* transformations to apply. This capability of privacy transformations is very useful for providing data protection beyond the more conventional access control mechanisms, such as passwords in their various forms, which can be circumvented or nullified through flaws in software, wiretapping, or outright physical theft of data-carrying, demountable storage media.[5]

This paper will first briefly review the relevant characteristics of several classes of privacy transformations, then present a set of suitability criteria for databank applications, and conclude with a discussion of implementation and operational considerations.

PRIVACY AND TRANSFORMATIONS

Historically, there has always existed a requirement to prevent access to information in a message when outside of the physical control of either the originator or the intended receiver, i.e., when the message is in some communication channel. Indeed, certain classes of messages have always been subject to interception, copying, and attempts to uncover the information they contain.[6] In the computer age this threat is also extended to stored messages and data.

Shannon[7] refers to the methods of protecting information in messages and data as *secrecy systems*. There are two kinds:

- *Concealment systems* where the existence of a message is hidden, such as in the case of using invisible ink, or mixing a message with other, unrelated text.

* The research reported in this paper was supported by the National Science Foundation Grant No. GI-29943. However, any views or conclusions contained in this paper should not be interpreted as representing the official position of the National Science Foundation or The Rand Corporation.

** The term "privacy transformation" is synonymous with "cryptographic transformation". It was coined in the early days of computer security research[5] to distinguish the use of cryptographic techniques in civilian and commercial systems from their use for protecting classified national defense information.

- "True" *secrecy systems* where the existence of a message is not hidden, but its *meaning* is concealed by the use of privacy transformations—encryption techniques.

In the following, only the "true" secrecy systems are considered, since concealment systems are not applicable to computerized databank systems—no one can assert that there are no data in such systems, although whether or not there is information worth protecting may be debatable.

A privacy transformation is a mapping $T(K)$, from the space $RS(A)$ of all possible records of finite length which are composed of symbols from a finite alphabet, A, according to the vocabulary, syntax, and grammar of a natural or artificial language, L, into the space $ES(B)$ of strings of characters from an alphabet B. The original, untransformed record, R, is called the "plaintext" and its equivalent transformed character string, E, the "ciphertext" or a "cryptogram." The transformation, $T(K)$, is usually a member of a large space, TS, of similar transformations. The set of parameters, K, of the transformation $T(K)$ are called the "key" which selects $T(K)$ out of the space TS.

Several classes of privacy transformations exist and are in use. A major classification criterion is the nature of the mapping T itself: it may be *irreversible* (i.e., many-to-one) mapping of records into ciphertext strings, or a one-to-one mapping with a unique inverse, T^{-1}. Both classes of privacy transformations find applications in protecting confidentiality and security in databank systems.

Irreversible privacy transformations

A many-to-one privacy transformation, T, when applied to record space $RS(A)$, may convert more than one record into the same ciphertext string E in the space $ES(B)$. That is, given E and the knowledge of the exact transformation used, an uncertainty remains which of the possible records was transformed into E. Unless the intended receivers possess additional contextual information for resolving the uncertainty, many-to-one transformations are inappropriate for precise communication of storage of information.

However, there are situations in databank systems where the maintenance of the original level of information content is not required or could be reduced in the interest of protecting the confidentiality of the information. For example, statistical databank systems, such as the U.S. Bureau of Census and various social sciences research projects, collect data on individuals under the authority of a law or with the individuals' voluntary participation. The data, especially in certain social sciences research projects, may be very sensitive and may lead to considerable harm to some individuals if disclosed.[8] The threats to the confidentiality of such data include legal means—subpoenae issued by courts, grand juries, and investigative committees with subpoena power.[9]

Irreversible privacy transformations can be used in such databank systems to hide personal characteristics of individuals in the group characteristics, and by reducing the credibility of the information. The following can be used:[10,11,12]

- *Aggregation.* The irreversible transformation T applied to a group of data records computes the averages of various data elements in the records and, in each record of the group, replaces the original data elements with the group averages. As the size of the aggregated group of records is increased, the transformation increases the uncertainty about the original information in the records.
- *Random modification.* The transformation consists of adding a randomly varying component to the original information in the records, thereby introducing errors. If the random variables are produced by a process whose statistical characteristics are properly chosen, the statistical value of the modified records are not altered, but credibility of each individual record is now reduced and along with this, the value of such record as incriminating evidence against an individual.

A prerequisite for effective use of the above classes of irreversible privacy transformation is, of course, the original, untransformed records be totally removed from the databank. The price paid for increased data confidentiality is, however, a reduction of the future statistical utility of the data—it will not be possible to make new, precise correlation analyses between various characteristics of individuals (these have been aggregated or innoculated with errors) or to make longitudinal analyses—studies of changes in persons' characteristics or attitudes over periods of time. The confidentiality protection vs. data utility tradeoff is an important question which is still being studied.

Reversible privacy transformations

Transformations in this class are those which are usually discussed as "cryptographic transformations"—the one-to-one mappings from the record space $R(A)$ into the ciphertext space $ES(B)$ which have unique inverses. The protection provided to the data rests in keeping the key, K, of the transformation $T(K)$ from falling into unauthorized hands, and in the expectation that the recovery of original records or the key from the ciphertext forms is a task beyond the resources and know-how of the potential interceptors.

Further classification of reversible privacy transformations, henceforth simply "privacy transformations," can be made on the basis of the mathematical or logical operations involved in applying the transformation. Four principal classes of privacy transformations used in databank systems—coding, compression, substitution and transposition, are briefly discussed below. More detailed discussions can be found in the literature.[6,13,14]

Coding

Coding is a transformation where an entire record, parts of it, words, or syllables of the language L_i used in the record space $RS(A)$ are replaced with words or groups of characters of some other (usually artificial) language L_j.[6,15] A coding transformation and its inverse are usually applied with the help of a coding dictionary (code book) or by using table look-up methods. The protection afforded depends on maintaining control over the code books and in frequent changes of codes. Besides providing confidentiality protection, coding can also provide a considerable degree of data compression in transmission or storage. The resulting economy is a main reason for the widespread use of codes in computer files.

Compression

Data compression transformations are used to reduce the redundancy in stored or transmitted data by removing repeated consecutive characters—blanks or alphanumerics, from the records. Other types of data compression transformations attempt to achieve more compact storage of records by "packing" more characters into the storage space normally occupied by a single character. The resultant, compacted data files contain records which have been distorted by the compression algorithms and which will be largely unintelligible when accessed with normal utility programs in the databank. For correct retrieval, decompression algorithms must be applied. Even though data compression is applied mainly to achieve storage or transmission time economies, the associated confidentiality protection may also be sufficient in mild threat environments.

Substitution

Substitution transformations replace single characters or groups of characters of the alphabet A_i of language L used in the record space $RS(A_i)$, with characters or groups of characters of some other alphabet B (or set of alphabets $B_1, \cdots, B_M$). That is, the transformed record is still-composed in language L, but transmitted or stored using alphabet B. Replacement of characters of English alphabet with six-bit binary codes is a very simple substitution transformation. The key K of the transformation $T(K)$ specifies a particular substitution correspondence. The protection obtained depends, in addition to protecting the key, on the number of possible substitution correspondences between alphabets A_i and B (i.e., the size of the key space) and the nature of the language L.

Substitution transformations can be subclassified as *monoalphabetic* and *polyalphabetic*. Each of these could be *monographic* and *polygraphic*. The latter classification refers to number of characters that are being substituted as a group: in monographic substitutions, single characters are substituted (independently of each other and the context of the message) with single characters (or groups of characters). In polygraphic substitutions groups of two or more characters are substituted by similar (or larger) groups.

- *Monoalphabetic substitution.* An alphabet B is chosen to correspond with the original alphabet A such that to each character in A corresponds a unique character (group of characters) in B. As will be discussed later, monoalphabetic substitutions leave the basic language statistics (average character frequency, average polygram frequencies) invariant and, thus, remain susceptible to basic cryptanalytic techniques.
- *Polyalphabetic substitution.* Here the alphabet B is actually a set of alphabets $B_1, B_2, \cdots, B_M$ which are used cyclically with period M. For example, in a monographic M-alphabetic substitution, the first character, r_1, of record R is substituted with a character of alphabet B_1, the second with a character from B_2, the M-th with a character from B_M, and the next character again from B_1. The effect of a polyalphabetic substitution is to hide the original characteristics of the language L, since a given character of alphabet A may now be transformed into M different characters of alphabets $B_1, \cdots, B_M$.

It is common to derive the alphabet B from alphabet A by making a permutation of the characters of A to correspond with the original characters. The simplest such permutation is a cyclic shift of the characters of A by a fixed number of characters, μ. This class of substitution transformations is called "Caesar ciphers." They are extremely simple to solve as, in the case of the English alphabets, a maximum of 25 trials are required to discover the "key," the number of characters that alphabet A was shifted to obtain alphabet B.

A polyalphabetic substitution transformation using M Caesar ciphers as the alphabets $B_1, \cdots, B_M$ (with repetition allowed, i.e., $B_i = B_j$, for some i and j, for several such pairs) is called a "Vigenere cipher." The key is now a set of M numbers which specify the shifts used to generate from alphabet A the alphabets $B_1, \cdots, B_M$. A special case of the Vigenere transformation is the situation where the number of alphabets, M, is larger than the number of characters in a set of records to be transformed. This transformation is called the "Vernam cipher" and it can provide a very high level of protection.[7]

Substitution transformations may be implemented in several ways. Table look-up operations are used for substitutions with alphabets B_i that are arbitrary permutations of the alphabet A. Certain *algebraic* operations, however, permit relatively simple computation of the required substitutions.[14,16,17,18,19]

In algebraic substitutions, the N_A characters of the alphabet A are set in a correspondence with the positive integers $0,1, \cdots, N_A - 1$ (for example, $a=0$, $b \leqq 1$, $\cdots$, $y=25$ in the English alphabet). These form an algebraic ring under the operations of addition module (N_A) and

subtraction module (N_A). Then, choosing an integer k in the range 0 to N_A-1 specifies a particular substitution transformation of characters r_i of records R into characters e_i of the transformed version, E, of R:

$$e_i=r_i+k(\text{mod } N_A),$$

and the inverse transformation

$$r_i=e_i-k \text{ (mod } N_A).$$

For polyalphabetic substitutions, a sequence of integers $k, k_0, \cdots, k_{M-1}$, are used cyclically:

$$e_i=r_i+k_j \text{ (mod } N_A), j=i \text{ (mod } M)$$

Polygraphic substitutions of n-character groups (n-grams) by other n-grams can be represented as sets of simultaneous linear congruences and computed by matrix operations.[16,17]

$$e_i=\sum_{j=1}^{n} c_{ij}r_j, i=1, \cdots, n$$

where the elements c_{ij} of the matrix C are selected among integers in the range $0, \cdots, N_A-1$, such that the matrix C has an inverse. If the matrix C is fixed, the substitution is monoalphabetic (in terms of n-grams). Polyalphabetic n-gram substitutions are obtained by introducing a cyclically varying parameter, t, in the matrix C.[18] The matrix $C(t)$ must have the property that its determinant is independent of the parameter t, and is a prime number modulo (N_A).

Transposition

Privacy transformations that permute the ordering of characters in the original message are called transposition transformations. The transformation may be applied to the entire message all at once, or on a block-by-block basis. The alphabet of the message remains unchanged. A common method for implementing a transposition is to write the block to be transformed in a matrix form following some rule and then rewrite in linear form using a different rule. For example, the message may be written first as rows of the matrix and then transcribed by taking the column of the matrix in some specified order.

Transposition transformations retain the character frequency statistics of the language but destroy the higher order statistics (polygram frequencies).

Composite transformations

The effectiveness of privacy transformations can be increased (although not always) by applying a sequence of transformations, $T_1(K_1), T_2(K_2), \cdots, T_S(K_S)$, such that $E=RT_1T_2\cdots T_S$. Typically, the transformation T_i are either all substitutions, all transpositions, or a mix of these.

The case where all transformations are substitutions is called an S-loop substitution transformation:[20]

$$e_i=r_i+k_{j_1}+k_{j_2}+\cdots+K_{jS} \text{ (modulo } N_A)$$

where $j_g=1 \text{ (mod } M_g), g=1, \cdots, S$. If the periods of the polyalphabetic transformations $T_1, \cdots, T_S$, are mutually prime, the period of the composite transformation $T=T_1\cdots T_S$ is the product of periods $M_1, \cdots, M_S$ of the component transformations.

A particularly effective composite transformation suggested by Shannon[7] is a "mixing transformation" which may consist of a sequence of n-gram substitutions and transpositions. Such mixing transformations can be highly effective in hiding the language characteristics, as well as possibly information in the ciphertext E of the nature of privacy transformations used.

SUITABILITY CRITERIA

Among a set of requirements stated by Kerckhoffs some seventy years ago[7] are:

- The cryptographic transformations used should be, if not theoretically unbreakable, unbreakable in practice;
- A knowledge by enemy of system's hardware should not compromise the protection provided to the messages;
- The key should be able to provide all the protection, it should be easily changeable;
- The application of the transformation should be simple, requiring neither complicated rules nor mental strain.

Kerckhoffs' requirements were derived for manually operated communication systems, but are also applicable in modern communication systems and computerized databanks.

The suitability of a particular class of privacy transformations for application in a communication network or in the files of a databank depends on: (1) the relevant characteristics of the particular application, (2) the inherent characteristics of the class of privacy transformations used, and (3) the technical aspects of the system that implements the application and the privacy transformation. Although the principal purpose of using privacy transformations is to provide security to information in transit or in storage, the effects of application of transformation to the utility of the system are equally important —a system may be designed to provide excellent security, but at such a cost in loss of performance that it may become useless.

Application characteristics

The characteristics that affect the effectiveness of a candidate class of privacy transformations in protecting information include the following:

a. *Value of the information.* Whether or not the value of information can be determined adequately depends largely on the nature of information involved. The most difficult to assess is personal information, the easiest to assess is business information. Information affecting national security is usually treated as invaluable and any cost in its protection is considered justifiable. Important is also the time dependency of the assessed value, and this has a direct bearing on the suitability of a class of privacy transformations. For example, if the transformations can resist a cryptoanalytic effort of reasonable intensity for T hours, and the value of the protected information is expected to decrease below a critical threshold in less than this time, the transformation will provide sufficient protection. Determination of the value of information is discussed in more detail in Section V of this progress report.
b. *Language(s) used.* The information to be protected by a privacy transformation is carried in the words of the message (or computer record) and is inextricably identified with these words and the language that provides the vocabulary, grammar, and syntax for embedding the information into the message. In natural languages the vocabulary, grammar, and syntax have evolved over periods of time with no regard to the possible application of privacy transformations. In artificial languages the need to provide protection through the use of privacy transformations can be taken into account already in the language design phase.
c. *Dimensions of the application.* The static and dynamic aspects of the application—the ranges of volumes of messages or data to be stored, processed, and/or transmitted; the required rates and maximum allowed time for operating on a message or data record; and the nature of the processing (sequential, random access, concurrent, etc.), establish criteria which must be satisfied in implementing the privacy transformation.
d. *The personnel characteristics* that affect their role in the application, control, safeguarding of the privacy transformation system: level of expertise, integrity, discipline, etc. Errors made will require repetition of processing or transmissions in providing more intercepted material for the cryptanalyst.

Inherent characteristics of privacy transformations

The most important overall criterion in selecting a privacy transformation is the *amount of security* that it can provide. In general, security appears similar to reliability—both are concerned with techniques for assuring proper operation of systems, and both require *a priori* prediction of the probability of proper operation. From the point of view of a particular implementation, however, reliable operation is a prerequisite of secure operation.

The inherent characteristics of privacy transformations which affect the amount of security provided, and the effective operation of the application process, include:

a. *Size (cardinality) of the key space.* The protection provided by privacy transformations depends on the intruder's uncertainty concerning the transformation used. In general, it must be assumed that the intruder knows the particular class of transformations, but does not know the specific set of key parameters employed. For example, the transformation may be a monoalphabetic substitution, but which one? There are 26! possible permutations of the English alphabet (although not all are permissible, such as the permutation that changes only two letters and leaves the rest the same). A large space of *permissible* keys, each selected with the same *a priori* probability is a prerequisite for any effective secrecy system.
b. *Effect on language.* A privacy transformation provides protection by drastically altering the appearance of the plaintext record (or computer record). Ideally, all the characteristics of the source language (the plaintext language) are altered and made unrecognizable. The extent to which this is achieved is one measure of the suitability of the transformations. For example, a simple (monoalphabetic) substitution is not very effective for languages that have prominent differences in the average frequencies of characters. On the other hand, simple substitution may be quite effective in enciphering numeric-data where all numerals are essentially equally likely.
c. *Complexity.* The complexity of the privacy transformation may contribute to the amount of security by providing more complete scrambling of the language characteristics, but it also contributes to the cost in its application: in computer data banks where transformations are applied by software techniques (programs) complexity translated directly into computer time used for nonproductive (from the point of view of the application) operations.
d. *Effects on dimensions.* Certain privacy transformations involving substitution of characters or polygrams with higher order polygrams (e.g., every character replaced by a pair of characters; a digram replaced by a trigram) increase the length of the ciphertext message compared to the plaintext message. This increases the transmission time or storage space required. Coding transformations, however, can be designed to reduce these requirements.
e. *Error susceptibility.* Compound transformations and super encryptions that involve several transformations applied sequentially may have very undesirable error propagation properties. Lease susceptible are monographic substitutions where error in applying the transformation to a character affects

only that character and does not propagate. However, substitutions that use the ciphertext itself as the key (with appropriate translation by a few characters) are extremely susceptible to error propagation.

f. *Length of the key.* The concept of a "key" to a privacy transformation T is often used in two senses. In the case of polyalphabetic substitutions, for example, the sequence of numbers k_i added to the corresponding message characters, n_i, is called the "key." The length, N, of this sequence $k_1, \cdots, k_N$ is the "key length"; it corresponds to the period in the use of different alphabets. For the cryptanalyst who attempts to discover the key by studying intercepted ciphertext messages, longer keys mean more unknowns that must be determined, hence, providing more protection to the information. In certain implementations, however, the key sequence is produced by a computational process which is specified by only a few parameters. Here the "key" that selects the privacy transformation (i.e., the production of the sequence applied to the plaintext) is the set of parameters, rather than the sequence produced. If the cryptanalyst can attempt to solve for the parameters of this process, rather than the entire sequence produced by the process, his number of unknowns is greatly reduced. An example of this is the generation of random numbers $X_i = AX_{i-1} + B \pmod N$. A large number of X_i are produced, but there are only three unknowns: A, B, and X_0.

The various characteristics listed above are evaluated for the different classes of privacy transformations in a following section.

System implementation characteristics

The third set of characteristics that determines the suitability of a particular class of privacy transformations is associated with the system implementation of the application.

a. *Processing capability.* The processing speed of the system and the storage capacity. Availability of instructions for easy application of the privacy transformations. Availability of hardware devices or software programs. Capability to use suitable file structures.
b. *Error environment.* The error characteristics of the communication channel, or the storage medium. The availability of error detecting/correcting codes.
c. *Security environment.* The capability to provide for the security of the keys for the privacy transformations, and to protect the information in the enciphering and deciphering processes.
d. *System personnel.* These may be the same as the applications personnel. Also included are the operators, programmers, maintenance engineers of the system. Their expertise in operating the system, as well as their integrity has an important role in making the use of privacy transformations a success.

Language characteristics

As stated previously, information is communicated by using a language. Concealment of the information in a written record (on any medium such as paper, magnetic surface, electronic circuitry, etc.) through the use of privacy transformations requires that the message is transformed in such a way that any resemblance with the original form is obliterated.

Natural Languages. Investigations of the structures of natural languages[21,22] have shown that there are a number of structural and statistical characteristics of their vocabularies that, in normal usage, are relatively insensitive of the context and can be used to identify the particular language used:

a. *Single character (monograph) frequency distribution*—there is a large difference in the usage of letters in the vocabularies of natural languages. For example, on the average the letter "e" appears 100 times more often than the letter "q"; in French the letter "q" occurs 11 times as often as in English. Special vocabularies, such as family names of persons or tactical orders.
b. *Polygram frequency distribution.* The data here is normally limited to pairs of characters (diagrams) which show transitions of letters to other letters in the word structure, and triplets of characters (trigrams). For example, the two most frequent diagrams in English are "th" and "he," but "es" and "en" in French and Spanish. The two most frequent English trigrams are "the" and "ing," in French they are "ent" and "que."
c. *Starting and terminal letter frequencies.* These differ sharply from the general letter frequency distribution. For example, the letter "e" (most frequent in the general distribution) ranks 14 as a starting letter, and first as a terminal letter. The letters "v," "q," and "j" have extremely low frequencies as terminal letters. Proper names, in general, have different starting and terminal letter frequencies.
d. *Word usage frequencies.* Word frequency distributions are much more dependent on the particular application areas than the various polygraph frequencies. The first ranking words, however, tend to be prepositions and connectives which are used in the same manner in all application areas. For example, the first nine are: the, of, and, to, a, in, that, is, was. The word frequency distributions form the basis of the so-called "probable word" method of cryptanalysis.

TABLE I—Effects of Classes of Privacy Transformations on Language Characteristics

Characteristic	Substitutions				Transposition	Composite
	Monoalphabetic		Polyalphabetic			
	Simple	m-graphic	Simple	k-graphic		
Single character frequency	Invariant (changed alphabet)	Changed	Changed*	Changed*	Invariant	Changed**
k-gram frequency distribution	Invariant	Changed***	Changed	Changed	Invariant	Changed
Word frequency distribution	Invariant (within the new alphabet)		Changed	Changed	Changed	Changed**
Pattern word structures	Invariant	Changed	Changed	Changed	Changed	Changed
Syntactic structure	Invariant	Partly Changed	Invariant	Partly Changed	Changed	Changed

* Within the period of applying alphabetic transformation; over large numbers of periods, some of the characteristics may show invariance.
** Assuming a composite of transpositions and polyalphabetic substitutions.
*** Changed for certain values of k in k-grams (e.g., for k not a divisor of m).

e. *Word structure patterns* (isomorphisms). There are groups of words which have similar patterns of letter occurrences in the word (e.g., aDDeD, sEEmEd, have the pattern -xx-x-). This structural information can be used to place words in "congruence" classes, and the classes can be used in cryptanalysis.
f. *Word length frequencies*. This information also characterizes different languages and, on occasion, application areas. For example, the mean word length in English is 4.5, but 5.9 in German.

Various other statistics about word structure, word-to-word transitions, etc. can be derived. Their utility from cryptanalytic point of view depends on the specific application area. Table I presents an assessment of the effects of privacy transformations on language characteristics.

The statistical structure of a language provides a certain degree of predictability in constructing words in that language. This predictability can be measured in terms of *redundancy*—the inefficiency in the use of the available character sequences from a given alphabet as words of the language. For example, a redundancy of .75 indicates that 75 percent of the possible character-sequences (up to some relatively small length) are not used as words. In general, languages with high redundancy require more complex privacy transformations than those with low redundancy.

The sentence structure and the rules of proper usage, *syntax and grammar*, likewise, place constraints in the formation of strings of words as sentences in the message. The more rigid the syntactical and grammatical requirements imposed on the message source, the more complex privacy transformations are required to effectively diffuse the structure and increase the uncertainty of the cryptanalyst.

Artificial Languages. Application of privacy transformations to information in computerized retrieval systems involves working with so-called artificial languages (e.g., codes, query languages, and programming languages) and data. These differ significantly from the natural languages and can be expected to influence the protective effectiveness of privacy transformations in different ways.

Four levels of artificial languages can be recognized. Starting with the level most similar to a natural language there are:

a. *Query languages*. These are languages designed for user interaction with the retrieval system—to request information, choose processing options, etc. For easy interaction with the system the vocabulary of a query language statement available to the user is usually a restricted subset of natural language words, arranged with precisely specified structure in natural language sentences. For example, a request may be stated RETRIEVE ALL NAMES (ENGINEER, CALF, AGE: 30-50). Many query languages provide menus of operations that are allowed. Here the wording of the choices is, likewise, kept relatively brief. Query language statements are used mainly in communication channels linking terminals with the retrieval system computers.
b. *Higher Order Programming Languages*. Programs written in higher order languages such as FORTRAN, PL/1, ALGOL, etc. may require privacy transformations if they are considered sufficiently valuable (such as certain proprietary programs) and stored in computer accessible form. Programming languages have a *fixed vocabulary* of words selected from the natural language to specify the program structure and designate dataprocessing operations (e.g., EQUIVALENCE, DIMENSION, DO, READ, WRITE, etc.) and an open-ended *variable vocabulary* specified by the programmer for variable names, numerical values, arithmetic logi-

cal processing statements, and such. The choice of some of these words is subjective with the programmer. Often these are similar to words in the natural language (e.g., ICOUNT, JSET, II, AVALUE=BVALUE(J)+INDEX, etc.). The *character set* of a typical higher-order programming language includes many special characters (PL/1, for example, has a 60-character alphabet). The *syntactical and grammatical* rules are very rigid and must be precisely followed.

c. *Assembly languages.* An intermediate step from the higher order language designed for increasing programming ease to efficient computer-executable form—the "machine language," is an assembly language. It is obtained from a higher-order language through a compilation process. The vocabulary of an assembly language consists of mnemonic names for the instruction set of the computer (usually two- or three-letter groups) and the variable names specified in the higher order language. The format is quite rigid. Programs are sometimes stored in the assembly language form.
d. *Machine language.* A machine language program is composed of instruction codes, constant numerical values, and addresses. All of these are coded as binary numbers. The instruction words are divided into fixed length fields that contain the different codes. The sets of allowed code numbers for the various fixed fields may have different cardinalities. No resemblance with a natural language is left. The alphabet consists of binary numbers with the ranges of values specified by the field lengths. Operating programs are usually stored in the machine language form.
e. *Interpretive languages.* In some interactive computer systems programs are stored in a higher-order language only in the execution phase (e.g., the JOSS language). For this, a dictionary and various analysis programs are maintained and used. The characteristics of higher-order languages discussed above are also typical of interpretive languages.

The statistics of higher order artificial languages tend to reflect the statistics of the underlying natural language, but this similarity decreases in assembly languages, and is essentially nonexistent in machine language.

The overall effect of the limited fixed vocabulary, large character set, rigid structure and lack of syntactic ambiguity is a reduction of the effectiveness of applying privacy transformation. On the other hand, the availability of the variable vocabulary can be used to change the statistical characteristics of the language almost at will.

Data. The principal use of privacy transformations in retrieval systems can be expected to center about protection of data, both in storage and in transit. Certain categories of personal information, in particular, require a degree of confidentiality sufficiently high to warrant the use of privacy transformation.

In general, personal information records consist of the following parts:

a. Person's name, address, and other identifying characteristics. In some data files the name and address may be replaced by a code number, where the name/address and code number correspondences are maintained in some dictionary. The name and address, if included, can be expected to be in the natural language. Other characteristics may be coded.
b. General descriptive information, a mixture of proper names (e.g., the birth place, parents), codes, and numeric information.
c. Narrative information. A mixture of natural language sentences, abbreviations, and codes (e.g., the description of a person's criminal history).

The inclusion of names, abbreviations, and numerical codes can be expected to considerably change the statistics of personal data as compared with the natural language text. In particular, it may be expected that the occurrence of proper names which have no identical natural language words will tend to "flatten out" the single letter and polygram frequencies.

In records with fixed formats (i.e., where fixed length fields are provided for names, addresses, etc.) the "blank" characters will have a relative high frequency of occurrence (just as in numeric data, zeroes will be the most frequent numerals). Sorting of the files into alphabetic or numerical order, likewise, in a structural feature of data files that can weaken the effectiveness of privacy transformations.

EFFECTIVENESS AND COST

The two most important considerations in selecting a class of privacy transformations for implementation in a databank system are the *effectiveness* of the transformations in providing data security and the initial and recurring *costs* of providing this protection. These must be weighed against the estimated *value* of the protected information in order to implement a *rational* protection system—one that provides a level of data security warranted by the value of the protected information.[1]

Effectiveness measures

The effectiveness of privacy transformations is usually discussed in terms of the resources and expertise required by the "enemy" cryptanalyst to "break" the privacy transformation used, i.e., to discover the key. The following assumptions about the intruder cryptanalyst must be made:

- He knows in detail the class of transformations being used; the language (vocabulary, syntax, grammar)

used in the records or programs; the general subject matter of the data. He does not know the specific key of the privacy transformations used or the exact contents of protected records, although he may know some words that are highly likely to occur.
- He is knowledgeable in computer technology, operation and use; knowledgeable in the operational procedures of the target databank; and has a digital computer at his disposal.

A necessary prerequisite for attempting to break a privacy transformation system is the availability of a sufficient amount of ciphertext. The minimum amount required for unique recovery of the record or message is called the *unicity distance* by Shannon.[7] It is a function of the size of the key space, the redundancy of the language, and the number of alphabets (key period) used in polyalphabetic substitutions, or the period of transposition transformations. For example, the unicity distance for M-alphabetic substitution transformation is $53M$ and for transposition of period M (i.e., character permutations take place in M-character groups) the unicity distance is 1.7 log $M!$. In general, it may be expected that in databank applications there will be large amounts of ciphertext available to the intruders. Note, however, that the ciphertext available must be longer than the key period, i.e., a key is used more than once to transform records or messages. In databank systems this can be expected to be the situation, as using nonrepeated keys to transform large amounts of data will be impractical from the point of key management for permitting information retrieval, and for providing security to the keys themselves.

Other information that helps the cryptanalyst includes:

- A number of different records known to be transformed with the same key—these can be used for simultaneous solution and checking of trial solutions.
- Fragments of plaintext corresponding to the available ciphertext, or paraphrased messages or records that are in the available ciphertext. These are very useful for generating trial solutions.
- Knowledge of the probable words in the records or knowledge of the key selection habits of the target databank—if keys are short, they may be *coherent*—are words of natural language or generated by some algorithmic process.
- As much knowledge of the statistical characteristics of the language used in the plaintext as possible.

Again, a great deal of this information, including plaintext fragments, must be expected to become available to the intruder. The ability of a privacy transformation system to withstand a cryptanalytic attack for sufficiently long (i.e., for the information to lose its value, or for the data to be retransformed) can be regarded as a measure of effectiveness of the transformations. There are two kinds of measures of effectiveness: information-theoretic measures and pragmatic "work-factor" measures.

Information-theoretic measures

These measures assess the theoretical effectiveness of a secrecy system against cryptanalysis where the intruder has unlimited resources and expertise available. Shannon[7] modeled the situation as follows: each message, R, and each choice of a privacy transformation key, K, has, from the point of view of the cryptanalyst, *a priori* probability associated with it. These are $p(R)$ and $p(K)$, respectively, and they represent the crytanalyst's knowledge of the situation before the message is transmitted.

After he intercepts and analyzes an intercepted ciphertext, E, he can calculate *a posteriori* probabilities of the various messages and keys, $p_E(R)$ and $p_E(K)$, respectively, that could have produced the intercepted ciphertext. *Perfect secrecy* is obtained if the $p_E(R)=p(R)$ and $p_E(K)$, i.e., the cryptanalyst has obtained no information at all from the intercepted ciphertext. Shannon shows that in order to have perfect secrecy, the number of keys must be at least as great as the number of possible messages.

As a measure of the theoretical amount of secrecy, Shannon defined *equivocation*—a statistical measure of how near to solution is an average cryptogram E of N characters. There are two equivocations, that of the key, $H_E(K,N)$, and the message equivocation, $H_E(R,N)$, where

$$H_E(K,N)=\sum_{E,K} p(E,K)\log p_E(K)$$

$$H_C(R,N)=\sum_{E,R} p(E,R)\log p_E(R)$$

where $p(E,K)$ and $p(E,R)$ are the *a priori* probabilities of cryptogram E and key K, and cryptogram E and message R, respectively. The summation is over all possible cryptograms of N letters and all keys or messages.

The equivocation functions for the key of the privacy transformation, $H_E(K,N)$, has the following properties:

- Key equivocation is an non-increasing function of N.
- For perfect systems, key equivocation remains constant at its initial value (when $N=0$).
- For non-perfect systems, the decrease in key equivocation is no more than the amount of redundancy in the N letters of the language L used in the plaintext.
- For most of the simple types of privacy transformations, equivocation becomes zero after the number of intercepted characters exceeds the unicity distance. After that point, a unique solution is theoretically possible.
- For certain privacy transformation systems, called *ideal* secrecy system, equivocation remains non-zero no matter how much ciphertext is intercepted.

These properties point out the importance of the *redundancy* in the language used in the plaintext records or language. If there is no redundancy at all, i.e., if all words are of equal length, say N, and if any combination of N characters of the alphabet used is a meaningful word of the language, the secrecy of the system will be

perfect. Such properties do not exist in natural languages, but can be designed into artificial languages. However, they tend to be in conflict with present trends of making artificial languages as close to natural languages as possible.

All presently existing large databank systems have redundancy in the stored data or programs. Large amounts of ciphertext, fragments of plaintext, etc. are likely to be readily available. Although exact evaluation of initial equivocation for such databank is a complex problem and has not been attempted, it is clear that simple privacy transformation systems applied here are theoretically solvable. Nevertheless, p privacy transformation systems for databank applications can be devised to have sufficiently high levels of *practical* security, i.e., sufficiently high *work factors* for the intruders, to discourage attempts to break these systems through cryptanalysis.

Work factor measures

On the practical side, an assessment of the effectiveness of privacy transformations can be attempted in terms of the effort and resources required to break the system through cryptanalysis. Such a measurement has been called the intruder's "work factor." The units of measurement can be the expected number of logical/mathematical operations. These can be converted into units of time and, subsequently, into dollars by specifying a computing capability which the intruder is expected to have available.

Several authors have examined computer-aided cryptanalysis and the effort involved.[20,23,24] Tuckerman,[20,25] in particular, has probed the computational effort involved in breaking of polyalphabetic single-loop and 2-loop substitutions under several assumptions of availability of plaintext fragments:

- For the simplest monoalphabetic substitution, the Caesar cipher, a single subtraction is sufficient if a fragment of the corresponding plaintext is available. If not, the "running down the alphabet" method can be used to generate N_A trial solutions (corresponding to the N_A characters in the alphabet) and examined for plausible plaintext. Alternately, character frequency distributions can be computed for the ciphertext and matched with the known frequencies of the language to produce solution candidates for examination. The time required on a moderately fast computer would be a few minutes at the most.
- A single-loop polyalphabetic (Vigenere) substitution of period M, can be reduced to M Caesar ciphers by a statistical analysis of the ciphertext. At least $20M$ characters of ciphertext are required. Considerable computation may be required to estimate the correct period—candidate periods are proposed, character frequency distributions computed, and correlation tests made. On a computer, however, the work is again measured in minutes or a few tens of minutes.

The 2-loop polyalphabetic substitution transformations can likewise be solved by conversion into single-loop cases and, subsequently, into Caesar ciphers. The computation required is more extensive but, by no means prohibitive. A larger hurdle to the would-be intruder is the development of programs that are needed.

Transposition transformations are solved by similar methods—by generating trial solutions, performing statistican analyses on n-grams, and using "heuristic" techniques to reduce the search space. Computational tasks, again, are not prohibitive. However, it is possible to construct complex composite transformations which require hours of computing time for their solution.

In general, the availability of digital computers and sophisticated computational algorithms has greatly reduced the protection provided in the paper-and-pencil days by the polyalphabetic and substitution transformations. Whether or not this protection is adequate in a given databank system depends on the value of protected information both to the intruder and to the owners.

Costs

The use of privacy transformations involves the initial costs of the necessary hardware or software, and the recurring costs of additional processing required and maintenance of the integrity of the privacy transformation system used.

Hardware costs are involved, in particular, in application of privacy transformations to terminal-computer communication links. Here the enciphering/deciphering device at the terminals is likely to be a hardware device. However, the logic circuitry involved is not necessarily excessive or costly since the integrated circuit prices are steadily falling. For example, the hardware involved in one, rather sophisticated ciphering/deciphering unit[26] for transforming 16-byte blocks consists of 162 TTL logic modules which could be placed on four LSI chips at a density of 280 circuits per chip. Transformation of one block requires 165 microseconds.

Software requirements, likewise, are not necessarily expensive. Programming of the mentioned transformation required some 1300 bytes of storage of 9 ms. on the IBM 360/67 computer.[26]

Other experimental data on the cost of applying privacy transformations yields similar results. The application of privacy transformations to 10-bit characters in a CDC-6600 computer[27] has shown the following percentages of processing time required for the transformations:

- Vernam type polyalphabetic substitution transformation (with one-time-only key): .66 percent to encode, .66 percent to decode.
- Polyalphabetic substitution with a short, periodic key (using table look-up technique): .25 percent to encode, 3.32 percent to decode.
- Polyalphabetic substitution with short, periodic key, using modular arithmetic for transformation: 1.94 percent to encode, 4.38 percent to decode.

As usual, there are the memory space vs. execution time overhead tradeoffs that can be applied.

The above cost figures are quite sensitive to the type of application, the computer system, and specific implementation of the transformations, and they represent only isolated data points. Estimates of decreased functional capability of a databank system due to the use of privacy transformations, as well as costs of maintaining the secrecy system integrity through providing key security, key changes, and the like, are even less available.

IMPLEMENTATION IN DATABANK SYSTEMS

Privacy transformations can be used in databank systems for protecting communications between the computer and remotely located terminals, the data stored in the files, or both. The suitability criteria for implementing privacy transformations in databanks—processing capability, error environment, security environment, and system personnel expertise—have already been discussed. These, and the specific application characteristics of the databank, provide the general criteria for selecting the type of privacy transformations to be used.

A privacy transformation system can be implemented by using hardware devices, software, or both.[26,28] Software implementation is more attractive for performing the transformations in the computer processor unit, while hardware devices appear more suitable for implementation in the remote terminals. However, the decreasing cost of hardware is making feasible the use of special privacy transformation modules also in the central processors.[26]

Application communication links and data files

The major differences in the application of privacy transformations in communication links (usually hardware switched or dedicated telephone circuits) and in data files include the following:

- In communication systems the encoding and decoding operations are done at two different locations and two copies of the key are required, while in file system application these operations are performed at the same location and only one copy of the key is needed.
- A specific communication usually involves one user, while a file may be shared between many users with different access and processing authorizations.
- In communication links, the message remains transformed for a very short time interval since encoding and decoding operations are performed almost simultaneously, in data file application this time interval may be days or months.
- The transformed records in files may be subject to selective changes at unpredictable time intervals and at unpredictable frequencies, while the message in communication link is not changed in transit.
- A change of the privacy transformation keys in a communication application is a simple replacement of the old key with the new one. In data files, this entails reprocessing the entire file of records, or maintaining an archival file of previously used keys and associated indices.
- A common-carrier communication link normally uses certain signal patterns for internal switching control. These should not appear in the ciphertext form of messages. There is no such problem in files as the control data parts.
- Communication links have higher error rates while errors in the file system are more amenable to detection and control.
- There is much less processing capability available at terminals than in the central processor.

Several other differences emerge when the implementation of one or the other of the three main classes of privacy transformations—substitutions, transpositions, composite transformations—are considered below.

Key management

The differences in the nature of communication systems and data files impact the choice of the type of privacy transformations and, in particular, the requirements for key generation, storage, logistics, and safeguarding. For example, in communication systems totally random keys can be used only once and then discarded, but in the file system they must be stored or means provided for their generation later, thus reducing the level of security such systems usually offer.

The need to store transformation keys in the data file application sets up different requirements for key possession and control than in communication links. In the latter case, individual users could be in possession of their own keys and copies stored in the processor. For file use, however, this is not desirable. The entire file, or the various classes of records in the file, should be transformed with the same key, but the keys need not be revealed to the users. Rather, the access to the file would depend on a different set of identification-authentication procedures which establish the authorization of a user to access the file and give him access to routines that retrieve or store the involved records. If the privacy transformations used have a high work factor and the key security is also high, reprocessing of the file for changing of the key need not be very frequent.

Key generation

A long key, i.e., the specification of different alphabets and their sequence, is necessary in substitution transformations where the transformation is performed by using modular arithmetic—modulo A_N addition of the key characters to the plaintext characters. As discussed previously, approximately $20M$ characters of ciphertext is needed for computer-aided solution of these transformations. If the key is sufficiently long such that this amount of ciphertext is not produced, a high degree of security is

achieved (although fragments of plaintext and the language characteristics may still make breaking of the key practical). Since storage in the computer memory of very long keys is not economical, various algorithms are used to *generate* the key as required. Computation of random numbers and feedback shift-register sequence generators are among the standard key generator techniques.[29,30] A drawback of algorithmic key generation approach is that now the *real* key is not the pseudo-random sequence added to the plaintext, but the much shorter set of parameters (an initial state and a few constants) that are used to specify a particular version of the key generation algorithms. For example, only $2n$ properly selected bits are needed in a linear feedback shift-register to produce a nonrepeating sequence of 2^n bits. Also, it is possible to recover the parameters and, hence, the key by analyzing fragments of the key stream.

Another problem with key stream generators in the communication links is the need for *synchronization* of the encoding key stream at the transmitting end with the decoding key stream at the receiving end. Such synchronization may be hard to achieve and maintain in noisy communication links. Self-synchronization is definitely a property which key stream generators should possess.[31]

Transposition and composite-privacy transformations are usually called *block transformations* as they are applied to a block of plaintext simultaneously. Very complex transformations with high work factors can be obtained.[26,32] The required keys can be stored in blocks of the main memory, special read-only memories, or generated algorithmically. Assemblying of the key at the transformation application time for several independent "subkeys" can provide additional protection against key compromises. A sophisticated block transformation can be expected to involve several sequentially applied transformations on the entire block or various subblocks. Since the computation time can be substantial for the software implementation in the processor, special purpose hardware may turn out to be more economical. In terminals, the hardware implementation is the only alternative.

CONCLUDING REMARKS

The need for data security in computerized databank systems is increasing. Privacy transformations can provide protection against a variety of threats—wiretapping to obtain transmitted information or system access control information, active entry into the system through illicit terminals, disruption through insertion of illegitimate information in the communication channel, snooping in the files, theft of removable storage devices, and the like. Their use in the databank systems, both in communication links connecting remote terminals to the processor and in data files, is now economically feasible.

On the other hand, digital computers greatly simplify the cryptanalytic tasks of the would-be intruders who must be expected to have available the necessary resources and expertise. It is important, therefore, for those charged with the design of data security mechanisms in databank systems to understand the capabilities and shortcomings of privacy transformations, and to be aware of the criteria which must be applied in their selection. This paper has strived to contribute to such understanding and awareness.

However, privacy transformations are only one facet of the general problem of access control and data security. The design of a data security system providing protection commensurate with the value of protected information requires consideration of all types of available data security mechanisms, their relative advantages and disadvantages, cost-effectiveness, and the structure and operation of the databank system. The measures of the amount of security provided by different mechanisms, measures of the value of information, and the tools for tradeoff analysis, are now beginning to crystalize into a discipline of "data security engineering." It is likely that in a few years the design of data security systems will be much less an art than it is today.

REFERENCES

1. Turn, R., Shaprio, N. Z., *"Privacy and Security in Databank Systems—Measures of Effectiveness, Costs, and Protector-Intruder Interactions,"* AFIPS Conference Proceedings, FJCC, Vol. 41, pp. 435-444, 1972.
2. Westin, A. F., Baker, M. A., *Databanks in a Free Society—Computers, Record-Keeping and Privacy,* Quadrangle Books, New York.
3. Younger, K., *Report of the Committee on Privacy,* Her Majesty's Stationery Office, London, July 1972.
4. Carroll, J. M. "Snapshot 1971—How Canada Organizes Information About People," *AFIPS Conference Proceedings,* FJCC, Vol. 41, pp. 445-452, 1972.
5. Petersen, H. E., Turn, R., "System Implications of Information Privacy," *AFIPS Conference Proceedings,* SJCC, Vol. 30, pp. 291-300, 1967.
6. Kahn, D., *The Codebreakers,* Macmillan, New York, 1967.
7. Shannon, C., "Communication Theory of Secrecy Systems," *Bell System Technical Journal,* Vol. 28, pp. 654-715, 1949.
8. "ACE Study of Campus Unrest—Questions for Behavioral Scientists, *Science,* Vol. 165, July 1969.
9. Nejelshi, P., Lerman, L. M., "A Researcher-Subject Testimonial Privilege—What to do Before the Subpoena Arrives," *Wisconsin Law Review,* No. 4, Fall, pp. 1085-1148, 1971.
10. Hansen, M. H., "Insuring Confidentiality of Individual Records in Data Retrieval and Storage and Retrieval for Statistical Purposes," *AFIPS Conference Proceedings,* Vol. 39, FJCC, pp. 579-585, 1971.
11. Fellegi, I., "On the Question of Statistical Confidentiality," *Journal of the American Statistical Association,* pp. 7-18, March 1972.
12. Boruch, R. F., "Strategies for Eliciting and Merging Confidential Social Research Data," *Policy Sciences,* Vol. 3, pp. 375-397, 1972.
13. Gaines, H. F., *Cryptanalysis,* Dover Publications, Inc., New York, 1956.
14. Sinkov, A., *Elementary Cryptanalysis—A Mathematical Approach,* Random House, New York, 1968.
15. Friedman, W. F., Mendelsohn, C. J., "Notes on Code Words," *American Mathematical Monthly,* pp. 394-409, August 1932.
16. Hill, L. S., "Cryptography in an Algebraic Alphabet," *American Mathematical Monthly,* pp. 306-312. June-July, 1929.

17. Hill, L. S., "Concerning Certain Linear Transform Apparatus of Cryptography," *American Mathematical Monthly,* pp. 135-154, March, 1931.
18. Levine, J., "Variable Matrix Substitution in Algebraic Cryptography," *American Mathematical Monthly,* pp. 170-178, March, 1958.
19. Levine, J. L., *Some Elementary Cryptanalysis of Algebraic Cryptography.*
20. Tuckerman, B., *A Study of the Vigenere-Vernam Single and Multiple Loop Enciphering Systems,* IBM Corporation Report RC 2879, May 14, 1970.
21. Miller, G. A., Friedman, E. A., "The Reconstruction of Mutilated English Texts," *Information and Control,* pp. 38-55, 1957.
22. Shannon, C. E., "Predilection and Entropy of Printed English," *Bell System Technical Journal,* pp. 50-64, 1951.
23. Fiellman, R. W., *Computer Solution of Cryptograms and Ciphers,* Case Institute of Technology Systems Research Center Report SRC-82-A-65-32, 1965.
24. Edwards, D. J., *OCAS—On-Line Cryptanalytic Aid System,* Massachusetts Institute of Technology Project MAC Report MAC-TR-27, May 1966.
25. Gridansky, M. G., "Cryptology, the Computer and Data Privacy," *Computers and Automation,* pp. 12-19, April 1972.
26. Feisel, H., Notz, W. A., Smith, J. L., *Cryptographic Techniques for Machine to Machine Data Communications,* IBM Corporation Report RC 3663, December 27, 1971.
27. Garrison, W. A., Ramamoorthy, C. V., *Privacy and Security in Databanks,* University of Texas Electronics Research Center, TM 24, November 2, 1970.
28. Kugel, H. C., "Three Cipher-Decipher Programs Make Good Os/360 Demo's" *Canadian Datasystems,* pp. 38-40, April 1972.
29. Carroll, J. M., McLelland, P. M. "Fast 'Infinite-Key' Privacy Transformation for Resource-Sharing Systems," *AFIPS Conference Proceedings,* Vol. 37, FJCC, pp. 223-230, 1970.
30. Reed, I. S., Turn, R., "A Generalization of Shift-Register Sequence Generators," *Journal of the Association of Computing Machinery,* Vol. 16, pp. 461-473, July 1969.
31. Savage, J. E., "Some Simple Self-Synchronizing Digital Data Scramblers," *Bell System Technical Journal,* pp. 448-487, February 1967.
32. Skatrud, R. O., "A Consideration of the Application of Cryptographic Techniques to Data Processing," *AFIPS Conference Proceedings,* Vol. 35, FJCC, pp. 111-117, 1969.

Design considerations for cryptography

by C. H. MEYER

International Business Machines Corporation
Kingston, New York

INTRODUCTION

Enciphering data can protect the transmission of information against unauthorized observers. One way of doing this when data is transmitted in binary form is to add random numbers to the data. If the string of random numbers is never repeated, the scheme is unbreakable.[1, p.398]

With the above rationale, using a long set of *pseudorandom* numbers (perhaps in the order of millions of bits) would appear to be a good enciphering technique. A linear shift register utilizing feedback is a convenient way to generate the necessary numbers (see Figure 1) and is an approach often taken. Addition of data and pseudorandom numbers is done via an *exclusive OR* (+) operation. The maximum period is equal to 2^{N-1} bits before repetition, where N is the number of stages of the shift register.[2] Hence, by making N large enough, extremely long periods may be realized.*

However, there is a fundamental weakness in this system. For an N- stage shift register, the signal sequence at any point can be determined using a given set of N initial conditions and N-1 switch positions (the N-th switch is always closed by definition). Hence, to break the system, all that is required is to set up $(2N\text{-}1)$ independent equations involving the initial conditions and the feedback switch position.

FORMULATION OF THE PROBLEM

To crack the code, assume that at least $(2N\text{-}1)$ bits of clear text and corresponding enciphered text are known. Such an assumption is not unreasonable. In many situations a small fragment of plain text and the corresponding enciphered text will be known to an unauthorized person. In the most general case, this fragment of text will be in some arbitrary location in the enciphered message. Also, very often, a highly formatted text is transmitted. Usually, the formatted data and the text data will both be enciphered. For example, the heading of a column of names being transmitted via the proposed cryptographic scheme could contain as part of the source message the transmitted characters, "space NAME space". If an eight bit code is assumed, this would provide an unauthorized person 48 bits of plain and corresponding enciphered text.

In other applications, an accomplice can be used to send a known message. This can be accomplished, for example, on terminals that are available to multi-users. The assumption allows the determination of $(2N\text{-}1)$ pseudorandom numbers which were used to encipher the message, i.e., $R = I + O$

where

R is the set of random numbers, I is the input message and O the output message.

To decipher the output message, it is only necessary to add to it the same random numbers used to encipher the input, i.e.,

$$I = O + R$$

(This follows from the property of the *exclusive OR* operation which states that if $A+B=C$, then $B+C=A$ and $C+A=B$).

ANALYZING THE SYSTEM

The following analysis and subsequent formulation of an attack is given to gain an understanding of why linear feedback techniques may result in poor cryptographic schemes.

To analyze the system, expressions must be evaluated for any output of the *exclusive OR* (see Figures 1 and 2), $R_{t,2}$ through $R_{t,N+1}$, in addition to the output from the first stage, $R_{t,1}$.

Let

$$a_i = \begin{cases} 0 \text{ if } SW_i \text{ is open} \\ 1 \text{ if } SW_i \text{ is closed} \end{cases} \tag{1}$$

N = number of shift register stages to the right of the last closed switch, Hence,

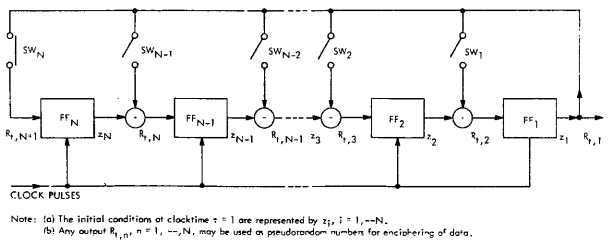
$$a_N = 1.$$

z_i = initial state ("0" or "1") of the i-th shift register stage.

$R_{t,n}$ = output at the *exclusive OR* junction of the n-th shift register stage at clock time t, $n \neq 1$

$R_{t,1}$ = output of the stage FF_1, at clock time t.

* With a 21-stage shift register, a selection of more than two million different sequences is possible, 84,672 of which are of the maximum length of 2,097,151 bits.[2]



Note: (a) The initial conditions at clocktime $\tau = 1$ are represented by z_i, $i = 1, \cdots N$.
(b) Any output $R_{t,n}$, $n = 1, \cdots N$, may be used as pseudorandom numbers for enciphering of data.

Figure 1—Linear shift register with feedback

With the aid of Figure 2 the following relations may be obtained

$$R_{t,n} = a_{n-1}R_{t,1} + R_{t-1,n+1} \tag{2}$$

$$= a_{n-1}R_{t-1,2} + R_{t-1,n+1};\ t = 2, 3, \ldots$$

$$R_{t,n} = z_n + a_{n-1}z_1;\ t = 1 \tag{3}$$

$$n = 1, \ldots, N+1$$

where

$$a_0 = 0,\ a_n = 0 \quad \text{and} \quad z_n = 0 \quad \text{for} \quad n > N$$

$$R_{t,n} = 0 \quad \text{for} \quad n > N+1$$

using (2) and (3),

$$R^{(n)} = r^{(n)}Z \tag{4}$$

where

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ \cdot \\ z_n \end{bmatrix} \tag{5}$$

$$R^{(n)} = \begin{bmatrix} R_{1,n} + z_n \\ R_{2,n} + z_{n+1} \\ \cdot \\ R_{t,n} + z_{n+t-1} \end{bmatrix} \tag{6}$$

$$r^{(n)} = \begin{bmatrix} r_{1,n} & 0 & 0 & 0 & 0 \\ r_{2,n} & r_{1,n} & 0 & 0 & 0 \\ r_{3,n} & r_{2,n} & r_{1,n} & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ r_{N-1,n} & r_{N-2,n} & r_{N-3,n} & r_{1,n} & 0 \\ r_{N,n} & r_{N-1,n} & r_{N-2,n} & r_{2,n} & r_{1,n} \\ r_{N+1,n} & r_{N,n} & r_{N-1,n} & r_{3,n} & r_{2,n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ r_{t,n} & r_{t-1,n} & r_{t-2,n} & r_{t-N+2,n} & r_{t-N+1,n} \end{bmatrix} \tag{7}$$

and

$$r_{t,n} = a_{n-1};\ t = 1$$

$$r_{t,n} = a_{n-1}r_{t-1,2} + r_{t-1,n+1};\ t = 2, 3, \ldots$$

$$n = 1, 2, \ldots, N+1 \tag{8}$$

To generate the expression for $r_{t,n}$ in general, relation (8) may be used. Table I shows this approach in more detail. With the aid of Table I and eq. (8) the expressions $r_{1,n}$ through $r_{6,n}$ used later are generated as follows

$$r_{1,n} = a_{n-1}$$

$$r_{2,n} = a_{n-1}a_1 + r_{1,n+1}$$

$$r_{3,n} = a_{n-1}(a_1 + a_2) + r_{2,n+1}$$

$$r_{4,n} = a_{n-1}(a_1 + a_3) + r_{3,n+1}$$

$$r_{5,n} = a_{n-1}(a_1 + a_2a_1 + a_2 + a_4) + r_{4,n+1}$$

$$r_{6,n} = a_{n-1}(a_1 + a_2a_1 + a_3a_1 + a_5) + r_{5,n+1} \tag{9}$$

BREAKING THE SYSTEM

Example of three-stage shift register

Assume that the output is taken from the first stage, i.e., $n = 1$. Using eq. (9) in conjunction with (2) through (7) and omitting the second index n, in $R_{t,n}$, results in

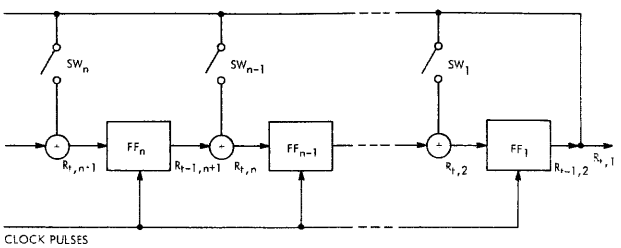
$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ a_1 & 1 & 0 \\ a_1 + a_2 & a_1 & 1 \\ a_1 + 1 & a_1 + a_2 & a_1 \\ a_1 + a_2a_1 + a_2 & a_1 + 1 & a_1 + a_2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

which permits solving for the a's and z's as follows:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ a_1 & 1 & 0 \\ a_2 & a_1 & 1 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix}$$

$$a_1 = \frac{R_2(R_5 + 1) + R_3(R_4 + R_1)}{R_2R_4 + R_3} \quad \text{if } R_2R_4 + R_3 \neq 0$$

For the case $R_2R_4 + R_3 = 0$, a solution for a_1 exists only if $R_2 = R_3 = 0$ and $R_1 \neq 0$, i.e., $a_1 = R_5/R_1$.



Note: For $\tau = 1$ the relationship $R_{t-1,n-1} = z_n$ (= initial conditions) holds.

Figure 2—General section of linear shift register with feedback

$$a_2=\frac{R_4+R_1}{R_2}+\frac{[R_2(R_5+1)+(R_4+R_1)]R_3}{(R_2R_4+R_3)R_2} \text{ if } R_2R_4+R_3\neq 0, \quad R_2\neq 0.$$

$$a_2=\frac{R_5}{R_3}+\frac{(R_4+R_1)(R_1+R_3)}{R_3} \text{ if } R_2R_4+R_3\neq 0,\ R_2=0,\ R_3\neq 0.$$

For the case $R_2R_4+R_3=0$, a solution for a_2 exists only if $R_3=R_4=0$ and $R_2\neq 0$, i.e., $a_2=R_1/R_2$.

From the conditions imposed on the solution, some randomly chosen sets of $(2N-1)$ bits will not provide the information needed to solve the $(2N-1)$ unknowns. To estimate what this means, from a practical engineering viewpoint, assume that the probability is 0.5 that a particular switch position can be determined. The selection of 0.5 is based on the fact that the output set is random and that it is always possible to arrive at a condition requiring the determination of one out of two possible values (0 or 1). Using $[(2N-1)+10]$ bits of clear and enciphered text instead of the theoretically required $(2N-1)$ bits, there would be 10 sets of random numbers available instead of one. Hence, the probability of being able to solve for a feedback switch position increases from 0.5 to $(1-0.5^{10})=1023/1024$. Thus for all practical purposes, adding 10 more bits than required assures a successful attack.

General case

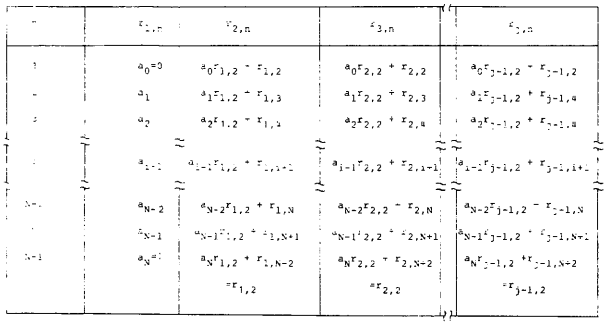
Using eqs. (4) through (7) the output at stage n can, in general, be written as follows (the index n in $R_{t,n}$ is omitted again)

$$R_t=a_{N-1}f_{t,N-1}(a_{N-2},\ldots,a_1,z_n,\ldots,z_1)+a_{N-2}f_{t,N-2}(a_{N-3},\ldots,a_1,z_n,\ldots,z_2)+\cdots+a_1f_{t,1}(z_n,\ldots,z_1) \quad (10)$$

The knowledge of $(2N-1)$ bits of clear and corresponding enciphered text allows the determination of R_t, $t=1$ to $2N-1$, providing the necessary information to solve the $(2N-1)$ unknowns (a_1 to a_{N-1} and z_1 to z_n). Factoring the a's as shown in eq. (10) is always possible because $a_i^k=a_1$, $k=1,2\ldots.$

One of the equations out of the total $(2N-1)$ equations may then be used to express a_{N-1} in terms of the other unknowns, thus eliminating one unknown.

TABLE I—Normalized Expression $r_{t,n}$ for Shift Register Output at Stage n and Clock Time t



If R_i is used to eliminate a_{N-1}, proceed as follows:

$$R_jf_{i,N-1}(a_{N-2},\ldots,a_1,z_n,\ldots,z_1)=a_{N-1}f_{j,N-1}(a_{N-2},\ldots,a_1,z_n,\ldots,z_1)f_{i,N-1}(a_{N-2},\ldots,a_1,z_n,\ldots,z_1)+a_{N-2}f_{j,N-2}(a_{N-3},\ldots,a_1,z_n,\ldots,z_1)f_{i,N-1}(a_{N-2},\ldots,a_1,z_n,\ldots,z_1)+\cdots+a_1f_{j,1}(z_n,\ldots,z_1)f_{i,N-1}(a_{N-2},\ldots,a_1,z_n,\ldots,z_1). \quad (11)$$

But,

$$a_{N-1}f_{i,N-1}(a_{N-2},\ldots,a_1,z_n,\ldots,z_1)=R_i+a_{N-2}f_{i,N-2}(a_{N-3},\ldots,a_1,z_n,\ldots,z_1)+\cdots+a_1f_{i,1}(z_n,\ldots,z_1) \quad (12)$$

which is not a function of a_{N-1}.

It is then possible to eliminate $a_{N-2},\ldots,a_1,z_n,\ldots,z_1$, thus solving for the unknown feedback switch position and initial conditions.

The ideas applied to the one shift register also apply when a variety of interacting linear shift registers are used.

A scheme could be devised in which different switches at one shift register can be randomly activated by the output from other shift registers. The a's of the previous analysis are then not constant any more, but depend on the feedback switch positions and initial condition of some other register or registers. There will, however, be at least one register whose switches are initially set and kept invariant over a period of time.*

Hence the "key" of the total system is represented by all the initial conditions and the feedback switches at one or more registers which are not controlled by other registers.

The output of such a general system can then again be written as before but the general term is now a function of all conceivable initial conditions and the factors representing the "fixed" feedback switch positions. Due to the fact that all variables are binary, it is again true that no higher order terms (i.e., quadratic, etc.) of the variables occur. In that case, the method illustrated before (eqs. (10) to (12)) can be used to solve for all unknowns and, hence, to obtain the "key".

A SUGGESTED SECURITY SCHEME

Having shown the weakness of the scheme illlustrated in Figure 1, the question is asked, "What approaches *should* be taken in order to make information secure?" A rule of thumb for the development of good crypto schemes is to combine several different mathematical operations, one of which is nonlinear, to encipher data. [In the linear shift register approach only one (linear) mathematical operation is used, i.e., modulo 2 addition.]

* If there is no point in the system where feedback switches are set over the time period where the "key" is used, i.e. if even these switches are varied continuously with the aid of a set of "true" random numbers, the system does not need any shift registers at all but is a "one-time system." Therefore, these random numbers may be used directly to decipher the message.

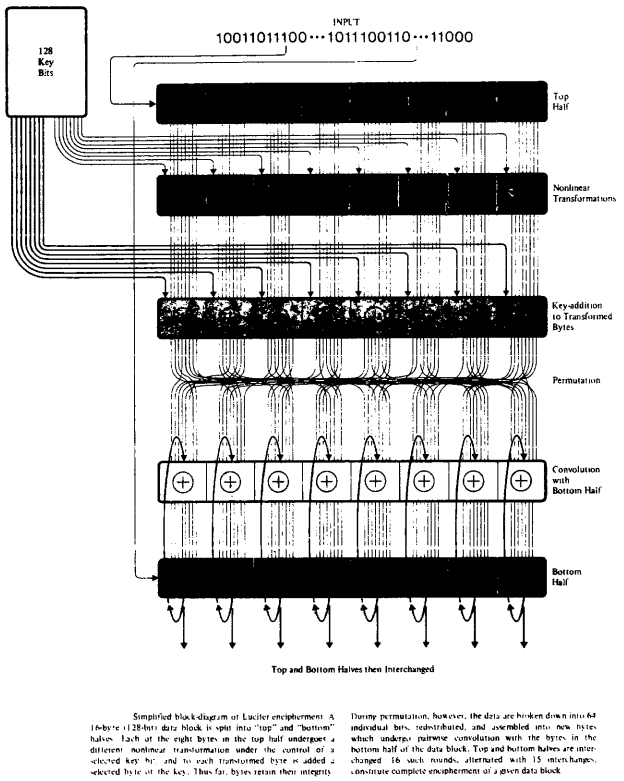


Figure 3—Cryptographic scheme developed by IBM

A block cipher product transform scheme which operates on a group of data (instead of one character at a time) under the control of a key (known only to the legitimate user) was developed by Feistel, Notz and Smith at the IBM Research Center, Yorktown Heights, New York.[3] The basic idea is shown in Figure 3.

An advantage of the block cipher is that *any* change of a single bit in the input affects, in general, *all* bits of the output. Therefore, as a by-product, this scheme may also be used for error detection purposes.

A single bit in the deciphered text is a complex function of all the bits in a transmitted message block. Therefore, if one or more of the transmitted bits are affected by transmission noise, it is almost certain that the entire deciphered text would be garbled.

CONCLUSIONS

A method was developed for determining the *output* at any point in a linear shift register with feedback in terms of the feedback switch position (a_i) and initial condition (z_i). It turns out that multiple terms of the a's, i.e., $a_i a_j \ldots a_n$, appear in the equations. Hence, a non-linear set of equations has to be solved if the feedback switch position shall be expressed in terms of the output bit stream.

The analysis is straightforward due to the fact that the a's and z's are either zero's or one's and, therefore, $a_i^n = a_i$ and $z_i^n = z_i$ where $n = 1, 2, \ldots$. This fact eliminates the occurrence of higher powers for the a's and z's and thus the mathematical approach is purely algebraic in nature.

As a general rule a set of $(2N-1)$ bits have to be known in order to solve for the $(N-1)$ unknown switch position and the N unknown initial conditions of an N-stage shift register. However for some sets of possible $(2N-1)$ output bits, no solution will exist. The mathematical reason is that some output combinations will result in dividing by zero in the analysis, which of course is not defined. To assure a solution with a high probability, all that is required is to provide an additional ten bits of output.

The output for the shift register can be evaluated if a set of clear text and corresponding enciphered text are known. Due to the fact that only a very limited amount of this information is needed, the cryptographic scheme using linear shift registers is not a very secure one. This is especially true since the information may be obtained from either an accomplice who can provide it in a matter of minutes (without much risk of being detected), or by guessing at the text.

Although the scheme of Figure 1 which was investigated in this paper is very basic, the same conclusion stated above can be made for much more complex schemes. For example, the feedback switches could be varied as a function of the output at different points in the shift register where these points by themselves could be chosen by a set of random numbers generated by another shift register.

The analysis of such a scheme will, admittedly, be more difficult. But the important factor to consider is that even in such a system, only basic equations like (10) through (12) have to be solved, thus enabling the determination of the feedback switch positions and initial conditions. Once these are determined, the output is determined for all times to come and the system is broken.

ACKNOWLEDGMENT

The author is indebted to Dr. W. L. Tuchman for his valuable suggestions and discussions.

REFERENCES

1. Kahn, D., *The Code Breakers*, the Mcmillan Company, New York, 1968.
2. Golomb, S., "Shift Register Sequences," *Modern Algebra Series*, Holen Day, 1967.
3. Girdansky, M. B., "Data Privacy, Cryptology and the Computer at IBM Research," *IBM Research Reports*, Vol. 7, No. 4, 1971.

More effective computer packages for applications

by WAYNE NELSON, MARY PHILLIPS, and LINDA THUMHART

General Electric Company Corporate Research and Development
Schenectady, New York

INTRODUCTION

This paper describes ways to make applications packages more useful to their intended users. The paper is based on our experience with the new General Electric STATPAC, a general purpose statistical package for analyzing data and for fitting models to data. The discussion is general and will aid those who use and develop applications packages in other areas. This paper is written from the viewpoint of the user and does not get into technical aspects of the statistical methodology and the computer programming. The methods we describe for making such packages more effective for users are not new. The purpose of this paper is to indicate how important and effective they are.

The following sections cover:

- Program philosophy of aiding the intended users as much as possible.
- Features that aid users.
- Simple input.
- Documentation that serves user needs.
- Education and publicity to get people to use a package.
- Statistical features of STATPAC.

Those wishing detailed information on STATPAC can find it in the Manual.[1] The Manual explains how to run STATPAC and interpret its output and presents its statistical basis. A short introduction to the basics of how to run STATPAC is given in "STATPAC Simplified,"[2] which most readers would prefer over the Manual. STATPAC's model fitting capabilities are described in detail in a separate paper.[3]

PROGRAM PHILOSOPHY

Developers of general packages must first decide on the technical content of their packages. The content, of course, depends on the specific application and is outside the scope of this paper. Developers must also identify the intended users and their needs and human nature. These aspects of developing a package are discussed in this and later sections. To illustrate these ideas, this section describes the intended users of STATPAC, their needs, and how STATPAC satisfies them. These things must similarly be considered for all applications packages.

STATPAC is intended for all in General Electric who analyze data. Such users range from those who are ignorant about data analysis and computers to those who are sophisticated. Thus STATPAC must be easy to use but powerful and general enough to satisfy sophisticated users. The ease of use cannot be overemphasized, particularly for statistical packages. They are at a disadvantage compared to other applications packages. Most users of packages in other applications areas understand the basis of the calculations and the output of their packages, and they are more highly motivated to learn to use their packages. However, many of those who wish to analyze data do not understand statistical calculations and output, and they are not motivated to put effort into learning to use a statistical package.

It is important to aid users by making a general package as easy-to-use as possible. This helps assure that the package will be useful to the largest number of users. This is done a number of ways. The package should aid users by relieving them of all unnecessary burdens. The commands for running the various features of the package should be simple and easy to remember, and their meanings should be self evident. The documentation for the package should be simple and tell the user what he wants to know. At the same time, the command language and documentation should serve the more sophisticated users. Each of these items is discussed in detail in the following sections.

FEATURES THAT AID USERS

STATPAC does a number of things to implement the philosophy of aiding the users as much as possible. This philosophy is necessary to encourage people to learn how to use the package with a minimum effort. This means that the package automatically does many things for the user. This results in a much more efficient package in terms of the time a user must spend. After all, the computer is better suited than users to detailed trivia. Of course, this convenience is usually bought at the price of greater development cost. The following provides examples of such aids, which may be useful in other packages.

It also provides examples of shortcomings that should be avoided.

STATPAC aids for users

STATPAC is one of the few statistical packages that can count. Most packages require that a user tell the package the number of variables and cases in a data set. STATPAC merely requires that a user give his names for the variables in the order they appear in the data set. Then STATPAC counts the number of variables and the number of data values it reads in. It divides the number of data values by the number of variables to get the number of data cases. STATPAC does this and many other minor bookkeeping jobs so they do not burden a user.

Data may be read into STATPAC with free format. This relieves a user of the necessity of struggling with fixed format. The package requires only that the data values be in the right order and be separated by commas or spaces. In reading data with free format, STATPAC automatically identifies which variables contain alphabetic values and which contain numerical values and keeps track of this throughout a run.

STATPAC automatically makes many decisions for users. However, STATPAC allows a user to specify certain details, if he does not want them automatically. Such details may be as simple as specifying scales for histograms and crossplots or as complicated as specifying the iteration logic for model fitting by maximum likelihood. This way of doing things allows novices to get what they want automatically and allows sophisticated users more control over the package. Where it was necessary to compromise between simplicity and sophisticated flexibility, we generally chose simplicity. We thought that sophisticated users would be able to figure out how to do what they want by clever use of existing STATPAC features.

STATPAC output is intended to be readable and self contained. Thus statistical output is accompanied by clear headings and descriptions. The error messages are complete to the point that they attempt to point out specifically where the problem occurs in the data or in a user command. If STATPAC encounters a bad command, it does not kill the run but tries to give good output for the following commands. Examples of such output are given in the next section.

We assume that most users of a general package would willingly sacrifice some computational efficiency to avoid wrong or inaccurate answers, which they might not detect. Thus, the computational routines in a general package must be robust; that is, they must be numerically accurate over a wide range of problems. This is more important than speed, particularly, in statistical work, which generally does not require appreciable computing. We tried to make the STATPAC routines for model fitting robust. Details of this appear in the appendices of the Manual.

STATPAC shortcomings

STATPAC fails to aid users in certain ways. These are consequences of software limitations of the GE and Honeywell computer systems. Some examples of these failures follow. They should be avoided in other packages.

STATPAC is a batch program. We would have preferred to make it an interactive Time-Sharing program. This would have encouraged people to use it, since many are attracted to sitting at a terminal and working directly with a responsive computer. We did not put STATPAC on Time-Sharing, because Time-Sharing would have put many severe limitations on STATPAC.

In General Electric, many STATPAC users are in departments that do not have a computer that can run STATPAC. To accommodate such users, STATPAC is designed so it can be run as a remote batch program. To do this, users can submit a run from a teletype terminal and get the output back from the terminal. All STATPAC output is formatted to conform with a teletype. This feature for remote runs is essential to make STATPAC available to virtually everyone in General Electric.

On GE computers, batch programs must be submitted with system control cards to specify what computer resources will be used. These cards involve a separate system language, which is quite cryptic and which typical STATPAC users do not know and understandably do not care to learn. Roughly half of the user problems in running STATPAC come from errors in these cards. This is particularly frustrating to users who are used to running programs on Time-Sharing systems, which automatically do the job of the control cards. To minimize the difficulty, we have developed an interactive Time-Sharing program. It asks a user questions to extract the necessary information and then sets up the command cards and submits the run. This program has been a big help to users.

STATPAC is a large package. However, it consists of modules and does not require more than 27 K words of core, since modules not in use stay on temporary disc. A STATPAC run involves considerable swapping between the disc and core. Because STATPAC spends much time moving files between disc and core, it spends about 50 minutes waiting for a minute of processor time, whereas typical programs spend 10 minutes. On longer runs, users are dissatisfied with this. However, reducing this problem would require more core and considerable restructuring of STATPAC.

STATPAC accommodates a data set with up to 30 variables and 500 cases. This has been adequate for most sets of data, particularly engineering data. The number of cases may be increased readily, and this requires proportionally more core. The number of variables cannot readily be increased. Although STATPAC can handle $30 \times 500 = 15{,}000$ data values, it cannot allocate them to get any number of variables and the corresponding number of cases. Such dynamic allocation would be preferable but is not feasible. The size of the data set has been a

problem in a few applications. For example, a computer survey analyzed with STATPAC contained over 100 questions. The data had to be analyzed in groups of fewer than 30 questions (variables).

SIMPLE INPUT

Simple input to a package is essential to aid users. The following features of STATPAC input may be useful in other packages.

STATPAC analyses are requested through simple commands. Users readily understand the meanings of most STATPAC commands—even users with a limited background in statistics and computing. Many new users are able to understand and use most of the commands after reading the summary of commands in Appendix A of the Manual. The simplicity of the command language has encouraged people to become users, since there is so little effort in learning to run STATPAC through its commands.

STATPAC commands

The following describes certain aspects of the STATPAC language that should be useful in other applications packages. This section concludes with an example to illustrate the use of the commands.

The commands employ words that are readily recognized by users, and the word order is generally logical. Thus, the appropriate form of a command is relatively easy to remember. Also, there are only about 15 basic commands, excluding data selection and transformation. Many users want only summary statistics and graphical displays; they make do with 9 or fewer commands. In addition, most commands have certain options, which most users do not need and can ignore. Only the one basic form of the command needs to be remembered. All of this makes it easy for a user to remember the commands.

Most commands do not involve trivial details that a user could easily forget or misuse. For example, STATPAC reads commands with a free format. This means that extra spaces or commas are ignored. Also, most commands follow consistent patterns. For example, the names of variables are always enclosed in parentheses, and constant values in a command always follow the names of the variables. STATPAC reads and executes the commands one at a time.

Example of STATPAC commands and output

The data in Figure 1 are used to illustrate the STATPAC commands. The data are shown as they would be punched on cards for STATPAC. This is one of six sets of data that were obtained from an experiment on the breakdown voltage of an insulating fluid. Voltage across a pair of electrodes in the fluid was raised linearly with time until dielectric breakdown was observed. For each breakdown, the variables consist of the RUN, the AREA of the electrodes, the RATE of rise of voltage, the NUMBER of the breakdown, and the breakdown VOLTage.

RUN	AREA	RATE	NUMBER	VOLT
3.	1.	10.	10.	41.
3.	1.	10.	11.	43.
3.	1.	10.	12.	42.
3.	1.	10.	13.	43.
3.	1.	10.	14.	44.
3.	1.	10.	15.	40.
3.	1.	10.	16.	38.
3.	1.	10.	17.	47.
3.	1.	10.	18.	45.
3.	1.	10.	19.	45.
3.	1.	10.	20.	38.
3.	1.	10.	21.	44.
3.	1.	10.	22.	49.
3.	1.	10.	23.	42.
3.	1.	10.	24.	42.
3.	1.	10.	25.	51.
3.	1.	10.	26.	39.
3.	1.	10.	27.	34.
3.	1.	10.	28.	41.
3.	1.	10.	29.	41.
3.	1.	10.	30.	35.
3.	1.	10.	31.	44.
3.	1.	10.	32.	46.
3.	1.	10.	33.	39.
3.	1.	10.	34.	41.
3.	1.	10.	35.	40.
3.	1.	10.	36.	52.
3.	1.	10.	37.	40.
3.	1.	10.	38.	35.
3.	1.	10.	39.	40.
3.	1.	10.	40.	39.
3.	1.	10.	41.	46.
3.	1.	10.	42.	47.
3.	1.	10.	43.	44.
3.	1.	10.	44.	41.
3.	1.	10.	45.	46.
3.	1.	10.	46.	46.
3.	1.	10.	47.	42.
3.	1.	10.	48.	45.
3.	1.	10.	49.	42.
3.	1.	10.	50.	44.
3.	1.	10.	51.	41.
3.	1.	10.	52.	44.
3.	1.	10.	53.	38.
3.	1.	10.	54.	36.
3.	1.	10.	55.	44.
3.	1.	10.	56.	50.
3.	1.	10.	57.	47.
3.	1.	10.	58.	49.
3.	1.	10.	59.	46.
3.	1.	10.	60.	34.
3.	1.	10.	61.	47.
3.	1.	10.	62.	49.
3.	1.	10.	63.	43.
3.	1.	10.	64.	43.
3.	1.	10.	65.	48.
3.	1.	10.	66.	34.
3.	1.	10.	67.	38.
3.	1.	10.	68.	47.
3.	1.	10.	69.	35.

Figure 1—Voltage breakdown data on cards

Commands

```
READ (RUN AREA RATE NUMBER VOLT) BYCASE FROM (31)
LIST(ALL)
HISTOGRAM(VOLT)
TABULATION(VOLT FROM 20. TO 70. STEP 1.)
CROSSPLOT (NUMBER ; VOLT)
SUMMARY(NUMBER VOLT)
PROBPLOT WEIBULL(VOLT)
DISTRIBUTION WEIBULL(VOLT)
FIT
PCTILES
PERCENT(LIMIT 50.)
STOP
```

Selected Output

```
STATPAC PROGRAM
VERSION OF SPRING 73

02/06/73  11.72

PROGRAM COMMANDS & ERROR MESSAGES

READ (RUN AREA RATE NUMBER VOLT) BYCASE FROM (31)

*     60 CASES READ IN


HISTOGRAM(VOLT)

* HISTOGRAM FOR VOLT

  EACH * REPRESENTS  1 OBSERVATIONS

              CELL
              LOWER
FREQ          ENDPT .    .    .

             30.0000
             32.0000
   6         34.0000 ******
   1         36.0000 *
   7         38.0000 *******
  10         40.0000 **********
   9         42.0000 *********
  10         44.0000 **********
  10         46.0000 **********
   4         48.0000 ****
   2         50.0000 **
   1         52.0000 *
----
  60 TOTAL           .    .    .


CROSSPLOT (NUMBER ; VOLT)

* CROSS PLOT OF NUMBER VS VOLT

     NUMBER        VOLT   CELL LOWER ENDPT
NO. CELL
IN  LOWER      34.0000   39.0000   44.0000   49.0000
ROW ENDPT          36.5000   41.5000   46.5000   51.5000
                <+....+....+....+....+....+....+....>
   2    10.000 +               1   1
   2    12.000 +                 1 1
   2    14.000 +             1       1
   2    16.000 +         1                 1
   2    18.000 +                       2
   2    20.000 +         1           1
   2    22.000 +                 1             1
   2    24.000 +                 1                 1
   2    26.000 + 1         1
   2    28.000 +               2
   2    30.000 +   1                 1
   2    32.000 +           1             1
   2    34.000 +             1 1
   2    36.000 +             1                       1
   2    38.000 +   1         1
   2    40.000 +           1             1
   2    42.000 +                     1     1
   2    44.000 +               1         1
   2    46.000 +                 1       1
   2    46.000 +                 1     1
   2    50.000 +               1     1
   2    52.000 +         1           1
   2    54.000 +     1               1
   2    56.000 +                           1     1
   2    58.000 +                         1     1
   2    60.000 + 1                         1
   2    62.000 +                   1           1
   2    64.000 +                   1         1
   2    66.000 + 1       1
   2    68.000 +   1                       1
        70.000 +
                <+....+....+....+....+....+....+....>
        NO.
        IN       3   1   4   4   5   7   5   1   1   1
        COL.
  60 TOTAL         3       3   6   4   3   5   3   1
```

```
PROBPLOT WEIBULL(VOLT)

**WEIBULL *        - VOLT   **

VOLT            CUMULATIVE PROBABILITY
         .3 .5 .7  1    2     4   7 10    20 30   50 60  80 9095  99 .95
         :   :  :  :    :      :   :  :     :  :    : :    :  :  :  :: :

     110..      .       .      .      .     .       .      .    .      .

     100.

     90.0.      .       .      .      .     .       .      .    .      .

     80.0

     70.0.      .       .      .      .     .       .      .    .      .

     60.0

                                                                   *
     50.0.      .       .      .      .     .       .      .  2***     .
                                                          222
                                                       232*
                                                   2233*
                                              *2223*
     40.0                                  **22*
                                       ****
                                      *
                      *     *  *  *  *  *

     30.0.      .       .      .      .     .       .      .    .      .

     26.0
     24.0.      .       .      .      .     .       .      .    .      .
     22.0
     20.0.      .       .      .      .     .       .      .    .      .

     17.0

     15.0.      .       .      .      .     .       .      .    .      .

     13.0
     12.0.      .       .      .      .     .       .      .    .      .
     11.0
```

DISTRIBUTION WEIBULL(VOLT)

FIT

ITER.	LOG LIKELIHOOD	C00001	C00002
0	-408.14567	1.3415062	-1.9961833

* SWITCHING TO UNTRANSFORMED STANDARDIZED OBSERVATIONS

5	-174.69513	3.7970427	2.3833659

AFTER 6 ITERATIONS 56 FUNCTION EVALUATIONS

MAXIMUM LOG LIKELIHOOD = -174.69513

* MAXIMUM LIKELIHOOD ESTIMATES FOR DIST. PARAMETERS
WITH APPROXIMATE 95% CONFIDENCE LIMITS

PARAMETERS	ESTIMATE	LOWER LIMIT	UPPER LIMIT	STANDARD ERROR
CENTER *scale*	=44.56919	43.48272	45.68280	0.5611896
SPREAD *shape*	=10.84133	8.943323	13.14215	1.064545

* COVARIANCE MATRIX

PARAMETERS	CENTER	SPREAD
CENTER	0.3149338	
SPREAD	0.1938376	1.133256

PCTILES

* MAXIMUM LIKELIHOOD ESTIMATES FOR DIST. PCTILES
WITH APPROXIMATE 95% CONFIDENCE LIMITS

PCT.	ESTIMATE	LOWER LIMIT	UPPER LIMIT	STANDARD ERROR
0.1	23.56872	20.63985	26.91320	1.595658
0.5	27.34570	24.62847	30.36272	1.460153
1	29.15793	26.57755	31.98883	1.378455
5	33.88841	31.75365	36.16669	1.124963
10	36.21487	34.33392	38.19886	0.9854848
20	38.81042	37.22260	40.46577	0.8270476
50	43.08762	41.91933	44.28847	0.6042951
80	46.56914	45.49443	47.66924	0.5547468
90	48.13327	46.97705	49.31794	0.5971073
95	49.31592	48.04976	50.61548	0.6544387
99	51.31121	49.78660	52.88250	0.7896508

Figure 2—Preliminary analyses

Figure 2 shows STATPAC commands for various analyses of the illustrative data. The figure also shows selected output. The purpose of these analyses is to examine the VOLT data to assess whether the data are valid and adequately described by a Weibull distribution, which is assumed in the theory for such breakdowns. For example, the crossplot of breakdown VOLTage versus breakdown NUMBER provides a check of whether the distribution of breakdown voltage is constant over time.

Figure 3 shows further STATPAC commands for various analyses of the six sets of such data obtained with different combinations of RATE of rise and electrode AREA. The purpose of these analyses is to fit the following nonlinear and nonnormal model to the data. For an electrode area A and rate of rise r, the cumulative distribution of breakdown voltage V is the Weibull distribution

$$F(V;A,r)=1-\exp(-(V/a)^{\beta}$$

where the Weibull shape parameter β is a constant and the scale parameter α is given by the inverse power law relationship

$$a=[r\beta\exp(-\gamma)/A]^{1/\beta}.$$

This nonlinear relationship contains the parameters β and γ which are to be estimated from the data. Figure 3 shows the commands for displaying the data, for fitting the model, and for examining the residuals. The output clearly shows that one of the observations is an outlier.

A complete explanation of this example and the output is given in the STATPAC Manual.[1]

DOCUMENTATION THAT SERVES USER NEEDS

The main way a user can learn to use most packages is to read documentation. Thus documentation is a key factor in the success of a package. To serve the users, documentation must be easy to use and organized with the user in mind. Ways of achieving this follow.

Good documentation

The trouble with many manuals for packages is that they are organized logically. Instead, they should be organized intelligently with the needs and psychology of the users in mind. Logically organized documentation presents all of the material on a topic in one place. This strains the patience of a new user who has to read about special detail and features that do not interest him. Intelligently organized documentation tells a user just what he wants to know and in the order he wants to learn it. Thus the most basic and useful features of a package should be presented first and in their simplest form, and their special options and the unusual features should be presented last. Of course, documentation must contain specialized material wanted by only a few users. But such material must not burden uninterested users. Such material is best put at the end of the documentation.

The STATPAC Manual tries to compromise between logical organization and intelligent organization. Wherever a particular feature of the package is presented, all of the details are presented in one place. However, the basic information is separated from the trivia by two means. First, the minor technical details and special options are relegated to the end of a section and are clearly labeled "Technical details and limitations." Second, the key material is highlighted so readers can readily spot it.

Documentation should be organized linearly; that is, so its readers do not have to know something that appears later in the document to understand what they read early in the document. One reader of the STATPAC Manual commented that it was the first manual he had read that did not assume he knew as much about the package as its programmer. An exaggeration perhaps, but his point about documentation is clear.

The Manual is organized with the simplest and most widely used features first. It starts with data input, then goes into simple data displays and summaries, and finally covers model fitting. Thus, a reader can start at the beginning of the Manual and read until he can handle his data analyses; he will not need to read much that he does not need. Also, page 1-6 of the Manual contains a flow chart showing the ways one can read through the STATPAC Manual to learn as quickly as possible about any features.

To say that documentation should be well written is to belabor the obvious. However, there are some common mistakes that should be avoided. Organizing the documentation intelligently and linearly was mentioned above. Also, documentation should be written in users' language. Computer jargon should be avoided at all costs, and jargon of the applications area should be minimized. This is particularly true for statistics packages, whose users are generally not acquainted with either jargon. Where necessary, technical terms should be introduced and defined clearly.

The documentation must tell the user what he needs to know to run the package. One way to assure this is have it written by someone who is not involved in the programming and computer aspects of the package. Such a person must find out how to run the package. Thus he can anticipate the needs of users, since he has been in their shoes. This was previously done quite successfully with the ADA manual,[4] and this was also done with the STATPAC Manual.

Shortcomings of the STATPAC manual

One difficulty with the STATPAC Manual is its size—234 pages. Most users need to read only selected material

Commands

```
READ (RUN AREA RATE  NUMBER VOLT) BYCASE FROM (31 32 33 34 35 36)
LIST(ALL)
USE LOG10(RATE) TO SET (LRATE)
USE LOG10(VOLT) TO SET (LVOLT)
USE LOG10(AREA) TO SET (LAREA)
USE OREQ(AREA ; 1.) TO SELECT
CROSSPLOT(LVOLT ; LRATE)
USE OREQ (AREA ; 9.) TO SELECT
CROSSPLOT (LVOLT ; LRATE)
USE OREQ (AREA ; 1. 9. ) TO SELECT
DISTRIBUTION WEIBULL(VOLT)
RELATIONSHIP SCALE(AREA RATE)
COEFFICIENTS(-55. 16. ; .001 .001)
FIT
PCTILES(ALL)
ESTIMATE EXRATE(10.)
ESTIMATE EXRATE(20.)
CROSSPLOT (RESIDUALS ; LRATE)
CROSSPLOT(RESIDUALS ; LAREA)
CROSSPLOT(RESIDUALS ; NUMBER)
SUMMARY(RESIDUALS LRATE LAREA NUMBER)
PROBPLOT EXTREME (RESIDUALS)
STOP
```

Selected Output

```
STATPAC PROGRAM
VERSION OF SPRING 73

02/08/73  24.144

PROGRAM COMMANDS & ERROR MESSAGES

READ (RUN AREA RATE  NUMBER VOLT) BYCASE FROM (31 32 33 34 35 36)

*    360 CASES READ IN


DISTRIBUTION WEIBULL(VOLT)

RELATIONSHIP SCALE(AREA RATE)

COEFFICIENTS(-55. 16. ; .001 .001)

FIT

ITER.  LOG LIKELIHOOD         C00001          C00002

  0       -1259.0066        -55.000000       16.000000

* SWITCHING TO UNTRANSFORMED STANDARDIZED OBSERVATIONS

  5       -1055.8244        -54.666520       15.678361

AFTER     8 ITERATIONS   118 FUNCTION EVALUATIONS

MAXIMUM LOG LIKELIHOOD =      -1055.8209

* MAXIMUM LIKELIHOOD ESTIMATES FOR MODEL COEFFICIENTS
  WITH APPROXIMATE 95% CONFIDENCE LIMITS

COEFFICIENTS  ESTIMATE     LOWER LIMIT    UPPER LIMIT  STANDARD ERROR

C00001 γ̂ = -54.67176      -56.99983      -52.34370       1.187787
C00002 β̂ =  15.67862       15.07156       16.28569       0.3097254

* COVARIANCE MATRIX

COEFFICIENTS  C00001        C00002

C00001          1.410838
C00002         -0.3675256   0.9592982E-01

* FISHER MATRIX

COEFFICIENTS  C00001        C00002

C00001          360.1400
C00002          1379.766      5296.573
```

```
CROSSPLOT (RESIDUALS ; LRATE)

* CROSS PLOT OF RESIDU VS LRATE

     RESIDU       LRATE  CELL LOWER ENDPT
NO. CELL
IN  LOWER         1.0000    1.5000    2.0000    2.5000    3.0000
ROW ENDPT              1.2500    1.7500    2.2500    2.7500
                  <+....+....+....+....+....+....+....+....>
  1  -11.000 + 1      <-- Outlier
     -10.500 +
     -10.000 +
      -9.500 +
      -9.000 +
      -8.500 +
  1   -8.000 +                                        1
      -7.500 +
  1   -7.000 +                    1
  3   -6.500 +                                        3
  2   -6.000 + 1                  1
  2   -5.500 +                    1                   1
  3   -5.000 + 1                  2
  4   -4.500 + 3                                      1
  6   -4.000 + 5                                      1
 11   -3.500 + 3                  6                   2
 13   -3.000 + 6                  2                   5
 24   -2.500 + 3                  C                   9
 31   -2.000 + D                  7                   8
 33   -1.500 + 8                  8                   E
 47   -1.000 + A                  J                   I
 62   -0.500 + N                  I                   L
 42    0.    + D                  A                   J
 43    0.500 + C                  M                   9
 22    1.000 + E                  4                   4
  7    1.500 + 3                  4
  2    2.000 + 1                                      1
       2.500 +
                  <+....+....+....+....+....+....+....+....>
       NO.                                           120
       IN       120               120
       COL.
 360 TOTAL
```

PROBPLOT EXTREME (RESIDUALS)

EXTREME VALUE * - RESIDU

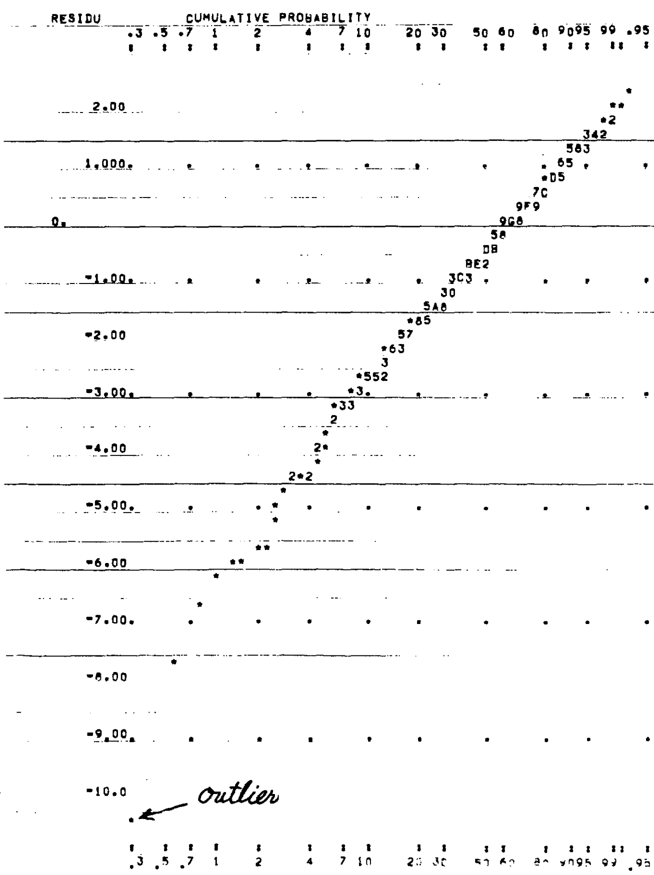


Figure 3—Fitting the inverse power law model and checking the model and data

from about 40 key pages (Sections 3 and 4). Unfortunately, such a complete reference manual is necessary. A manual this size has two drawbacks: (1) it discourages readers so they do not read it and learn how simple the command language is, and (2) it is costly. The needs of most users are better served with a short handbook on the basics. Thus we are developing such a 20 page report called "STATPAC Simplified,"[2] which people can read more readily.

One reason the STATPAC Manual is so long is that much of it explains how to do data analyses (primarily engineering applications) and how to interpret the STATPAC output. Documentation of most statistical packages assumes that a user knows what analyses to run on his data and what the output means. However, this is not true of many users, even for basic analyses. Moreover, STATPAC contains novel features that require explanation even to sophisticated analysts. Thus, the Manual contains many examples of actual data analyses and considerable explanation of them. For application packages in other areas, it may be more reasonable to assume that users are better acquainted with the meaning and use of the output, since it concerns their field of expertise. An important benefit of the Manual is that people learn to do better data analyses from it.

Section 2 of the Manual provides a general overview of data analysis and model fitting. This kind of general discussion appears early in the documentation of many packages. It is a waste of time. Most readers do not need it and do not understand it, and it tries their patience. Section 2 should have been the last appendix of the Manual. Also, Section 1 of the Manual, which describes the package, is much too long. Readers would rather get right to the business of how to run the package.

EDUCATION AND PUBLICITY

Users generally need assistance in learning what a package can do for them and how to run it. Thus publicity and education have been a major concern for STATPAC. Continuing activities which contribute to spreading the use of STATPAC are described here. Such activities would contribute to the use of other packages.

At the outset, STATPAC was supported by funds from a number of departments that recognized that STATPAC uniquely satisfied many of their needs for data analysis tools. Having a stake in STATPAC, such departments are then committed to carry through so their people can use the package. This, of course, requires training people how to run the package. Thus, STATPAC was fortunate to have a ready-made nucleus of a group of users who had a real need for the package.

The Manual has contributed to wider use of STATPAC. General Electric has the Technical Digest System which sends abstracts of all reports to individuals who have indicated interests in certain technical areas. As a result of this system, over a thousand requests for the Manual came from all over the General Electric Company.

STATPAC was announced in the General Electric Statogram Newsletter, which has a mailing list of about five thousand GE employees who asked to be put on the list. There was considerable response to such Newsletters describing the package.

Presentations on STATPAC have been given to several hundred people in General Electric. A forty-five minute presentation briefly describes the types of results that STATPAC provides and their practical value in real applications. A four hour presentation describes how to use STATPAC. Most of this time is spent on describing actual problems and how STATPAC was used to extract useful information from the data. This involves showing how to go about doing a data analysis and how to interpret STATPAC output. The smallest portion of the time is devoted to the STATPAC commands.

The Manual and presentations serve real needs. However, the most effective way for individuals to learn how to use STATPAC has been for them to work with us on a one-to-one basis on one of their applied problems. This is very quick and effortless for such users, since they are shown exactly what they need. Also, guidance from us helps them learn still more ways to look at their data and to extract more information. Those who have received such tutoring rapidly become adept at independently doing good data analysis, despite having only a minimal statistical background. This method of teaching has been remarkably successful for a moderate investment of time.

CONCLUDING REMARKS

The aim of this paper was to describe various means for making an applications package suitable for users. These means include simplicity of the package, easy-to-understand documentation that tells users what they want to know, and continuing publicity and education. Descriptions of such means were given. However, the most important thing is to identify the intended users and to satisfy their needs.

ACKNOWLEDGMENTS

The authors wish to express their sincere appreciation to Dr. Richard L. Shuey, Manager of the Information Studies Branch of General Electric Corporate Research and Development, for his support of this work and the stimulus to write this paper. The authors are grateful for beneficial comments on this paper from Dr. Paul Feder, Dr. Gerald J. Hahn, Mr. John Hamm, Mr. Joe Radosta, and Mr. John Williams, of General Electric. The authors wish to thank Dr. Carl Hammer, of UNIVAC, for his considerable help in publishing this paper.

REFERENCES

1. Nelson, W. B., Hendrickson, R., *1972 User Manual for STATPAC —A General Purpose Program for Data Analysis and for Fitting Statistical Models to Data*, General Electric Co. Corp. Research and Development, TIS Report 72-GEN-009, May 1972*
2. Nelson, W. B., *STATPAC Simplified—A Short Introduction to STATPAC—A General Statistical Package for Data Analysis*, General Electric Co. Corp. Research and Development, TIS Report 73-CRD-046, May 1973.*
3. Nelson, W. B., *Generalized Methods for Fitting Statistical Models to Data*, in preparation.
4. Watson, J. M., Moore, H. W., *ADA70 Users' Manual—Automatic Dynamic Analyzer 1970*, General Electric Research and Development Center, TIS Report 70-C-263, August 1970.*
5. Chambers, J. M., "A Computer System for Fitting Models to Data," *Applied Statistics*, Vol. 18, No. 3, pp. 249-63, 1969.
6. Schucany, W. R., Minton, P. D., Shannon, B. S., Jr., "A Survey of Statistical Packages," *Computing Surveys*, Vol. 4, No. 2, pp. 65-79, June 1972.

APPENDIX—STATISTICAL FEATURES OF STATPAC

This section briefly describes the statistical features of STATPAC. It is mainly of interest to those who are concerned with statistical packages. Others may wish to skip this section.

Statistical features

STATPAC is a general purpose statistical package for data analysis and for fitting models to data. STATPAC and its Manual[1] were developed so those with limited backgrounds in statistics and computers can analyze data easily. The generality and power of the package also satisfies highly sophisticated users. Written in batch Fortran for GE and Honeywell 600 and 6000 computers, it may be run as a batch program or by teletype or 115 terminal as a remote batch program.

All analyses are conveniently available through one easy-to-use package requiring the data to be input only once. Simple commands select subsets of the data, transform variables, and perform analyses. STATPAC provides versatile graphical displays of data, including histograms, tabulations, crossplots, and probability plots. It fits a great variety of statistical distributions and engineering relationships to data and provides standard relationships on request. Fitting is done with a general approach based on the method of maximum likelihood. This includes as special cases the fitting of linear and nonlinear regression relationships and analysis of variance by least squares. (Chambers[5] describes a similar system for model fitting.) Also, STATPAC provides displays and summary statistics of residuals from such fitted models. Both quantitative (numerical) and qualitative (alphabetic) data are automatically handled. An important feature of STATPAC is its ability to analyze censored data on product life. Such data, which consist of a mixture of times to failure on failed units and running times on unfailed units, occur frequently in General Electric.

The statistical features of STATPAC were developed with engineering applications in mind. Thus STATPAC lacks features for certain types of analyses including time series analyses, multivariate analyses (e.g., factor analysis, discriminant analysis, etc.), and nonparametric analyses. However, users may add such routines of their own to STATPAC. A comparison of STATPAC with some statistical packages appears in Appendix H of the STATPAC Manual. Other statistical packages are described in the survey by Schucany, Minton, and Shannon.[6]

Applications of STATPAC

STATPAC is being used in a wide variety of applications ranging from simple data display and summaries through sophisticated model fitting. Examples include:

- Graphical display of performance data from a computer operating system; this provided information on the factors influencing computer performance.
- Routine reduction of voluminous data on characteristics of solid state devices measured during manufacture; simple displays and summary statistics are providing easy-to-grasp information.
- Analyses of accelerated life test data on different capacitor designs; analyses of censored life data at accelerated conditions provided estimates and comparisons of the life distributions of the different designs at operating conditions.
- Analyses of data from a designed experiment on the yield of the manufacturing process of a medical device; data displays revealed factors affecting yield and showed which manufacturing tolerances had to be tightened and which could be relaxed.

Those who are using STATPAC are doing better analyses than before. This is true of both novices and sophisticated users. There are several reasons for this. First, STATPAC makes it easy to run a large number of diverse analyses on a set of data. Such analyses are primarily graphical displays of the data involving several inches of output, which can be quickly scanned. Such analyses help extract more of the information in the data. The STATPAC Manual introduces many users to standard methods for data analysis. Also, STATPAC makes possible many useful new analyses not previously available at GE, particularly, model fitting with various distributions and relationships and censored life data. Indeed, the original motivation for developing STATPAC was to provide such new model fitting capabilities.

* Reprints and General Electric reports may be requested from the Distribution Unit, Building 5, Room 237, General Electric Co. Corp. Research and Development, Schenectady, New York 12345.

EASYSTAT—An easy-to-use statistics package

by ALLEN B. TUCKER

Georgetown University
Washington, D.C.

INTRODUCTION

When surveying the available computer statistical packages (see e.g. (1)), one usually has in mind several *ideal* characteristics which should be possessed by such a package. First, the user envisions that the package will be both easy to learn and natural to use thereafter—hopefully without any help from his local computer specialist and without any knowledge of computers or of programming. Specifically, knowing exactly what particular statistical operation he wants performed by the computer and having his data in hand, the user should have a means by which he can (1) describe in a natural way that operation (e.g., a multiple linear regression) and that data (e.g., 6 variables and 23 observations per variable), and (2) present that data in a reasonable natural and unconstrained way.

Second, from the system's point of view, one envisions that the package provide a wide selection of statistical operations, accommodate a reasonably large amount of data, contain as few other "system restrictions" as possible, be reasonably transportable among a large number of computers, run in a modest amount of main storage, and run identically in either batch or interactive mode.

EASYSTAT has been designed to meet these objectives. Its second version is now in use at Georgetown. That version has all except the last characteristic. This paper describes those characteristics of that version.

LEARNING TO USE EASYSTAT

The first encounter a user has with EASYSTAT is its text. Unlike other computer manuals, this text is written in plain English without a lot of "computerese." Furthermore, it is self-teaching. That is, knowing what statistical operation he wants performed and the characteristics of the data upon which he wants that operation performed, the user is directed to appropriate sections of the text, answering questions relevant to his particular problem as he reads. When he has finished, his resulting list of answers is his "problem description" to the EASYSTAT system.

For example, suppose you require that two multiple linear regressions be performed on data with 6 variables and 23 observations. In the first regression, you want variable 6 to be the dependent variable and variables 1, 2, and 3 to be the independent variables. In the second, you want variable 4 and variables 3 and 5 to be the dependent and independent variables, respectively. Finally, suppose you want a table of residuals to be printed for the first regression.

The first question you encounter in the text asks you simply to identify the statistical operation you want performed. In this case, your answer will be the following.

MULTIPLE LINEAR REGRESSION;

You are then led to a section of the text which will ask you to describe the particular characteristics of the operation you have chosen. For this example, that section is shown in Figure 1. Note there the following question/answer conventions. Each question is written in all-caps, followed immediately by the form in which its answer should be written. An optional question (i.e., one that need not be answered) is so-indicated by its enclosure in square brackets. Each optional question has an associated default, which is the answer implied if one chooses not to explicitly answer it.

For this example, one would have the following list of answers in response to the questions in Section 3.7.

```
9 VARIABLES;
23 OBSERVATIONS;
DEPENDENT VARIABLE 6;
INDEPENDENT VARIABLES 1,2,3;
RESIDUALS;
DEPENDENT VARIABLE 4;
INDEPENDENT VARIABLES 3,5;
```

Finally you are led to a section which shows you how to prepare your problem description (i.e., your answers) and your data to be run on the computer. Keypunching each of these into cards is quite easy since they can be punched in relatively free form. For this example, Figure 2 illustrates a complete keypunched deck ready to run under EASYSTAT.

The results of a successful EASYSTAT run are quite standard, as the example's results show in Figure 3. An unsuccessful EASYSTAT run, however, will contain an error message indicating both what went wrong and a

```
3.7     MULTIPLE LINEAR REGRESSION

        Your data may consist of one or more data groups, all having the same
number, v, of variables and the same number of observations.  In a data group,
each variable is identified by an integer from 1 to v.  An "observation" is
understood here as consisting of exactly v data values, one for each variable.
A "missing  observation" is understood to be any observation which has at least
one missing data value.  Missing observations will be excluded from the calculation.
        For every data group, MULTIPLE LINEAR REGRESSION operates as follows.
You may specify from 1 to 10 "selections," each consisting of a dependent variable
and one or more independent variables for the regression.  For each selection,
this operation  gives the means and standard deviations of the selected variables,
the correlation coefficients, the regression coefficients and their standard
errors, the computed t values, and the beta coefficients.  In addition, it gives
the intercept, multiple correlation coefficient, standard error of the estimate,
analysis of variance with the appropriate F ratio for the regression, and
(optionally) a table of residuals for each such selection.

        [HOW MANY DATA GROUPS DO YOU HAVE?]
        Answer:  _____ DATA GROUPS;
        Default:  1 DATA GROUP;
        Notes:  You need answer this question only if you have more than one
             data group.  In that event, the maximum permitted is 99.

        HOW MANY VARIABLES DO YOU HAVE IN EACH DATA GROUP?
        Answer:  _____ VARIABLES;
        Note:  The maximum permitted is 30 and the minimum is 2.

        HOW MANY OBSERVATIONS DO YOU HAVE IN EACH DATA GROUP?
        Answer:  _____ OBSERVATIONS;
        Note:  The maximum permitted is 8000/v, including missing observations.

3.7.1   Your answers to the following questions select dependent and independent
variables for the regression and specify whether or not you want a residuals
table for that selection.

        WHAT IS THE DEPENDENT VARIABLE?
        Answer:  DEPENDENT VARIABLE ____;
        Note:  Your answer here must contain the identifying integer from 1 to v,
             of the variable selected to be dependent.

        WHAT ARE THE INDEPENDENT VARIABLES?
        Answer:  INDEPENDENT VARIABLES ____;
        Note:  Your answer here must contain the identifying integers of at
             most ten (10) of the v variables, separated by commas.  Naturally
             those numbers must be both mutually unique and different from
             the dependent variable specified in your previous answer.

        [DO YOU WANT A TABLE OF RESIDUALS FOR THIS SELECTION?]
        Answer:  {RESIDUALS;   }
                 {NO RESIDUALS;}
        Default:  NO RESIDUALS;

        IF YOU WISH TO SPECIFY ANOTHER SELECTION AND YOU HAVE NOT ALREADY
SPECIFIED TEN (10), THEN GO BACK TO SECTION 3.7.1.  OTHERWISE, PROCEED TO
SECTION 0.5.3.
```

Figure 1—Sample EASYSTAT text section

reference to an appropriate section in the text for corrective action.

ADDITIONAL CHARACTERISTICS OF EASYSTAT

In addition to this basic operating procedure, EASYSTAT supports a wide variety of options. First, a list of the statistical operations available in EASYSTAT is given in Figure 4.

Second, one's data may be either in free form as discussed above or formatted. If data is formatted, the user is free to dictate what that format is and thereby must describe that format to EASYSTAT. Here, one does *not* give a FORTRAN-like format, but instead gives a graphical description of his data layout. For instance, suppose one has data keypunched with 3 numbers on a card, in columns 1-6, 7-12, and 16-18. Then his graphical description of this format is as follows.

```
Column   1          7          16
        (V*****    V*****     V**
```

It is felt that this alternative to the FORTRAN-like format is easier to learn, more natural to use, and thus less susceptible to error.

Third, one's data may be either on cards or on tape. This is an important feature for those who input data on tape from sources outside the EASYSTAT installation.

Finally, it is important to emphasize that the EASYSTAT text attempts in no way to teach elementary statistics or its methodologies in the various disciplines. We feel that this is best done via a standard textbook. To support this viewpoint, a substantial number of references to standard textbooks are given in the EASYSTAT text. Thus, the EASYSTAT text is naturally used in conjunction with, rather than in place of, any one of a number of standard textbooks in a course of study.

SYSTEM CHARACTERISTICS

In this section we briefly discuss EASYSTAT's design characteristics adopted to meet the system objectives outlined in the Introduction. In addition to those objectives, the following two constraints bore upon these design characteristics:

1. The entire system must run in 108K bytes of main storage on an IBM 360/40 running under OS-PCP

```
//SAMPLE JOB 02220018,LARSEN
// EXEC EASYSTAT
//EASYIN DD *
 MULTIPLE LINEAR REGRESSION;
 6 VARIABLES;
 23 OBSERVATIONS;
 DEPENDENT VARIABLE 6;
 INDEPENDENT VARIABLES 1,2,3;
 RESIDUALS;
 DEPENDENT VARIABLE 4;
 INDEPENDENT VARIABLES  3,5;
 DATA NAME IS SAMPLE;
    44    349    252     88     21     1
    44    141    236    129     56     1
    44    245    236     97     24     1
    45    297    256    111     45     3
    45    310    262     94     20     2
    45    151    339     96     35     3
    45    370    357     88     15     4
    45    379    198    147     64     4
    45    463    206    105     31     3
    45    316    245    132     60     4
    30    313    239     91     10     0
    45    280    225    108     36     4
    35    243    275     95     30     2
    35    365    219     95     21     2
    44    395    215    101     27     1
    29    289    216     85     14     1
    43    396    267    100     39     3
    43    356    274     79     19     2
    44    346    255    126     56     3
    44    156    258     95     28     0
    44    278    249    110     42     4
    30    391    244     92     16     2
    30    424    246     90     18     2
```

Figure 2—Sample EASYSTAT run deck

```
BEGIN EASYSTAT RUN

ANS# ANSWER

  1  MULTIPLE LINEAR REGRESSION;
  2  6 VARIABLES;
  3  23 OBSERVATIONS;
  4  DEPENDENT VARIABLE 6;
  5  INDEPENDENT VARIABLES 1,2,3;
  6  RESIDUALS;
  7  DEPENDENT VARIABLE 4;
  8  INDEPENDENT VARIABLES 3,5;
  9  DATA NAME IS SAMPLE;

DATA SUMMARY FOR DATA NAMED SAMPLE
TOTAL NUMBER OF ERRONEOUS OBSERVATIONS =   0
TOTAL NUMBER OF MISSING OBSERVATIONS =   0
TOTAL NUMBER OF EFFECTIVE OBSERVATIONS =   23

MULTIPLE LINEAR REGRESSION FOR DATA NAMED SAMPLE

SELECTION  1

INDEPENDENT  MEAN       STANDARD      CORRELATION   REGRESSION    STD. ERROR     COMPUTED      BETA
VARIABLE                DEVIATION     X VS Y        COEFFICIENT   OF REG.COEFF.   T VALUE      COEFF.
   1      41.00000      5.94673       0.42171       0.09747       0.04128       2.36107       0.45040
   2     315.34783     85.83790       0.26743       0.00617       0.00292       2.11812       0.41184
   3     250.82609     37.12099       0.20652       0.00798       0.00676       1.18013       0.23026

DEPENDENT
VARIABLE
   6       2.26087      1.28691

INTERCEPT                -5.68460
MULTIPLE CORRELATION      0.58960
STD. ERROR OF ESTIMATE    1.11849

                ANALYSIS OF VARIANCE FOR THE REGRESSION

    SOURCE OF VARIATION       DEGREES      SUM OF        MEAN          F VALUE
                              OF FREEDOM   SQUARES       SQUARES
ATTRIBUTABLE TO REGRESSION        3        12.66556      4.22185       3.37476
DEVIATION FROM REGRESSION        19        23.76919      1.25101
    TOTAL                        22        36.43476

             TABLE OF RESIDUALS

 CASE NO.     Y VALUE     Y ESTIMATE      RESIDUAL
     1        1.00000       2.77042       -1.77042
     2        1.00000       1.35844       -0.35844
     3        1.00000       2.00057       -1.00057
     4        3.00000       2.57876        0.42124
     5        2.00000       2.70692       -0.70692
     6        3.00000       2.33985        0.66015
     7        4.00000       3.83571        0.16429
     8        4.00000       2.62207        1.37793
     9        3.00000       3.20458       -0.20458
    10        4.00000       2.60826        1.39174
    11        0.00000       1.07982       -1.07982
    12        4.00000       2.22634        1.77366
    13        2.00000       1.42233        0.57767
    14        2.00000       1.72858        0.27142
    15        1.00000       2.75909       -1.75909
    16        1.00000       0.65057        0.34943
    17        3.00000       3.08289       -0.08289
    18        2.00000       2.89179       -0.89179
    19        3.00000       2.77585        0.22415
    20        0.00000       1.62667       -1.62667
    21        4.00000       2.30810        1.69190
    22        2.00000       1.60134        0.39866
    23        2.00000       1.82106        0.17894

SELECTION  2

INDEPENDENT  MEAN       STANDARD      CORRELATION   REGRESSION    STD. ERROR     COMPUTED      BETA
VARIABLE                DEVIATION     X VS Y        COEFFICIENT   OF REG.COEFF.   T VALUE      COEFF.
   3     250.82609     37.12099      -0.35069      -0.08436       0.03018      -2.79515      -0.18551
   5      31.60870     15.83075       0.93892       0.96510       0.07077      13.63727       0.90507

DEPENDENT
VARIABLE
   4     102.34783     16.88089

INTERCEPT                93.00172
MULTIPLE CORRELATION      0.95647
STD. ERROR OF ESTIMATE    5.16658

                ANALYSIS OF VARIANCE FOR THE REGRESSION

    SOURCE OF VARIATION       DEGREES      SUM OF        MEAN          F VALUE
                              OF FREEDOM   SQUARES       SQUARES
ATTRIBUTABLE TO REGRESSION        2        5735.34269    2867.67135    107.42948
DEVIATION FROM REGRESSION        20         533.97047      26.69352
    TOTAL                        22        6269.31316

END EASYSTAT RUN
```

Figure 3—Results of sample EASYSTAT run

Class I: Univariate Operations

NAME
TALLY
TABULATION
1-SAMPLE KOLMOGOROV-SMIRNOV TEST

Class II: Two-sample Tests

NAME
2-SAMPLE KOLMOGOROV-SMIRNOV TEST
T TEST
F TEST
MANN-WHITNEY U TEST
KENDALL RANK CORRELATION
SPEARMAN RANK CORRELATION

Class III: Multivariate Operations

NAME
TALLY
TABULATION
SIMPLE LINEAR REGRESSION
CORRELATION
CANONICAL CORRELATION
AUTOCORRELATION
MULTIPLE LINEAR REGRESSION
STEPWISE REGRESSION
POLYNOMIAL REGRESSION
CHI-SQUARE CONTINGENCY TEST
KENDALL COEFFICIENT OF CONCORDANCE
COCHRAN Q TEST
FRIEDMAN 2-WAY ANALYSIS OF VARIANCE TEST
FACTOR ANALYSIS

Class IV: Structured Data Operations

NAME
ANALYSIS OF VARIANCE
DISCRIMINANT ANALYSIS

Figure 4—Operations available in EASYSTAT

2. The entire system had to be implemented within 6-8 months by the designer working approximately 1/3-time on it, and a programmer working approximately 2/3-time on it.

Thus the first decision was to write the *entire* system in PL/1. This was done both to take advantage of the statistical routines already available in the PL/1 SSP (2) and to expedite programming those parts of the system which interpret users' answers. Of course, the PL/1 choice might appear to render the system less transferable than would the choice of FORTRAN IV (at least for the present). On the other hand, however, one would probably need to write some assembler code to support implementation of the string-processing aspects of the system in FORTRAN, thus negating the transferability afforded by "standard" FORTRAN.

Second, due to core limitations, the system had to be highly overlaid and somewhat restrictive in the amount of data that could be accommodated in a single run. The system's organization is shown in Figure 5. EZMAIN is the main control section for the system. In turn, it calls EZEDIT, EZDATA, and the appropriate functional routine. EZEDIT reads the user's answers (problem description) and initializes appropriate control fields. EZDATA reads the user's data into the area cabled *A*. Finally, the appropriate functional routine (e.g., linear regression) is called to perform the required operation and print the results. For this organization, the overlay structure and core requirements are shown in Figure 6. The resulting data limits are that no statistical operation in EASYSTAT can accommodate more than 8000 individual data values (numbers) in a single run and that the maximum number of variables permitted per run is 70.

Design of the system was begun in November 1971 and the version described here was fully implemented by August 1972. Given the same limited resources, it is difficult to see how this system could have been achieved using any other language than PL/1 in a comparable amount of time.

A common argument put forth in defense of Assembly Language as a superior system programming tool to PL/1 is that the resulting code executes far more efficiently. Of course it does, and it always will. But that gain is not without its price—both in implementation time (or cost of programmers) and self-documentation of the source programs. As for EASYSTAT, our tests using typical data show that any statistical operation in the system will run within 1-2 minutes on a 360/40. That time includes all I/O time as well as CPU time. For example, the regression example shown above runs in less than 1 minute. Furthermore, this compiled version of EASYSTAT was *not* optimized for execution speed. Nevertheless, this speed is quite acceptable. The gain which would have been possible had the system been written, in whole or in part, in Assembler seems to be minimal and might well have been totally offset by increased implementation cost and/or time.

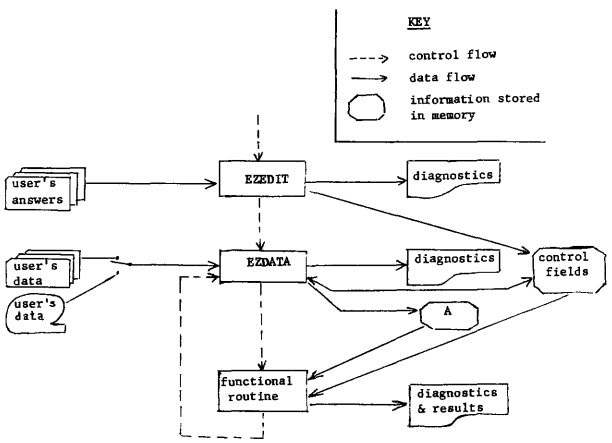


Figure 5—EASYSTAT system organization

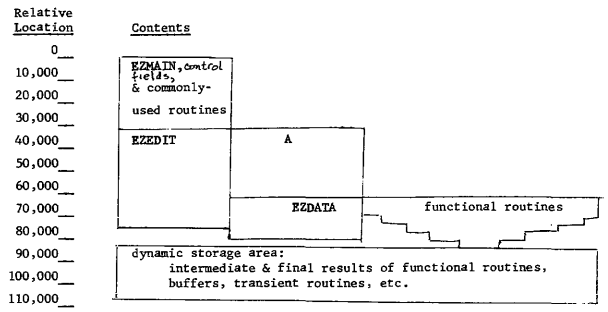


Figure 6—EASYSTAT overlay structure and core requirements

CONCLUSIONS AND RECOMMENDATIONS

Early experience at Georgetown indicates that EASYSTAT will indeed be widely used in support of both course-work and research. Many who have used it have been favorably surprised by the ease and speed with which they can use the system to do their statistical calculations.

This evidence is, of course, by no means conclusive. EASYSTAT is presently keeping account of the day-to-day numbers of successful and unsuccessful runs being made, as well as other information. This information will later assist us in reevaluating and improving the system's performance and ease of use.

We strongly recommend that the philosophy exhibited here be widely-considered, especially by those confronted with the problem of making the computer's power more directly accessible to the non-computer specialist. It is quite feasible to consider, for example, the development of other packages which use the same approach as EASYSTAT—e.g., a text-processing package, or a linear algebra package.

REFERENCES

1. Schucarry, W. R., et al., "A Survey of Statistical Packages," *ACM Computing Surveys*, Vol. 4, No. 2, June 1972, pp. 65-80.
2. *System/360 Scientific Subroutine Package (PL/1) Program Description & Operations Manual*, Form #*GH*20-0586, IBM Corp., 1968.

ACID—A user-oriented system of statistical programs

by R. A. BAKER and T. A. JONES

Esso Production Research Company
Houston, Texas

A common problem in any scientific or business effort is to organize large masses of data for human interpretation. Today we will describe a program system which is designed to do just this; we call it ACID, which is an acronym for Automated Classification and Interpretation of Data. Our system may not be unique, but we believe it is well-designed. The user can get his work done with this system, which is set up as an "executive design"—one program controls a lot of other programs.

The executive, or interface, program puts a friendlier face on the system for the user. It works with him, not against him, which for many users is a unique experience. Too often we find a puzzled and bedraggled user on the point of either rage or defeat in his contest with Program X, which has stifled every effort on his part to communicate what is to be done. Running the cantankerous program becomes his major project, and the objectives of the real project get lost in the effort. This is less likely with ACID because the user gets in and out quickly. The major problems fall to making the analyses, not running the program. This is particularly a problem if the user must run several programs in succession.

ACID is focused on a somewhat heuristic pattern recognition problem (in the general sense of that term) although the concepts discussed here are certainly not limited to this. In the pattern recognition studies, the computer organizes the information, and the user tries to recognize and interpret the organization. ACID may be instructed to provide several alternative organizations, perhaps with different interpretations. The heart of the classification scheme we use is cluster analysis. This technique is as old as large-scale computer processing, but is none the worse for wear, as we have had good success with it. One of the secrets of success is having turnaround within the time frame of the project and the user's patience; this includes pre-processing of data as well as the actual analysis. The user will not experiment with his computerized tool if it's all he can do to get it to work once or twice. If he does not experiment, he will not be likely to get the results he is looking for. So far ACID has kept well within the patience and project times of its users.

As an example of the kind of project ACID is designed for, we studied Galveston Bay, a major estuarine system in the vicinity of Houston. We have 85 sampling stations; approximately 50 species of mollusks were observed and counted at the stations. Our objective was to find a pattern among these samples by use of the contents of the biological collections at the stations. One might think that it would be fairly easy to quickly and efficiently solve such a problem. One need only go to the published literature for programs that will do the job. After all, cluster analysis is some 15 years old, and there are at least 20 programs available for doing the job. That was our original thought and course of action.

We obtained a program and ran our data with it. And ran it. And ran it. We soon found the practice to be nowhere near as simple as the concept. First our data arrays wouldn't fit the program dimensions. So we trimmed some data vectors. Then we couldn't transpose the matrix for an alternate analysis. Then we found the data formats were not usable with other programs, creating added complications in the computation of simple statistical summaries, making data transformations, and in the use of different cluster and multivariate analyses.

After great tedium we managed to perform the desired cluster analyses, and obtained four sample groupings. The procedure divided the bay complex into four areas: ship channel and tidal pass, forebay, backbay, and brackish bay-margins. This concluded the classification phase. Next, in the interpretative phase, we wished to determine the species that defined each of the groupings. This required the use of other programs, thus repeating the above problems with formats, dimensions, etc. Again, after much effort unrelated to the interpretation, we managed to find that only 6 species were needed in generating the pattern in the bay complex.

We found the actual analysis to take less time than the preparatory work of getting the data ready. And we're talking about data manipulation, not data reduction. This made cluster analysis a tool of questionable value. Was it worth months of effort, especially when much of this time would be spent in re-coding and keypunching and so forth in getting ready to run the analytical program? We decided "no," and that's why ACID was developed. Over a period of a year-and-a-half we wrote some 25 programs —large and small—and toward the end of that period combined them into the executive framework of ACID. Most of the programs in ACID are actually auxiliary to the cluster analysis step.

ACID is essentially a sub-operating system. It does for the applications user what the operating system does for the programmer. It keeps track of those essential but nuisance items such as the location of programs and data sets within the system. The user may have three or four data sets each at various levels of transformation in the system. If he wants to operate on a particular data set, he prefers to call it up by a code name, not by location, because the physical location of the data set on tape or disc is of no interest to the user. The system also provides useful services during operation, such as error checking. For example, within the limits of Fortran, ACID attempts to tell the user what happened before going off the air with an abnormal end. It also checks input parameters and data sets for certain types of errors. In addition, if the user is requesting more storage than is available, ACID tells him how much more is needed. It also dynamically dimensions arrays, so as to allow virtually any size data set to be processed. As with the operating system, most of what ACID does as a system is invisible to the user. He sees only the functional programs that were stand-alone before being incorporated in ACID.

The executive design gives us the flexibility of a library and the convenience of a single program. The basic system is set up with an executive that can call any of the individual programs. Figure 1 shows the structure whereby each of the operational programs is independent of the others and at the same level. Each is interfaced to the executive by a small driver subroutine. The user inputs a control card with the name of the desired option on it (e.g., STATISTICAL SUMMARY). At this point the executive calls the appropriate program. This program reads any necessary controls and performs the analysis. Control remains in this program until a special control card instructs a return to the executive. The various options are processed sequentially.

The control language used is basically hierarchical in nature (Figure 2). Some commands go to the executive program and some to the functional programs, and in general the commands are kept on separate cards and follow a sequence from highest to lowest level in the structure. For instance, we might wish to perform transformations on some of the variables in our data set, followed by the computation of some simple statistical summaries. The TRANSFORMS card instructs the executive to call the appropriate program. The program control then is used to define the data set required, and to set other controls. Following in the hierarchy comes the controls for each of the individually desired transformations. The controls for a second data set may follow. The end-of-program card returns control to the executive; by this time, two new data sets, 1A and 2A, have been created. The STATISTICAL SUMMARY card calls for the next program option to be executed. This sequence is similar to that of the computer operating system, and we have found it to be effective.

The ACID data sets must be easily accessed by the user. In ACID most of the data are in the form of rectangular arrays, with samples arranged along one side and variables across the top; the body of the array consists of the observed values. The data sets are stored as arrays in binary form on a sequential file. Tapes are normally used if the data sets are to be retained permanently. In addition to the data, there is header information for each data set. This information includes a code name assigned to the data, the dimensions of the data array, and code names to identify each sample and variable. These names are invaluable when the user is interpreting his results, and are also conveniently carried with the data in this form.

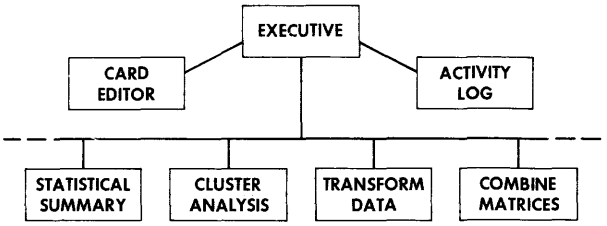


Figure 1

Several data sets may be placed on the same file, and new data sets are normally created in the editing steps. The user assigns a four-character code name to each data set as he instructs the program in its operation. The user thus inputs the name of a data set to operate on and a name for the resulting data set. The program finds the first data set on the file and inputs it; it then operates on the data and forms a new data set. This new set is placed on the data file under its own name; the original data set is also left unchanged on the file.

```
•
•
•
CARDS FOR PREVIOUSLY EXECUTED PROGRAMS
•
•
•
TRANSFORMS
        PROGRAM CONTROLS, DATA SET 1
            M TRANSFORMS DESIRED
        PROGRAM CONTROLS, DATA SET 2
            X TRANSFORMS DESIRED
END OF PROGRAM
STATISTICAL SUMMARY
        PROGRAM CONTROLS, DATA SET 1A
        PROGRAM CONTROLS, DATA SET 2A
END OF PROGRAM
•
•
•
```

Figure 2

One of the original difficulties we found in running several independent programs was that we invariably had data sets with dimensions that exceeded those of the programs. The ACID system solves this by using a form of Dynamic Core Allocation. In this case, no arrays are defined directly in the programs. Whenever a data set is input, the number of rows and columns in the data set are used to compute the actual dimensions required for the program to process that data set. Then a routine is called that allocates the required number of words of core and defines the arrays. This core is released after the data set has been analyzed.

In addition, the executive structure allows us to modify and add programs with little disturbance of the program system. The functional programs are essentially stand-alone applications programs. When they are put into ACID, the main change is in their input sections. In the system they read data sets formatted and maintained by ACID on a sequential file, rather than reading decks from the card reader. In addition, a short driver routine is used to allocate the array space dynamically. Thus, we have the ability to develop functional programs independent of the system or to obtain programs from the published literature or elsewhere and add these programs to the system.

As part of the internal structure of ACID we have consciously kept the I/O as high up in the subroutine stacking as practical. This makes it much easier to come into a program and make a patch because the route from input to output is shorter and more direct. It also makes any data-editing more apparent, and the editing can be specific to the input data. Whether making a modification to the program or tracing a bug, a direct route from input to output makes the task much simpler. On the other hand, we found it impractical to concentrate all of the I/O in one program, such as the executive, because that reduced our ability to treat the programs in modular fashion. Program modification is naturally much simpler if the modification in no way affects another program.

Along this line, we might mention a "good" idea that didn't work out. An unsuccessful feature, since abandoned, was to concentrate all the output, error messages in particular, into one program. The concept was to store several error codes and then call a program at the end of execution that would interpret the codes and print the messages. Changing a message would be a simple matter of changing the error-printing routine. The functional programs would not have to be changed at all. The idea is not new with us, and we expect that it might work in some environments. We found it to be a poor idea for ACID. In the first place, we found that the error message format statements were very useful in supplementing our internal documentation. Secondly, we found that considerable library effort would be required to maintain an error printing program since every new option in ACID would probably require an addition to the error-printer. Thirdly, we found that the probability of an error message changing was low once it was established. This slow-changing character pretty well obviated the need for an error printer.

One feature of ACID which seems common to library maintenance programs but that is rare in applications programs is the activity log. The activity log keeps track of which functional programs have been called, in what sequence they were called, whether they operated error-free, and the data of the call. The activity log tells us how often each function in the system is used. It provides an objective evaluation tool. Keeping track of the sequence of functional steps gives us an idea of what functions might be combined. We might mention that the greatest problem in the interpretation of the activity log seems to be user habits. Once a user establishes a sequence of steps that work he's unlikely to change that sequence even if he no longer needs some of the steps. We cannot say that the activity log has been a resounding success, but it does provide interesting information.

Another useful feature in ACID is a card editing program. The card editor lists all the input cards before any action is taken on them and makes whatever general editing operations are possible at that level. It also has the capability to generate control records from a more general input instruction. We mean rather low quality editing at this level; really good editing can be done only by the functional programs, which know specifically what to look for. The best that can be hoped for from the card editor is that it will catch the gross absurdities which would totally confuse the functional programs. Both the activity log and the card editor are part of what might be called the "executive suite"; their existence is useful only because the functional programs have been put together as a system.

In conclusion, we have found the executive, as exemplified by ACID, to be very effective. It brings us previously lacking user orientation. It brings us flexibility in large program systems. It brings us the opportunity to grow in a changing environment. We have four other comparable, but non-statistical, program systems with this same design currently operating. The design has been successful in every case; the systems are in use and rarely out of order for repairs. We recommend the design as a natural extension of the operating system.

Graphics and engineering—Computer generated color sound movies*

by LARA H. BAKER, JR. and EDWIN K. TUCKER

University of California
Los Alamos, New Mexico

INTRODUCTION

Over the history of computing, substantial effort has been directed toward improving the throughput of digital computer systems. Only recently, however, has a significant effort been devoted to enhancing the intelligibility of output. Computers are excellent tools for the inhalation of massive quantities of input and the disgorgement of massive quantities of output. Frequently, as applications and systems programmers, our task is to create some order out of the chaos. The need for this effort is nowhere more apparent than in the display of computer-generated engineering data.

Engineering calculations have several characteristics whose ramifications are not obvious. The calculations are in general reasonably straightforward to define. Compared to problems in cellular microbiology, for example, the parameters of most engineering problems are fairly well defined. This would lead one to expect that the presentation of their solutions would be equally straightforward. This is not the case. Further, many engineering problems are dynamic, time-variant problems. Their solution is a function of time and time may be the most critical parameter in the solution. The presentation of dynamic, time-variant data in a static, two-dimensional medium can be worse than useless. Finally, engineering systems are usually complicated, if not complex. This implies that whatever intelligible information exists in the output may exist in a large dilute quantity.

The data presentation difficulty is common to many forms of problem solving. Engineering problems suffer particularly from this bottleneck because of the quantity and type of their solution data. The solutions to problems involving optimization may often be simply displayed, but some classes of problems, e.g., time history calculations, must be examined by the engineer at many points in their development. The engineer is also faced with the problem of extracting a few numbers out of a great many and trying to retain an overall picture from these numbers. These data are generally far too valuable an analysis tool to give up, but faster, even if less precise, methods of data scanning are required. Another significant obstacle facing the engineering analyst is the necessity for the perception of trends in the data rather than their precise numerical value. These trends are frequently hidden by the inherent synergism among the various problem parameters. Creating order out of chaotic output, then, requires the development of efficient, effective and feasible techniques for data presentation.

PROBLEM PARAMETERS

Time frames

One of the greatest difficulties with engineering problems is the time frame in which the real world operates; very few engineering problems occur at the optimal rate for examination by an analyst. The occurrence rate of the problem is usually either far too rapid or much too slow. Some of these problems involve a long period of real time; e.g., heat transfer through a shipping container may take hours of real time. Some of the problems are very close to the analyst's time frame; e.g., the analysis of a building's response to an earthquake loading may be within one order of magnitude of the analyst's speed of interpretation. Some problems, for example, the catastrophic failure of an overstressed cylinder, may take place in such a short time (i.e., milliseconds) that their duration must be stretched to allow the analyst any comprehension at all.

Since the amount of time which an analyst is physiologically capable of using to extract information from a display is relatively fixed, the display techniques used must be flexible enough to provide a meaningful transition from the real time to the analyst's time. While the necessity for spreading the real time involved in a very rapid event is obvious, the requirement for compressing the time spent in a near real time event is more abstruse. The amount of time an analyst can spend concentrating on a relatively slowly changing scene is much shorter than one would estimate; that is, reasonably intelligent people have a low boredom threshold.

* Work performed under the auspices of the U. S. Atomic Energy Commission.

Nontemporal variables

The variables to be examined in an engineering problem are obviously functions of the analyst's interest. But regardless of the particular problem, a great number of interrelated variables are usually involved. Expanding on the examples given above, heat transfer problems are quite amenable to computer solution, particularly in the case of the time dependent flow of heat through a complicated object. The parameters of interest in this problem usually include indicators of the temperature and state (i.e., solid, liquid or gas) of the object, the rate of heat flow, and various representations of external boundary conditions. These parameters are of interest at essentially every coordinate within the object. The second example involves the response of a multi-story building to an earthquake or blast loading. Here the parameters of interest include the building's shape and velocity, and the location and magnitude of material deformations (both plastic and elastic), as well as the stability of the entire structure or portions thereof. Again, it is the continuous interaction of these variables that is of interest to the analyst. In the third example, we consider the deformation of a heavy multi-material cask under impact loads. Parameters of interest in this problem include the position of the cask, its stability, the state of stress around the point of impact, and the amount and rate of structural deformation wherever it occurs.

All of the above problems are time dependent and multivariant, and generally require a massive quantity of information for the useful interpretation of the parameters of interest. Frequently, the numerical values of many of the variables are generated in tabular form, usually on microfilm, since these values are the most detailed, but least comprehensible form of output. Certain of the detailed numerical results are necessary to the ultimate solution of the problem. The trick is to identify which ones. Due to the time varying nature of the problems, static graphs are almost as bad as numerical tables for data presentation. The dynamic nature of the problem must be reflected in the medium used for data presentation.

CORRECTING THE BOTTLENECK

There are several approaches to "solving" the problem of the data bottleneck. The simplest "solution" is to stop using computers, i.e., to abandon any attempts to solve big analysis problems. Alternatively, in order to cut down output, the problem can be oversimplified or some of the output deleted. A third approach involves hiring smarter analysts, that is, people who are willing and able to figure out what is going in all that pile of numerical output. In all of these approaches, the supposed solution really means ignoring the bottleneck rather than alleviating it. But the evaluation of computer output need not be a tedious process of numeric interpretation. The viable approach is to increase the comprehensible information density of the output we currently have.

In attempting to increase this comprehensible information density, the industry has gone from strictly numerical output (whether on paper or microfilm) to graphic displays. But while static graphs can improve the presentation of static information, it is only with dynamic displays that a significant blow is dealt to the bottleneck problem. Interactive CRT displays have significantly increased the effective information density of computer output, and for a large class of problems offer the best available method of data display. Only when the computer time required to generate a solution prevents interaction is this technique patently unusable.

Another quantum jump in information density came about with the widespread use of color graphics. The ability of the human eye to perceive and distinguish different colors can increase the comprehensibility of a display significantly. In many cases, color graphics provide the only technologically feasible means of presenting a particular display.

One of the more straightforward and effective methods of showing the time-varying phenomena in engineering problems is the use of motion pictures. Computer driven microfilm plotters provide efficient methods by which to generate these movies. Their development has received much attention and their use is now routine. The recent development of one pass multi-color COM considerably increases the effective information content of this output medium.[1] Motion pictures allow the analyst to view the problem solution at a pace of his choosing rather than at the pace of the solution time or of the real world. Data presentation by computer movies is not a theoretically difficult task, but neither is it trivial when one considers the man and equipment requirements needed to accomplish it. However, motion picture film is probably a medium of data presentation which will assume greater importance in the future. Indeed, it presently enjoys a high level of utilization at the Los Alamos Scientific Laboratory, where computer film usage approaches 400 miles per year.

While the above techniques have provided greatly increased comprehensible information density for computer output, the bottleneck problem has only been alleviated but not solved. Perhaps the problem will never be completely solved. However, techniques for using multisensorial output can go a long way toward optimal data presentation.

THE STATE OF THE ART

Of the five primary senses, two are highly developed in man. Vision and hearing seem to be the major channels through which he receives information about his environment. These two are therefore the logical choices for data presentation using current technology. The use of the other three senses as channels for communication with

computers is being studied for specialized applications, but these senses have not received the intensive study that has been devoted to vision and hearing.

While both the eye and the ear are capable of accepting great volumes of data, they are not equally receptive to all messages. The eye excels in the perception of spatial relationships, but hearing is the better temporal sense. Vernier visual acuity is the best our senses can offer for fine discriminations. Yet only the ear can perceive the high information content of voice inflections and emphases.[2] Audio output alone from computers can be quite useful in certain applications. For instance, it is currently being used to answer "disconnected-number" queries by the telephone system in many parts of the country. But for complex problems, one must consider the nature of the various messages that can be involved in the output before assigning particular information to a particular sensory medium.

Due to the complexity of many problems, it is highly cost effective to selectively combine sight and sound to provide a sufficiently dense but comprehensible output. This combination is now being used at the Los Alamos Scientific Laboratory, where a method has been developed for the production of computer-generated optical sound tracks on a microfilm plotter. Pictures and sound can be produced simultaneously in one pass through the plotter. The sound tracks can be used to facilitate the interpretation of data that is presented visually.[3] However, from the programmers' point of view, the addition of another channel for data presentation is as important as facilitating the interpretation of data. Not only can a sound track present explanatory and narrative material efficiently and appealingly, it can also be used to represent additional data that might otherwise be lost. For example, it is always difficult to clearly represent the movements of a large number of particles within a bounded three-dimensional space. If, however, the collisions of particles—either with each other or with the boundaries of the space—are represented by sounds, the interpretation of the phenomenon is greatly facilitated. This is feasible only if the sound track is computer produced and not "dubbed in" after the fact.

SUMMARY AND CONCLUSIONS

The enormous quantities of relatively unstructured computer output associated with large simulation and analysis codes make data interpretation a very forbidding task, particularly in the engineering disciplines, where the meaning of the data can be so obscured by the density of numbers that the value of the analysis itself becomes doubtful. Furthermore, the effective presentation of dynamic, time-variant data demands handling that cannot be provided by a static two-dimensional medium. Dynamic interactive CRT displays and computer generated motion pictures are two techniques that have substantially improved data presentation, and the addition of color even further increases the comprehensible information content of these displays. But optimal displays will only come about through exploitation of more than just man's visual sense. Current developments allow the use of both vision and hearing as channels for communication with computers, but whether this will prove to be the truly optimal display or not will depend on man's ingenuity in broadening the concept of computer graphics.

REFERENCES

1. Baker, L. H., Donham, B. J., Gregory, W. S., Tucker, E. K., Computer Movies for Simulation of Mechanical tests, *Proceedings of the Third International Symposium on Packaging and Transportation of Radioactive Materials*, Richland, Washington, Vol. 2, pp. 1028-1041, August 1971.
2. Geldard, F. A., "Some Neglected Possibilities of Communication," *Science*, Vol. 131, No. 3413, May 1960.
3. Tucker, E. K., Baker, L. H., Buckner, D. C., "Computer Generated Optical Sound Tracks," *Proceedings of the Fall Joint Computer Conference*, Anaheim, California, pp. 147-152, December 1972.

Graphics and computer-aided design in aerospace

by RONALD E. NOTESTINE

Lockheed-California Company
Burbank, California

INTRODUCTION

In 1964, certain hardware developments in graphics equipment were announced by IBM. The interest of management of the Lockheed-California Company was immediately aroused. Possibilities for the development of highly effective programs through the utilization of this equipment were foreseen and action was started to take advantage of this new technology.

A task force was set up to study various applications and make recommendations. Representatives from Structures, Aerodynamics, Loads, Stress, Project Design, and Computer Services participated in this work. This study resulted in the delivery of an IBM 360/40 with one IBM 2250 scope in 1966. This task force also recommended the implementation of certain initial programs. The prime criteria for their recommendation being immediate need and use for the design processes associated with aircraft.

From these early beginnings, we have progressed to a Model 50, 65, 75, and finally the present shared 360/91. On the 360/91, we have fine tuned the system to a high degree of effectiveness. We feel it is our most proficient and cost-effective configuration.

CONFIGURATION

The 360/91 on which we are presently operating graphics has 2 million bytes of high speed core. Those devices dedicated to graphics include a 2314 with 8 disk drives, 2-2301 high speed drums, a 1403 printer, and 4 tape units. One 2250 is located in the computer room and is primarily used for graphic program development, batch program checkout and graphics systems maintenance. Three more 2250s are located in the same building, but in a user environment. Eight scopes are located in an engineering building which is about a mile from the graphics computer. They are connected to the main frame through a 2916 Long Line Adaptor and coaxial cable.

Graphics occupies 528 thousand bytes of the total of 2 million bytes of high speed memory. The rest is used by the system and the batch streams which are sharing the computer with graphics. Normally, at least three batch streams of programs are concurrently being processed. The graphics region is split into four partitions, which in turn drive the twelve scopes. This is done by utilizing a data roll technique for our design package, which allows multiple scopes to be run out of one partition. The design package will be described later.

PROGRAMMING CONSIDERATIONS

Certain programming considerations had a strong influence on the direction taken early in the development of computer graphics. These considerations had their effect not only on the selection of applications, but on how those applications were developed.

The first consideration was to provide a tool that was user oriented. The displays, and the actions taken should not be completely foreign to him. If he is a designer, and the scope is his drafting board, then the methods of construction on the scope should be basically the same as he would use on the drafting board.

The prime reason for placing a user in direct contact with a high speed computer is to be able to achieve a high degree of interactivity. The application should be one where close interaction is essential to the effectiveness of the work to be performed. In addition, the program should use its ability to interact to assist the user in doing his work. This can be done with tutorial messages which always leave the user knowing what he did last, and what his choices are to do next.

Cost had to be kept low in order to make cost effective a wide spectrum of applications. The numerator of the basic cost equation consists of the driving computer, the scopes, and the programming support. The number of scopes becomes the denominator in this equation, and as their number increases, the cost per scope is driven down. Our present need is for 12 scopes, however, the system we have today could be increased to 16 or even a 20 scope system with the only increase in cost due to the scopes themselves and a significant decrease in cost per scope.

The tool that has been given to the user should be responsive. Neither his thought processes nor his actions should be slowed down by the system. Also, the responses should be consistent. Inconsistency in response time, even though fairly fast, can be very frustrating and disrupting to the smooth flow of thought processes.

Another way to reduce costs is to share the equipment with batch processing. This sharing can take place in two ways. The first way is by operating in a time sharing environment where the batch stream is soaking up the CPU time which is left over from graphics. However, graphics tends to be a prime shift capability which sits idle the rest of the time. Therefore, the second way is by scheduling batch processing to fully utilize these off shifts. There are several cost advantages which can accrue. The first is a reduction in the computer cost which is in the numerator of the previously mentioned cost equation. Because graphics does experience peaks and valleys of workload, it would be desirable to be able to increase or decrease without a major impact on cost. Sharing the computer allows a greater flexibility for change.

A tool must be reliable to be effective. This reliability manifests itself in two ways. The tool must not only be available, but it must be continuous. A one minute downtime may have no effect on a batch user (he probably won't even know about it), but a one minute down time can have a major impact on all graphics users. The system should have quick and complete recovery for the user.

Because the users are usually located at a place other than where the computer is, the scopes should be capable of remoting. High speed transmission equipment had to be used in order to satisfy the volume and response requirements of the equipment.

USER CONSIDERATIONS

Graphics systems were not developed in a research environment at the Lockheed-California Company. Our present uses are cost effective and have been since the early stages wherever they were applied.

First it was necessary to plan and select the proper graphics applications. These applications were selected on the basis of need, and this activity was participated in by representatives from all potential user areas.

Our new development was confined to programs having broad, immediate day to day use. We did not want to fall into the trap of planning for what we thought might be needed a year or two years down the road. This too often results in finding that the problem has changed and that the program does not now solve the present problem.

We had to assure centralized forecasting of workloads. If we were going to have the equipment available at the proper time, we had to know what the needs were.

Uneven workloads were smoothed out by either planning new work for the valleys, extending scope time available, or modifying the work plan in some cases.

The scope time was in all cases scheduled and coordinated in order to assure a high degree of utilization. If one user is unable to utilize his time, another user is always available to take up the slack.

We utilized a swing shift whenever possible. Not only to take care of peak loads, but to reduce cost. Scope hours go into the denominator of the cost equation.

Adequate training of the proper type of individuals was a requirement. We found that many good, competent engineers did not relate well, did not work well in an interactive environment. Others were extremely effective in their capability of utilizing the equipment. The Lockheed Training Department participates in this extremely important function.

We have developed a comprehensive system for reporting the highly essential statistics generated by Graphics. There are accounting programs that generate the costs of scope time so that it may be charged back against the user. There are programs that generate reports on utilization which may be used by management for measuring past performance and planning schedules and workloads. And there are programs which give information related to performance. What is the response being experienced? How many interrupts per user are being serviced? These help our programmers to improve the performance of the various graphic applications.

ARRANGEMENT OF SCOPE FACILITIES

At present, all of our scopes, except one, are located in the user environment. Proximity to the actual user was deemed necessary so that the engineers and other users would be able to have quick and easy access to the powerful tool. Eight scopes are located in our commercial engineering building and support over fifty engineers on a two shift basis. They not only utilize analytical programs, but prepare drawings for production release in fuselage interior, wiring diagrams, floor beams, and other structural applications. One scope located in another area is manned by engineers doing surface development which is known as lofting. Another is used to prepare tapes for numerically controlled milling machines which cut parts and tools for manufacturing.

APPLICATIONS

We have divided our program development into two general area—design analysis and design drafting. Design drafting includes all work on the Computergraphics Augmented Design and Manufacturing System (CADAM) which is the powerful, general purpose program for design, drafting, numerical control, and lofting. It also includes all of the related system of supporting programs. The design analysis area covers all of the other graphic programs. These include structures analysis, simulation, scrolling, data display, editing, and other programs.

SIMULATION OF AIRCRAFT FLIGHT CHARACTERISTICS

The Interactive Continuous Systems Modeling Program (ICSMP) is a problem oriented program designed to facilitate simulation of continuous processes on a digital computer. The components of the continuous system for

both input and output are represented graphically on the graphic display. It is an adaptation of IBM's CSMP digital simulation language to graphics. Through the use of this language, both static and dynamic simulations are possible. This program has enabled us to simulate closely actual flight conditions. For example, trim, sideslip, wind, ground runs, and take-offs can be simulated for various aircraft configurations. Even feel or touch systems employing friction, detent, spring action or damping such as control column, wheel, or rudder pedal response are possible. The program provides for online editing, correction, display of graphs, and extensive operational interaction through the modification of modeling parameters.

ANALYSIS OF AIRFRAME STRUCTURE

The Two Dimensional Structures Program was developed to complement our larger batch computer programs. The program can analyze any two dimensional structure which can be modeled by assembling axial elements, beam elements and shear panels. It is also capable of analyzing shell supported rings or frames. External loads must be in the plane of the structure. In the case of rectangular panels, the engineer can insert special data on the panel to provide for cutouts, tapered stiffeners, and variations in thickness. Modifications may be made at anytime to any of the input data. Some typical modifications are (1) changing geometry, (2) changing external loads, (3) changing the section properties of any member, or (4) changing support conditions. In addition to modifications of the basic structure, new members and supports can be added to the existing configuration.

ANALYSIS OF LINKAGE MECHANISMS

The Spacebar program was developed to provide for the analysis of linkage systems in three dimensions. The program provides for the construction, on the graphics scope, of an operable or movable mechanism train made up for one, two, three, or four classic four bar units tied together in tandem and operating in three dimensions. Also provided are means to determine internal loads at any mechanism position, displacements under load and input-output curves resulting from the mechanism motion.

The program constructs a discrete element mathematical model of the composite mechanism and projects it in two views onto the plane of the graphics screen. It is viewed as a series of nodes, joined by elements of fixed length and relative position, all moving on the screen in accordance with the mathematical relationship of the physical mechanism in three dimensions.

The engineer establishes the geometry by describing the plane of motion of the crank arm and giving a line of reference for angular measurements within the plane. He then establishes the relative position of the linkage nodes and their lengths. The mechanism can then be dynamically operated in three dimensions by rotating the bell crank and producing the input-output curve. The program provides the means to determine internal loads at any mechanism position and the displacements under load.

INTERSECTING BEAM PROGRAM

The Intersecting Beam program is another fully operational program available in the Computer Graphics Library. Its primary purpose is to mathematically define a discrete element structural model of two limited series of parallel beams intersecting at mutual right angles, to apply concentrated loads at the points of intersection, and to compute the resulting bending moments and deflections at those intersections. All the beams are straight and lie in a single plane, the plane of the display screen. The loads are applied perpendicular to the grid of beams at the intersection points. Deflections and bending moments are measured in the same direction. Uniform loads must be distributed to the node points as concentrated loads.

The necessary input data includes the applied loads, sufficient information to fully describe the geometry of the grid desired, and the stiffness of each beam segment. All information may be entered at the screen by altering the data of a standard grill called from the data library.

In addition to the necessary minimum geometry and stiffness information, two options of additional data are available. Up to a maximum of fourteen flexible supports may be used to give intermediate support to the grid at the selected beam intersections. The end fixity of the beam may also be provided at the scope to vary the beam ends from full fixity through continuous beam to a condition approaching simple support with no fixity.

This program was used extensively in designing the floor beam structure of the Lockheed L-1011 TriStar.

CADAM

The **C**omputer-graphics **A**ugmented **D**esign **A**nd **M**anufacturing System (CADAM) is a complete system of programs which give design, drafting, lofting, and numerical control capabilities from design concept to manufactured product through computer graphics. It is the result of eight years of development testing and use, and has been effectively applied wherever design input is required. Basically, it is a graphics drawing program, however, its capabilities have been developed to the point where, to my knowledge, I believe it is the most effective and productive graphics program in use.

Let's check some of its capabilities. CADAM is based upon descriptive geometry. Information is stored within the computer in the form of a math model which is retrievable for various uses in the system. Complex shapes are readily constructed. Straight lines, circles, ellipses, splined curves, etc., are readily available by the selection of a function key on the console to enter the proper mode. Design sketches can be translated into precise engineering drawings at considerably higher speeds than by the manual methods.

CADAM provides a common data base for all user disciplines. Design models may be viewed at once by retrieving them from the data base storage and displaying them on scopes in different function areas. Several systems in one model can be developed simultaneously and independently with no loss of time.

With CADAM, views can be stacked, separated, or otherwise located within a drawing perimeter in minutes —the designer sees immediately what he has done. If necessary, he can take immediately any further action required. Batch processing operations would require hours or days for successive adjustments.

Accuracy to at least four decimal places is always obtained with CADAM. The machine plots which it produces are highly accurate and may be produced at any scale.

Interfacing components or details may be shown simultaneously on the scope to assure compatibility. Conversely, an assembly can be broken down into separate details, as may be required in the manufacturing process. It is done accurately, and in a matter of minutes for each detail.

By using CADAM, a detail like a bracket, need be constructed only once and still be positioned and repeated on a drawing wherever needed. One may change sizes and dimensions automatically without distortion.

Provisions in the program permit variations in scale or size to be readily extracted from master drawings to fit individual needs. Changes to drawings can be made accurately, and in minimum time without reconstructing unchanged elements.

A complete library of standards, symbols, and details can be stored in computer memory and then called up by the user as needed.

Error detecting logic within CADAM tends to reduce the incidence of human and construction errors. Interaction with the scope display allows the operator to detect an error in his design without resorting to hardcopy print. If he makes a syntax or geometrically impossible error, a message to that effect will appear on the scope so that immediate, corrective action can be taken.

A key code in CADAM makes it possible to automatically control the release of drawings and prevent unauthorized changes. This makes the data available only to authorized individuals.

Multi-axis numerical control programming is available through CADAM. The interactive computer programs are designed to help the user develop a sequence of operations on a punched tape which in turn will be used to direct a specific machine tool to produce a part swiftly, accurately, and with a minimum of tool tries. It is now possible to produce a machined part straight through from the design on the scope to the cutting of the part on the numerical control machine.

CADAM makes available to the user the ability to dynamically display the cutter centerline path and the part geometry in both the plan and elevation views simultaneously. An image of the cutter actually moves along the cutter path on the screen. This permits ready and easy editing of the cutter path. If the programmer notes an error, he can go backward or forward at any time to check the sequence of cutter motions. Errors of small magnitude are eliminated at an earlier stage by analysis verification of part geometry. In actual use, we have found 30 percent of the control tapes required no change to be put in production. The total number of tool tries was reduced by 50 percent.

CADAM allows the complete definition of the surface geometry of a vehicle such as an airplane. This process is known as lofting. Its use falls mainly in three general areas. First is the building of contours and the smoothing of these contours from raw data. Next comes the interrogation as required of these previously defined contours. Finally, there is the assembly of related contours into layouts suitable for plotting as undimensioned loft drawings.

At present, work is proceeding on developing a system to generate parametric surfaces. This program is needed to enhance our capabilities in the generation of surfaces for compound curvature areas of an aircraft such as fillets, nacelle leading edges, and other difficult surfaces.

CADAM SYSTEM

One of the particularly unique features of CADAM is the ability to operate multiple scopes from a single region of memory in the computer. Up to seven scopes can be operated from a single region with no noticeable degradation in response time. As many as ten scopes have been operated from a single region, but certain action on the part of the users can cause lengthy and somewhat unacceptable response times. Each of the multiple scopes operating in this data roll mode is working on a different drawing, but to each user it appears that he is the only one using the computer.

The interactive graphics program requires only 126,000 bytes of high speed memory. Of that memory, 84,000 bytes are rolled to and from high speed drum storage. The

Figure 1—These scopes are located in an engineering environment approximately one mile from the graphics computer.

Figure 2—Here is a loftsman employing manual methods laying out a contour through the use of a flexible spline.

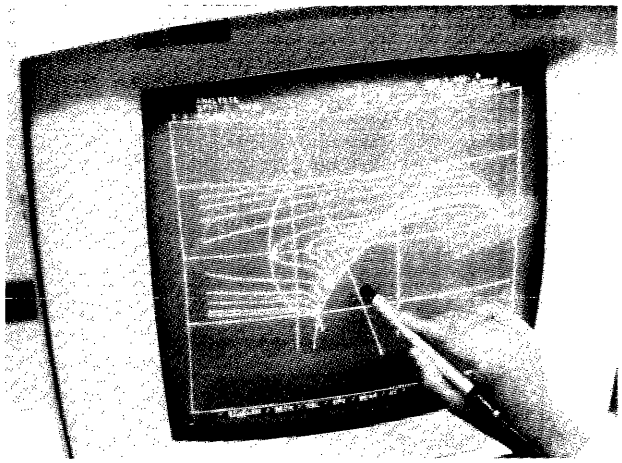


Figure 3—Through the use of CADAM the loftsman is shown laying out contours including mathematical splines on the graphics scope

remaining 42,000 bytes contain the input-output buffers, the control monitor, the attention queue and the other fixed parts of memory.

A comprehensive system of saving the drawing files has been implemented. Even though the drawing is released in its normal hardcopy form, we safeguard the drawing while it is being constructed and after it is released. Both drawings and programs are saved on a daily basis, a weekly basis, and finally on a monthly basis at a remote site.

If the computer or the system goes down during the day, a "warm start" can be made to recover the drawing. The drawing will be in the same condition as it was the last time it was rolled to the drum storage. Within seconds after the computer is up again, the user can again be working on his drawing. Occasionally it happens that the drawing cannot be recovered, and the user has to go back to the drawing file or start over. For this he uses what is called a "cold start."

The drawing files are managed by the user (i.e., the engineer, numerical control programmer, or loftsman). He may purge a drawing or file a drawing as his needs dictate. Normally, only one man in an area is assigned the task of data management. The user area is therefore responsible for and does all the work in maintaining their drawing files.

The interactive graphics program of CADAM consists of 60 percent FORTRAN programs and 40 percent Assembly Language. The assembly language programs are those which are most frequently used and require very fast execution. It is in this way we achieve the high effectiveness and the fast response time of the system. Common response time is less than a half a second. System up time is greater than 96 percent where the system up time is defined as what the user sees. The down time may be due to hardware, the operating system, the program, the computer operator, or the user himself. Occasionally, down time is generated by the transition time necessary to change allocations of memory or scopes.

SUMMARY

The Lockheed-California Company has developed a strong and effective computer graphics capability. It has grown from an idea based on new equipment announcements in 1964 to many scopes used on multiple shifts in 1973. Adherence to sound considerations relating to both the programming and the user were important factors in this successful development. Computer Graphics need not be restricted to a research environment, but can contribute profitably to all phases of work. At Lockheed, it is a growing computer-aided design tool with contributions ranging from design concept to manufactured product. Its potential for growth to meet future needs is unlimited.

Graphics and digitizing automatic transduction of drawings into data bases

by CHARLES M. WILLIAMS

Virginia Polytechnic Institute and State University
Blacksburg, Virginia

INTRODUCTION

Figure 1 is a computer graphic rendition of a sketch of Einstein by Hans Erni.[1] The original hard-copy source document was digitized in less than 1 minute by a Visicon AD-1 system.[2] Approximately 20 seconds of IBM 370/165 digital computer time were then required to organize the resultant data into list structures and generate plot commands. The plot itself required roughly 7 minutes of Calcomp 718 digital plotter time. The entire process was simple, fast, accurate, and inexpensive. It is a sample of the power of an automatic digitizing system operating in a computer graphics environment.

The automatic digitizing system used contains a drum type raster scanner which yields the digital coordinates of the black points on hard copy drawings. These coordinates are then processed by a graphic collation software package to produce a list structure representation of the black areas on the drawing. These list structures can then be processed to yield such plots as shown in Figure 1.

DIGITIZATION PROBLEMS

Historically the development of computer graphics has been severely constrained by an inability to transduce hard copy drawings into computer data bases in an economical and rapid manner. Although excellent methods have been developed for displaying and recovering permanent copies of drawings from such data bases, the input mechanisms have been largely constrained to tedious and error prone line tracing methods inadequate for complex work. In essence, computer graphics has operated as a television industry with no television cameras. The development of the automatic digitizer has now provided the camera, the graphic collation software, the signal converter between the camera and the television set.

The magnitude of the task of transducing even a simple drawing into a computer data base may be glimpsed by examining statistics taken from Figure 1. This drawing of Einstein required 531 distinct pen strokes which were constructed from 13,475 straight line segments. The digital representation of such data more than amply justifies the adage that a picture is worth a thousand (computer) words.

The visual information in this drawing is contained in the nuances of the shape and thickness of the individual lines that compose it as well as the spatial inter-relationships of those lines to each other. Each line carries its own important message, and the impact of the drawing is reproduced only by faithfully copying all of its features by a camera-like process.

Line tracing methods involving such equipment as manual digitizers, automatic line followers, or graphic data tablets are inadequate as they cannot capture the drawing in its original form. Aside from the fact that they do not standardly record variations in line thickness, such devices suffer from positional accuracy problems generated by the requirement that they track individual lines separately. On the one hand there is a high correlation in the positional accuracy of the points which determine an individual line so that the detection and removal of errors by smoothing methods are ineffective. On the other hand, as there is no correlation in the positional accuracy of the points on separate lines, the inaccuracy of separate lines relative to each other can be twice that of the individual points themselves. As a result, traced lines can fail to intersect or intersect when they should not, and there is no contextual information provided which can identify and revolve the problem automatically. The resulting distortions can create large deviations in both shape and topology of the drawing which are exceedingly difficult and expensive to detect and remove.[3]

These problems, however, are minor when compared to those a human can interject into the system when manual tracing methods are used. Errors of commission and omission become rampant when a human is required to concentrate both visual and motor skills in the laborious and tedious act of line tracing. His muscular coordination and mental attention will deteriorate rapidly if tracing is required over an extensive period of time. Although the equipment he uses is often highly accurate itself, the human is not, and worse yet his errors are not sufficiently repeatable for a verification process to be meaningful.

Figure 1—Digital replot of an automatically digitized drawing

AUTOMATIC DIGITIZATION

Much of the power of an automatic digitizer is embodied in its ability to capture all the visual aspects of a drawing to an accuracy exactly specifiable by the mechanical and optical properties of the device itself. Its performance must be predictably repeatable within these known limits. Figure 1 for example was digitized at 200 samples per inch, and each digitized point was repeatable to an accuracy of 0.005 inches.

The power of the graphic collation process, on the other hand, is derived from its ability to transduce this data into graphic data structures in a rapid and efficient manner with no attendant loss of information. The graphic collator software performs this function in times essentially proportional to the length of lines on the drawing and independent of their complexity. The derived data structures are organized by curves in such a manner that distinct lines on the drawing are represented by distinct list structures.

As these data structures include both line width and line intersection information, they constitute both a pictorial and a topological representation of the source data to an accuracy identical to that of the digitizing process. Moreover, their manipulation and analysis can be performed by standard list processing techniques.

Figure 1 for example was obtained by plotting lines around the peripheries of the dark areas of the drawing. The points defining these lines were smoothed by least squares techniques to remove the 0.005 inch digitizer quantizing noise. Alternatively line thinning techniques could have been employed, or the data could have been fed directly to electrostatic (raster) plotting devices. As line connectivity information is also included, network analyses of the drawing can be obtained directly.[4,5]

PROBLEMS OF AUTOMATIC DIGITIZATION

Unfortunately, automatic digitization does not yet imply that all desirable input functions will automatically be performed. First, the labeling, or identification, of graphic elements must still be done by hand. On-line methods can use standard interactive techniques employing such tools as data tablets, light pens, and joy sticks. Off-line techniques can profitably employ manual digitizers for locating and identifying important points on the source document. The computer correlation of these points with those derived through automatic digitization can be efficiently accomplished during graphic collation provided reference lines are included for orienting the document. The accuracy of the manual digitizer can now play a decisive role in resolving the identity of tightly spaced lines. Moreover, the human error factor has been greatly reduced as the muscular coordination required in line tracing is not involved.

The second problem involves the identification and removal of unwanted information which is included in the data because the entire drawing has been transduced. Such information is readily and economically identifiable provided it has unique localized characteristics. Noise, for example, is easily handled when it occurs in the form of isolated dots or short, thin line segments. Lettering can also constitute unwanted information if there is a requirement to replace it by coded text. Such lettering can be isolated if it can be uniquely described by its localized high spatial frequency characteristics.[6] The recognition of individual letters is not necessary if the coded text is inserted and correlated by other means.

Automatic editing will in some instances not be totally successful, and human auditing of the results may be required. The effectiveness of such interaction on-line will be considerably enhanced if the data structures describing deleted material are available. The power of the system will still be in evidence even if considerable on-line manipulation is necessary. The suppositions are that the bulk of the task has been accomplished correctly, and that the interactive erasure of unwanted information on a line-by-line basis is a far easier and more accurate task than the creation of new information by sketching.

SUMMARY

This process is a realization of an economical, accurate, and easy method for capturing drawings for use in com-

puter graphics systems. The implication is a far more productive utilization of computer graphics console time with emphasis on image manipulation, editing, and analysis rather than on image creation.

The value of automatic digitization lies in its ability to rapidly and accurately transduce drawings into computer data bases. Such ability implies that drawings may directly act as the original source of information for the various purposes for which they were conceived. That is, the data base will be in one-to-one correspondence with its own drawing and may therefore hopefully provide the same information to the computer that the drawing does to the human.

The dream of computer graphics is of man and machine working intimately toward the solution of complex problems by combining the visual power of the human with the computational power of the computer. This dream is indeed realizable provided that the data upon which it must feed is readily accessible, for the strength and viability of any computer system is measured by the data which it processes.

REFERENCES

1. Erni, H., *Study of Albert Einstein*, Meggen, Lucerne, Switzerland.
2. *Visicon AD-1 Automatic Digitizing System*, Visicon Inc., State College, Pennsylvania.
3. Burch, G., *Personnel Communication*, Graphic Data Entry, Inc., Washington, D.C.
4. Williams, C. M., "An Algorithm for Rapidly Parsing Automatically Digitized Drawings", *Proceedings of the 1972 Army Numerical Analysis Conference*, Biomedical Laboratory, Edgewood Arsenal, Maryland, April 1972.
5. Novoshielski, J., *An Interactive Program for Automated Network Description*, Unpublished Master's paper, Computer Science Department, The Pennsylvania State University, Pennsylvania, March 1972.
6. Williams, C. M., *Automatic Graphic Data Entry*, A paper to be presented at the Annual Meeting of the American Congress on Surveying and Mapping and the American Society of Photogrammetry, Washington, D.C., March 1973.

Graphics in medicine and biology

by CAROL M. NEWTON

University of California
Los Angeles, California

INTRODUCTION

Interactive graphics is not new to biology. The need felt for intuitively guided, graphically supported model exploration early predisposed physiological modelers to analog systems. Small computers derived from the LINC, which was designed for biomedical laboratory research, have long found favor among biomedical investigators both from experiment control and off-line interactive analyses with graphical displays. The decision to transfer from these "hands-on" approaches to larger digital systems often was made with considerable reluctance, despite the latter's greater capacity, versatility, precision, and rich libraries of statistical and other applications software. Modern interactive graphics terminals time-shared from major digital systems not only obviate the necessity to choose between these types of capability, they enable more versatile and meaningful forms of interaction than heretofore realized.

Consider some basic characteristics of biology. Complexity is encountered from the molecular level through the physiological or population systems one seeks to model realistically. Hypothesis discovery is as important an activity as hypothesis validation in so rapidly developing a field, and hence there is great need to support human intuition. The pictorial and other data bases in which one seeks to discern patterns entail ill-defined variances and overlaps that often yield better or more efficiently to visual inspection than to computational algorithms. Much of the computer-related progress in biology and medicine today requires collaboration of people with different backgrounds; at the graphics console, mathematicians and biologists can exercise and cross-check the concepts they are seeking to communicate to each other. The interface problem is of controlling importance when one wishes physicians or other dedicated health-care professionals to use computers. Light-pen selection, conversational displays that include access to instructions, and meaningful graphical feedbacks have won acceptance from these professionals.

The relationship of these considerations to indications for graphics support will be illustrated in a number of biomedical applications areas. Finally, some NIH-supported developments in graphics technology will be mentioned.

SOME IMPORTANT CATEGORIES OF GRAPHICS APPLICATIONS IN BIOLOGY AND MEDICINE

3-dimensional structures

Many people became aware of graphics' potential in biomedical research through Levinthal's displays of rotating molecules.[1] Molecules encountered in biological systems tend to be complex. One wishes to view these molecular structures from different directions, with an ability to highlight various substructures, alter substituents, explore allowable structural changes, and discern geometrical relationships such as planar or helical surfaces rich in atoms, which may not be easily suggested by a molecule's structural formulas. The limitations of conventional ball-and-stick molecular models are apparent. It is not surprising that a number of biochemical investigators have developed and used interactive graphics molecular modeling systems. Levinthal and others now are exploring similar approaches to depict neural systems. Even an embryonic nervous system can be so complex that the unaided mental reconstruction of 3-dimensional relationships from serial sections is prohibitively difficult.

Image processing

The 2-dimensional images encountered in biology also tend to be complex. Early hopes that computers soon would fully automate interpretations of chest x-rays, chromosome spreads, and microscopic preparations have given way to a healthy respect for the human eye. Neurath's[2] and Frey's[3] interactive approaches to chromosome analysis delegate to the human and to the computer what each does best. By light-pen, one can easily isolate individual chromosomes in graphics scope displays of overlapping spreads. Thereafter, the computer performs the required measurements and classification analyses. It is of considerable diagnostic interest to be able to estimate heart volume as a function of time. Biplanar cineangiograms provide a stereoscopic view of the beating heart by recording a rapid sequence of pictures taken while a contrast material is present in the heart. The processing of this large amount of data can be facilitated greatly if a

human operator can intervene with light-pen whenever an otherwise automatic system for tracking heart boundaries derails.[4] A broad program in interactive image processing is being jointly undertaken by engineers and radiologists[5] at the University of Missouri.

Examination of data

Both physiological signals and the data bases encountered in clinical studies are likely to be complex and noisy. The need to infuse human intuition and pattern perception into the analysis of these data has motivated a variety of interactive graphics developmental efforts at UCLA's Health Sciences Computing Facility[6,7] and elsewhere. For the neurosciences, shifting details that may be obscured in large-sample averaging of evoked responses are sought by visual inspection of many smaller subsample averages. Different functional forms and initial parameters for non-linear regression fits to interspike interval histograms can be explored at the graphics console, and sorts on various fitted parameters for large sets of research data provide ordered listings of descriptive data concerning each experiment, which can be inspected as a first step toward recognition of trends or clusters. Interactive graphics programs for file manipulation, regression, time-series analysis, and discriminant function analysis are among the statistical utilities being developed.

It is especially helpful to present data in association with the structures to which they relate, often not tabular in form when biological. In Sheu's neuropharmacological retrieval system,[8] a light-pen guides exploration of brain-section diagrams for domains where injection of a given pharmacological agent most frequently results in a specified Boolean combination of physiological and behavioral reactions. Interaction with displays of data superimposed on a map of Los Angeles County facilitates the study of epidemiological variables in conjunction with census-tract data and other sources of socioeconomic information,[9]

Data handling in pharmacology presents special problems, among them the storage and retrieval of molecular structural information with subsequent exploration of its correlation with complex data bases derived from animal research or clinical trials. It therefore is not surprising that the PROPHET system being developed under NIH's Chemical/Biological Information-Handling Program to support pharmacological researchers[10] has been designed for time-shared, remote graphics terminals.

Designing of special-purpose medical equipment

Interactive graphics systems frequently are used as design tools in engineering. Their use is especially appropriate when inexpensive bedside equipment or economical computer algorithms are being developed for the monitoring of physiological systems under realistic conditions where patient movements or environmental factors introduce signal artifacts. For example, Saltzberg and his associates[11] compare inexpensive algorithms for electrocardiographic monitoring, applying them to digitized recordings from a large number of patients who have been studied in intensive-care wards. The calculated statistics are displayed under a plot of each heart beat, along with markers at defined locations on each beat as determined by the computer. The program pauses and an alarm is indicated whenever one of the statistics being tested falls outside of allowable bounds. The concurrent display of statistics, markers, and ECG tracings facilitates insight into the reasons for algorithm failures and advantages. Hon[12] is using a similar approach to design bedside equipment for monitoring during labor.

Guidance or implementation of therapy

Computers have been used for many years in radiation treatment planning.[13] However, clinical acceptance rose significantly when the special-purpose interactive graphics Programmed Console was introduced.[14] More recently, remote interactive graphics terminals that access major computers over voice-grade lines enable somewhat more sophisticated computations and displays of dose distributions and organ outlines to be presented to the therapist via a convenient conversational interface[15] that health professionals can use with very little instruction.

An audio-graphics system is itself the therapist[16] for non-speaking autistic children. It is known that these children tend to love machinery and shy away from speaking adults. The system responds to the child's interventions at the keyboard with graphics displays that are correlated with spoken words or other sounds. Initial results have been very encouraging; a high percentage of the children using this system have developed sufficient motivation to acquire speech, to enable productive resumption of conventional forms of therapy.

Modeling

Realistic mathematical description and productive investigation of the difficult models encountered in biology and medicine demand the highest levels possible in both analytical skills and biomedical knowledge. A very small number of people exist or now are receiving training that might enable them eventually to supply both types of professional support in a given problem area. For some time there will be a need to bring together people of very diverse backgrounds to attempt the truly difficult problems. In addition, it often is necessary to involve experts from different biological disciplines in problems of great practical concern, such as treatment of cancer, where one seeks to expedite transport of recent experimental findings throughout biology into the clinical domain. At a graphics terminal, problem specifications and results can be presented in forms that are familiar to all of the scientists involved. When requested, a more precise descrip-

tion of underlying mathematical structures and the biological assumptions upon which they are based can be made available. On the other hand, when the biologist is shown a graph of a proposed dose-response function, presence of a leading shoulder supports his confidence that the mathematician's formula may in fact correctly represent a reparable form of cellular damage.

Characteristically, modeling problems in biology and medicine are complex. A large amount of information may be required to define the model to be explored or computation to be performed. Specifications in one area often risk conflict with those in another. Interactive expedition of problem specification can help both to reduce the incidence of conflicts and to alleviate a distraction from more demanding intellectual tasks required of the user.

Biologically realistic models having few parameters or closed solutions are not common. Model exploration therefore tends to require the running of numerous computational probes representing a variety of model conditions. Total coverage of all conditions seldom is required or economically feasible. Since outcomes often are difficult to predict in such models, strategies for efficiently terminating individual runs or varying conditions for subsequent runs may have to be developed as model exploration proceeds. Thus, the most effective possible graphics feedbacks to investigators guiding model exploration are both a scientific and an economic necessity.

Better understanding of biological phenomena has been and probably remains the primary motivation for modeling. However, there is an increasing awareness of its promising roles in facilitating the infusion of basic research into medicine, and in education. Interactive graphics is particularly supportive to these roles.

Models developed to expedite transferrence of biological research findings to improved strategies for treating or diagnosing disease must be detailed and flexible enough to accept new findings while preserving a usable, intelligible interface to the clinicians and medical scientists involved in their exploration. Ideally, the biological portion of the model should be embedded in a system which facilitates both its perturbation by the various therapeutic approaches being considered and its incorporation of labeling or other processes that support simulation of laboratory assessments of the treated system's status. When such a level of complexity and flexibility is required, it becomes difficult to envision other than interactive graphics approaches. Lincoln's[17] model for guiding leukemia therapy simulates the application of different schedules of cytotoxic treatment agents to the various types of cells that comprise the blood-forming system. Program construction was facilitated by RAND's BIOMOD system for interactive graphics modeling.[18] Another interactive graphics model for exploring cancer therapy assesses the comparative cycle-specific damage inflicted upon two cellular systems by a variety of combined radiation and chemotherapeutic treatment strategies.[19] Mittman's[20] lung model has been designed to aid the development of better pulmonary function tests and of a better understanding of how conventional uptake-washout tests should be interpreted when complicating factors, such as marked inhomogeneity of compliances, are considered.

Any of the above models can be used to aid education. Other interactive graphics instructional programs[6] include illustrations of basic concepts in sampling and other supports to the medical curriculum in biostatistics.

NIH-SUPPORTED DEVELOPMENTS IN GRAPHICS TECHNOLOGY

Many of the early NIH-supported projects in computing involved some developmental work in analog or hybrid computation, or in the designing of convenient laboratory computers, which entailed an interactive graphics component. Some more recent projects have focused directly on graphics: Neilsen's[21] low-cost, refreshable interactive graphics terminal was developed with NIH support at a time when commercial efforts to achieve an order-of-magnitude slash in the cost of interactive graphics were not evident. A project now is being supported to field-test this terminal as well as an inexpensive commercial system that subsequently became available. As mentioned earlier, the PROPHET project is developing systems support for time-shared remote graphics terminals.[10] The development of languages to support compiler-level access to interactive graphics systems has been a longstanding activity at UCLA's Health Sciences Computing Facility.[6] This work recently has been extended to cope with the low-baud and noise problems entailed in supporting remote graphics terminals over conventional voice-grade lines.[15,22] Although Clark's macro-module approach to constructing computer systems[23] was developed on the basis of a number of considerations relevant to needs in biomedical computing, one notes that one of its first major applications has been to graphics-supported molecular modeling.

CONCLUSION

The foregoing descriptions of interactive graphics activities in biology and medicine are intended to be illustrative rather than complete. These activities are widely based and unmistakably increasing. The author apologizes both to readers and to other investigators for omission of some very excellent projects that would be included in a more comprehensive review.

A deeper presentation of several projects at the conference will be illustrated by motion pictures.

REFERENCES

1. Levinthal, C., "Molecular Model-Building by Computer," *Scientific American*, Vol. 214:42, 1966.
2. Neurath, P. W., Bablouzian, B. L., Warms, T. H., Serbagi, R. C., Falek, A., "Human Chromosome Analysis by Computer—An Opti-

cal Pattern Recognition Problem," *Ann. N. Y. Acad. Sci.*, 128, pp. 1013-1028, 1966.
3. Frey, H. S., "An Interactive Computer Program for Chromosome Analysis,"*Computers and Biomedical Research*, Vol. 2, pp. 274-290, 1969.
4. Desilets, D. T., Beckenbach, E. S., "Myocardial Function from Cineangiocardiograms with a Digital Computer," *Radiology*, 99, pp. 319-325, May 1971.
5. Hall, D. L., Lodwick, G. S., Kruger, R., Dwyer, S. J., "Computer Diagnosis of Heart Disease," *Radiological Clinics of North America*, 9, pp. 533-541, 1971.
6. Dixon, W. J., *Annual Reports*, 1967-68, 1968-69, 1969-70, 1970-71, 1971-72, Health Sciences Computing Facility, University of California, Los Angeles.
7. Britt, P. M., Dixon, W. J., Jennrich, R. I., "Time-Sharing and Interactive Statistics," *Bull. ISI*, 191, 1969.
8. Sheu, Y., George, R., Jenden, D. J., Newton, C. M., "Topographic Information Retrieval in Neuropharmacology by Using Graphic Display," *Proc. Assoc. Comput. Mach.*, Vol. 24, pp. 485-498, 1969.
9. Eskind, N., "Large File Storage and Retrieval with Emphasis on the 1970 Census," *Eighth Annual Symposium on Biomathematics and Computer Science in the Life Sciences*, Houston, 1970.
10. Raub, W. F., "Automated Information-Handling in Pharmacology Research," *Proc. Spring Joint Computer Conference*, pp. 1157-1165, 1972.
11. Saltzberg, S. L., Haywood, J., Harvey, G., Vereeke, D., "Electrocardiographic Monitoring and Wave Form Analysis Utilizing Interactive Computer Graphics," *Journal of the Assoc. for the Advancement of Med. Instrumentation,* No. 5, pp. 91-96, 1971.
12. Dixon, W. J., *Annual Reports*, 1969-70, pp. 166-7, 1970-71, pp. 257, Health Sciences Computing Facility, University of California, Los Angeles.
13. Stovall, M., "Bibliography of Computers in Radiation Therapy," *Brit. J. Radiol.*, Special Report No. 5 (Proc. Third Internat. Conf. on Computers in Radiotherapy, Glasgow, 1970) p. 125, 1972.
14. Holmes, W. F., "External Beam Treatment-Planning with the Programmed Console," *Radiology*, 94, pp. 391-400, 1970.
15. Ryden, K. H., Newton, C. M., "Graphics Software for Remote Terminals and their use in Radiation Treatment Planning," *Proc. Spring Joint Computer Conference*, Vol. 40, pp. 1145-1156, 1972. (Note: Sternick is doing similar work at Dartmouth.)
16. Smith, D. C., Newey, M. C., Colby, K. M., "Automated Therapy for Non-Speaking Autistic Children," *Proc. Spring Joint Computer Conference*, Vol. 40, pp. 1101-1106, 1972.
17. Lincoln, T. L., "The Clinical Significance of Simulation and Modeling in Leukemia Chemotherapy," *Proc. Spring Joint Computer Conference*, Vol. 40, pp. 1139-1143, 1972.
18. Groner, G. F., Clark, R. L., Berman, R. A., Deland, E. C., *BIOMOD—An Interactive Computer Graphics System for Modeling*, The Rand Corporation, R-617-NIH, July 1971.
19. Newton, C. M., "Planning Radiotherapeutic Strategy," *Proc. San Diego Biomedical Symposium*, 11, pp. 189-203, 1972.
20. Mittman, C., Raum, D., in W. J. Dixon, *Annual Report* 1968-69, p. 164, Health Sciences Computing Facility, University of California, Los Angeles.
21. Neilsen, Ivan R., "On-Line Graphics Display Techniques Enhance Clinical Utility of Computers in Health Care Systems," *Eurocon '71 Proceedings*, Lausanne, Switzerland, October, 1971.
22. Newton, C. M., Ryden, K. H., "Remote Interactive Graphics and Cancer," *Proc. First USA/Japan Computer Conference*, Tokyo, October 3-5, 1972, pp. 277-286.
23. Clark, W. A., Molnar, C. E., "Macromodular Computer Systems" in *Biomedical Research*, Vol. IV, Ed. by R. W. Stacy and B. Waxman, Academic Press (in press).

The topological design of sculptural and architectural systems

by RONALD D. RESCH

University of Utah
Salt Lake City, Utah

INTRODUCTION

In my work over the past 12 years I have consistently created systems with an approach which I will describe as topological design. I have tried to capture the spirit of this work in my film, "The Ron Resch Paper and Stick Thing Film," which will be shown elsewhere in the conference. I would like to present here, and in a color slide presentation, the concept of topological design using examples from my work for clarity. I will show how the conception and fabrication of designed objects have a topological and a geometric aspect. Currently, these objects may be characterized as emphasizing the geometric aspect, by limiting an entire class of objects to be a unique one. Finally, I will show where the introduction of the computer into design and production process makes possible the topological aspect, so that the automation of custom made objects might replace a series of identical 'ready mades.'

HISTORICAL BACKGROUND

Before the Industrial Revolution, one imagines that manufactured objects were made by that commonly named group of people, the "Smiths." The blacksmith, the silversmith, the locksmith, and the gunsmith were the craftsmen that made the objects of daily living. The smithery, and the objects it made, emerged from, and were maintained by, an immediate collection of patrons whose wants and needs it satisfied. The legacy handed down by tradition was not just the object itself, nor was it the object and its smithery as caretaker. It was a symbiosis of user, object, and maker that lived together in very close communication, both in time and in space.

The gunsmith, for example, possessed a "soft prototype" in his intimate and complete understanding of a gun. That his concept was topological is clear, from his ability to accommodate the varying needs of his users by varying the design of a gun to fit the user. The grip could be made a little heftier, the barrel a little longer.

Industrialization came with the desire for greater volume. The inherent understandings of one man, in touch with his craft and his patron, gave way to an explosive fragmentation of this intimate process. The fragmentation of implicit functions into explicit roles to be played by many persons and machines, brought with it an incredible communications problem.

In response to this, there came standardization and regimentation. The varying needs of the user were averaged by the researcher, the designer, and the engineer, and frozen into a "hard prototype." The responsive variations of the craftsman were replaced by a cumbersome and complex organization that seems to be a patchwork of stop-gap measures, with a commenserate growth of communications problems.

The introduction of the computer into this complex set of relationships, as the greatest and fastest of machines, may only lengthen and slow down the communications process. Its introduction as a medium of communication, however, could bring together the maker, the object, and the user into a cohesive whole once more. My work in structure design has produced the following concepts and structures which make this seem plausible.

PERSONAL BACKGROUND

In 1961 I began examining wadded sheets of paper and trying to understand the random folding that occurred. My desire was to understand what sculptural forms were possible. The study limited the possibilities from the outset by two highly controlling restrictions:

1. Only folding of the flat sheet was allowed, i.e., no cutting or gluing.
2. The folded edges were forced to be straight line segments; no curves.

With these restrictions as goggles and blinders, I looked at a wadded sheet for randomly occurring wrinkles of interest. I selected several patterns, diagrammed them on a separate sheet in accordance with the imposed limitations, and folded them. Each of these forms had visual appeal, but was seemingly unrelated. The search was for something that had both conceptual and visual appeal. In an attempt to achieve this I was influenced by the classic

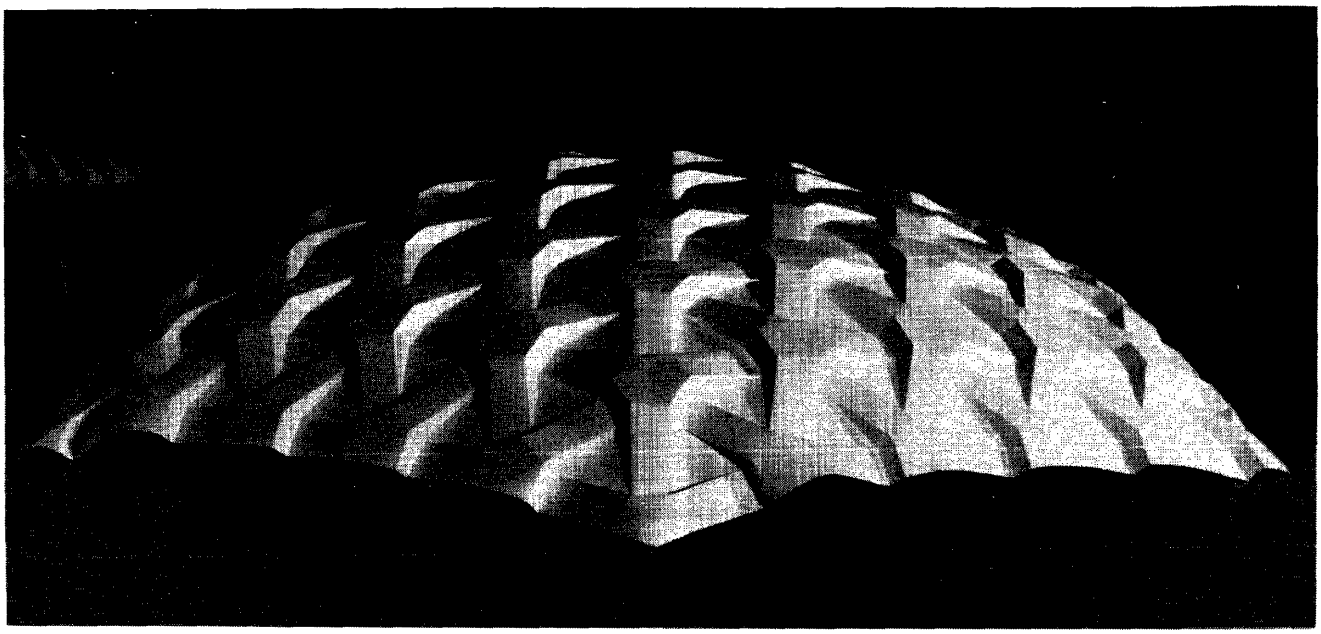


Figure 1

notion that a form should be consistent with the material from which it is made.

Consequently, I decided to impose a third restriction in the form of a question. It was the key to a rich set of possibilities. Could these simple forms be replicated through symmetry operations into a single folded surface which would be an expanse of delicate and intricate folds? Could I rearrange the folding diagrams of these elementary forms such that when combined with symmetry operations they would create two-dimensional mosaic patterns? The folded paper dome in Figure 1 was the first structure resulting from these restrictions.

I was pleased with this first folded paper form, even though I understood it to be only one of a class of forms. This led me to ask, how many ways could I change the scoring pattern? How would variation in a particular scoring diagram affect the three-dimensional folded structure? What were the limits within each class of change? Before introducing the structures which resulted from these questions I will describe a classification scheme for man-made objects. While the scheme is broadly applicable, I will focus on architectural structures including my own.

GENERATING SYSTEMS

The geometric surfaces employed in architectural design are typically the plane and the sphere. When one wished to build a network approximation to these ideal surfaces, or more complicated ones, a generating system is required. A generating system is defined with the existence of two associated rules which describe:

1. How to determine the geometry of each part
2. How each part is assembled into the system

Following are three classifications for generating systems: combinatorial, geometric and topological.

COMBINATORIAL GENERATING SYSTEMS

A combinatorial generating system has a part rule which determines a finite set of elements, and which allows for each to be replicated. Its rule for connectivity is not definitive, but is permissive within contextual constraints by allowing the user to combine the parts in various ways to form a single system. The geometry of the part, or syntactic rules, may dictate these constraints of part connectivity. Some examples of this would be the letters of the alphabet, modular architectural systems and certain constructional toys, such as Tinker Toys.

GEOMETRIC GENERATING SYSTEMS

As most structural systems in architecture are composed of parts arranged in triangulation, it is more revealing to look at the rule for determining the geometry of the part rather than its connectivity rule. Two types of architectural structure which make this consideration clearly relevant are flat space frames and domes.

The shape of most architectural roof systems is flat, whether the system be horizontal or inclined. Space frames are often used where a large free-span, (uninterrupted by supports), is required. Of the many space frame geometries used, the "octet-truss" (composed of octahedrons and tetrahedrons), is the most common. It has the simplest part rule: cut all the pieces the same. It is not surprising that the simplest structure has the simplest rule. This simplicity contributes to its popularity. Such a flat structure was assembled into the well-known

pattern at the bottom of Figure 8. One can note that the identical pieces in this structure are equilateral triangles, the structure being more commonly fabricated from rods of equal length.

The part definition for domes, however, requires a more complicated rule resulting from a geometric impossibility. It is impossible to transform a regular network of triangles, squares, or hexagons onto a domed surface without distortion. The regular network is usually transformed to fit a dome by a relative expansion of the interior and relative contraction at the outer edges. What distinguishes various dome designs is the method used to distribute the network on the dome surface, and the surface shape permitted.

Of the several generating systems for the geometry of a dome, the geodesic system has become the most publicized. It combines the geometry a regular polyhedra, the icosahedron, with the transformation of a triangular network onto a sphere.

The transformation rule patented by Fuller* determines that consecutive nodes of the network will lie along great circle arcs of the sphere. Thus, it borrows its name and method from classical geometry where a geodetic line on a sphere is defined to be a great circle arc. This means, of course, that the shape of geodesic domes is limited by its defining rule to be sections of a sphere. Also, Fuller's geodesic system has somehow achieved a popular misconception, that the triangles composing its surface are all equilateral. The fact is, there is a 50-50 chance (depending on whether the subdivision of the icosahedral face is even or odd) that there is not a single equilateral triangle in the surface of the entire dome!

The octet-truss and the geodesic dome characterize nearly all current "generating systems." They each produce only one shape of structure. The former produces a flat structure and the latter produces a spherical structure. These shape-dependent systems I refer to as "geometric generating system," since they produce a fixed system shape with associated fixed parts, and geometry is the study which concerns itself with determining size and shape.

The distinction between geometric and topological generating systems is in the rule for part definition; the rule for connections being held constant. A generating system is topological if the rule for part definition is a continuous mapping of the geometry of some or all parts. It is geometric if the rule for part definition produces a unique part geometry for each of the parts. The discriminating question here is: Does the system define a continuous transformation which varies the geometry of some parts while maintaining connectivity? If the answer is 'yes,' the system is topological; if the answer is 'no,' the system is geometric.

TOPOLOGICAL GENERATING SYSTEMS

As noted above, architectural generating systems have two rules: (1) a rule defining the part and, (2) a rule defining its connectivity to other parts. The first rule is geometric in nature and the second rule, as here defined, I would characterize as topological.

There is no need for a duality in viewing a designed object as a system of parts, but it is useful to focus attention on an object's quantitative aspects separate from its qualitative aspects. In the field of geometry, topology has been defined as the geometrical theory of situation without respect to size or shape. The popular definition is rubber sheet geometry. I will use topology to describe the continuous transformation of a system that may preserve only the connectivity of its parts. A topological generating system is a generating system whose rule of connectivity and whose set of parts remain constant, while a continuous transformation varies the geometry of its parts, or some subset thereof.

TWO LEVELS OF CONTINUOUS TRANSFORMATION FOR FOLDED STRUCTURES

My concern in the design of structures has been with the continuous transformation of the parts of a system. This concern is the basis for my concept of topological design. Some structures designed with this concept reveal that there is a two-level hierarchy of differentiable spaces; one traced out by a topological transformation, the other by a geometric one.

First, there is the two-dimensional space of the mosaic in which the scoring pattern may be varied topologically. A continuous transformation of this space maintains connectivity while changing the geometry, i.e., the size and shape of the system parts, the simplest of these transformations is the "ribbon transformation," which increases or decreases the width of the "ribbon" while not altering the size or shape of the remainder of the mosaic. The effect of this transformation can be seen in Figure 2 where three intersection "ribbons" of the mosaic are being decreased to a zero width.

Secondly, there is the three-dimensional space in which a folded structure may be varied by folding. The triangles resulting from the topological transformation are connected to each other in such a way that each edge is an axis of folding. Thus the folding of the individual triangles relative to each other makes possible a rigid body motion of the triangular parts, while also determining a continuous, geometric transformation of the entire system of parts. The shape of the total system may become rigid by fixing the angular relations between the parts at the boundry of the system alone, or by fixing these angular relations throughout the entire system.

Therefore, the selection of a distinct folded pattern from the topological space of a foldable mosaic will determine an entire class of structures, the members of which are a rigid instance of the dynamic folding of its specific mosaic pattern. This can be seen by comparing the top module of Figure 2 with the folded form created from it, shown in Figure 1.

For further illustration, pick the bottom module from the topological space of Figure 2. Repeat this pattern in

* R. Buckminster Fuller, Patent #2,682,235, June 29, 1954

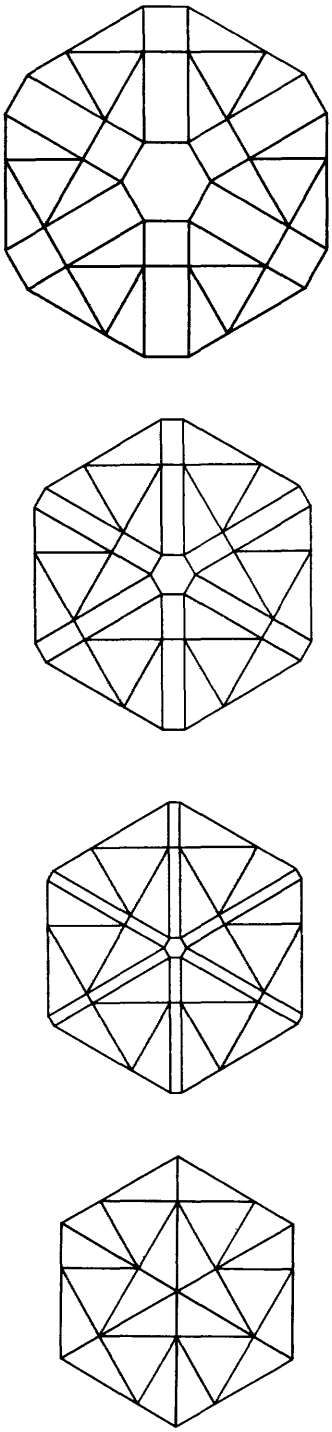


Figure 2

the plane; and fold. One instance of the class of realizable structures from this pattern is Figure 3.

Christiansen has written a computer program to simulate the folding transformation.* It was used to compute the geometry in the sequence of computer simulated video pictures of Figure 4. It is evident from this sequence that a single scoring pattern may be continuously transformed by folding, to achieve a wide variety of architectural surfaces.

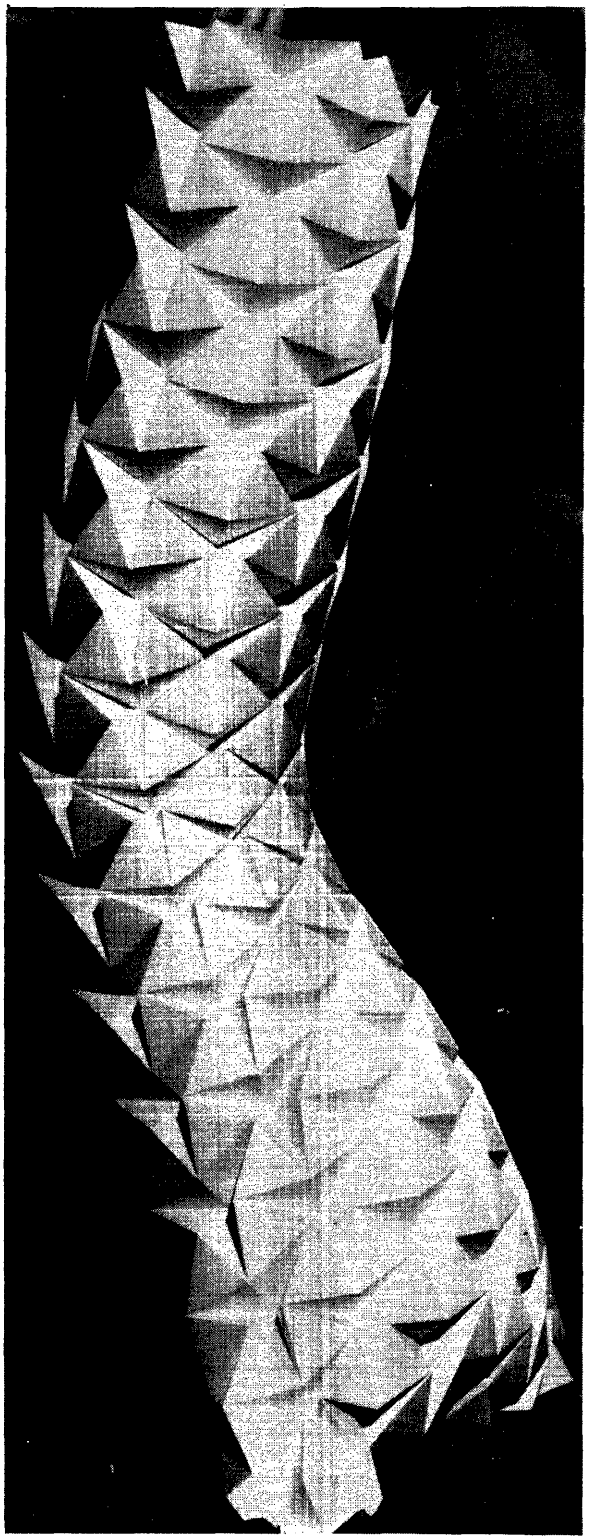


Figure 3

* Reported in R. D. Resch and H. N. Christiansen, "The Design and Analysis of Kinematic Folded Plate Systems." Proceedings of the Symposium for Folded Plates and Prismatic Structures, International Association for Shell Structures, Vienna, Austria, October, 1970.

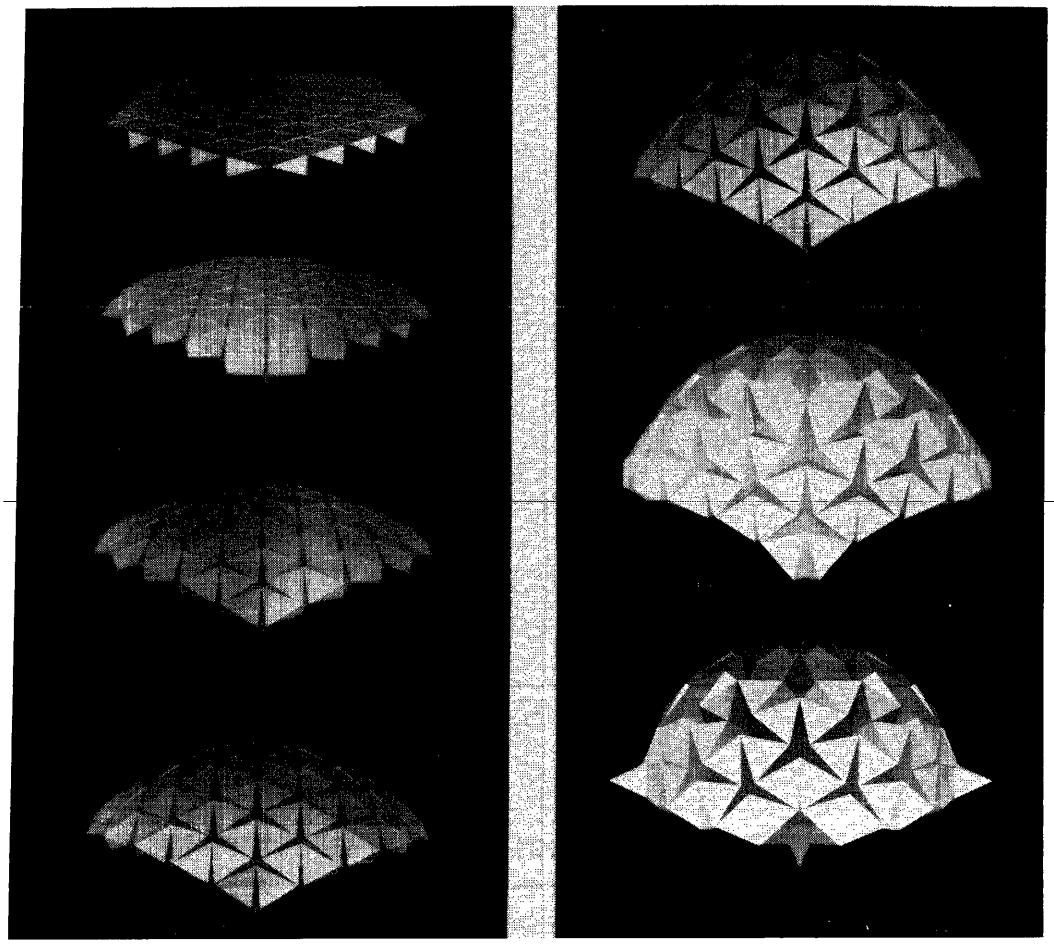


Figure 4

I have discussed the two levels of transformation (the topological and the geometric) which may be performed upon a "folding mosaic" of the two-dimensional plane. The combined transformations produce a continuum of physically realizable structures, from which a designer may select a specific architectural shell form, by specifying his needs.

FOLDING MOSAICS AND MODULAR FOLDINGS

The two levels previously described are imbedded within two more. While the first two levels are sets whose members exist in a continuous space, the members of the remaining two levels exist in a discrete space. They are:

1. The set of all folding mosaics of the plane*
2. The set of all modular foldings of the plane

Above, I have shown one of my "folding mosaics" and allowable transformations of it. Many others may be seen in my film previously mentioned. This set may be included within a set of modular foldings of the plane. It will include modular foldings which are not infinite, two-dimensional patterns, but are limited by their definition to some local, finite graph whose modules are interrelated by a rule. I will describe one example of this set which was published as the cover design of the "Communications of the ACM" for November, 1970.

This structure is named "bird form," as I have used it most often to create abstract, bird-like shapes. As in the folded mosaics, the scoring diagram for the bird-form has its associated topological transformations for which I have written a computer program with parameterized input. Each parameter controls a class of continuous topological change to the scoring diagram.

Some of the classes of change are depicted in Figure 5. From top to bottom they are:

1. Proportions of the "page" before subsequent transformations
2. Variation of vertex location on inner and outer broken line segments
3. Scaling of outer broken line segments
4. Angular variation of outer broken line segments

A specific geometric scoring diagram of the bird form, resulting from these topological transformations, defines a class of possible three-dimensional structures. For

* R. D. Resch Patent #3,407,558 October 29, 1968.

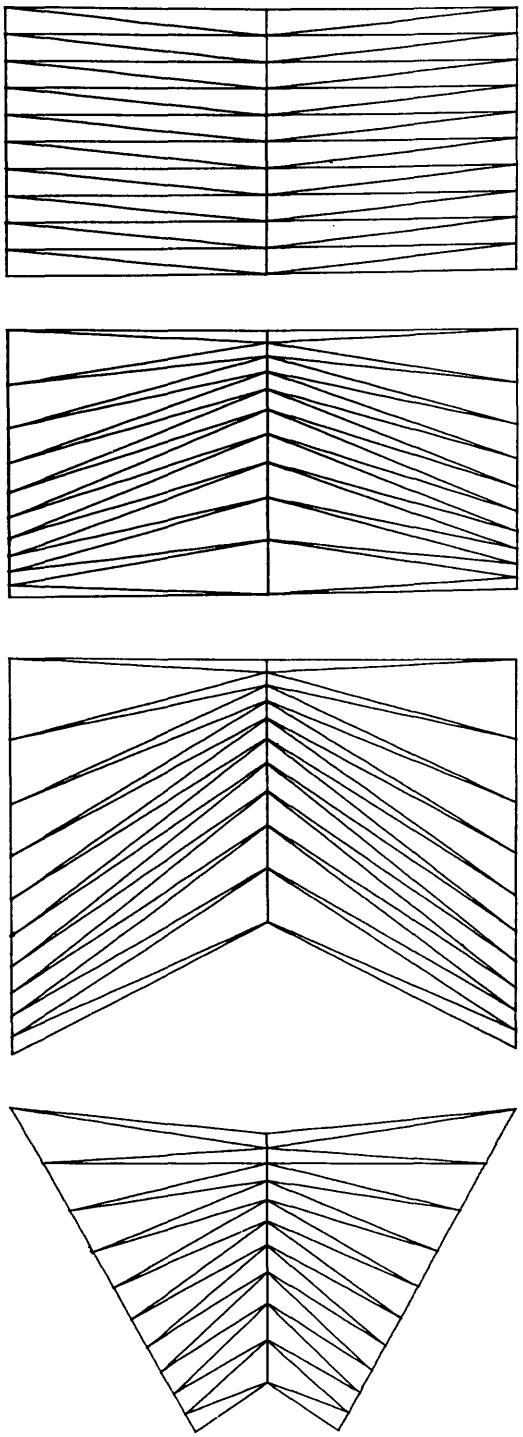


Figure 5

instance, the bottom scoring pattern in Figure 5 is shown as a simulated video picture in Figure 6, and is shown as a photo of the actual three-dimensional folded object in Figure 7.

Both of the examples sited above, the 'bird form' and the 'folded mosaic,' have an associated geometric and topological transformation for achieving a specific three-dimensional shape. The following structure is determined solely by a topological transformation.

OCTET-TRUSS

The traditional 'octet-truss' is produced by a geometric generating system whose part rule predetermines it to be exclusively flat. I have created a number of topological transformations for this well-known structure which will map it to some desired surface shape. Figure 8 shows a sequence of domes of increasing curvature being created from one of these transformations. Each transformation preserves some geometric feature of the structure while distorting others, in order to achieve the required surface discription.

RESCH SYSTEM FOR HANDLING DISTORTION

The impossibility of directly mapping a regular network onto a domed surface without distortion has been noted earlier. I have overcome this logical impossibility of building domes from identical pieces, by putting the required expansion and contraction in the folded crevices. The required variation of member length and facet dimension has been shifted to a variation in the angle between the modular plates. The procedure clearly isolates the necessity for variable length from the desire for identical modular components.

Therefore, it is possible to construct any variety of different domed surfaces from a repetition of identical modular plates by simply varying the dihedral angle between the plates. The mosaic fold shown in Figures 3 and 4, for instance, is composed of only two distinctly different triangles. A small collection of these two triangles will create a module which when repeated will form the mosaic of an entire structure.

It is easy to conceive of machines which would readily change this collection of plates by varying the fold angle between the plates rather than the dimensions of the plates themselves. In developing such an automatic manufacturing process for architectural structures, it has seemed wise to first develop experience at automating the production of models. Two automatic techniques have been explored.

First, hand scoring of the flat sheets to be folded has been replaced by numerically controlled scoring. This was achieved by modifying a flatbed plotter with the addition of a custom made scoring stylus and four pounds of dead weight to emboss a score line into a sheet of rigid vinyl. In exploring the sculptural and architectural possibilities of the 'bird form,' this technique proved invaluable, as every designed form had a unique scoring diagram.

Second, the automatic scoring procedure of scoring paper and vinyl sheets was extended to metal plates by a photo-chemical milling technique. Instead of pressure scoring, as in paper or vinyl, a metal sheet requires the removal of material along the line to be folded. Again a digitally controlled plotter was used to draw the score

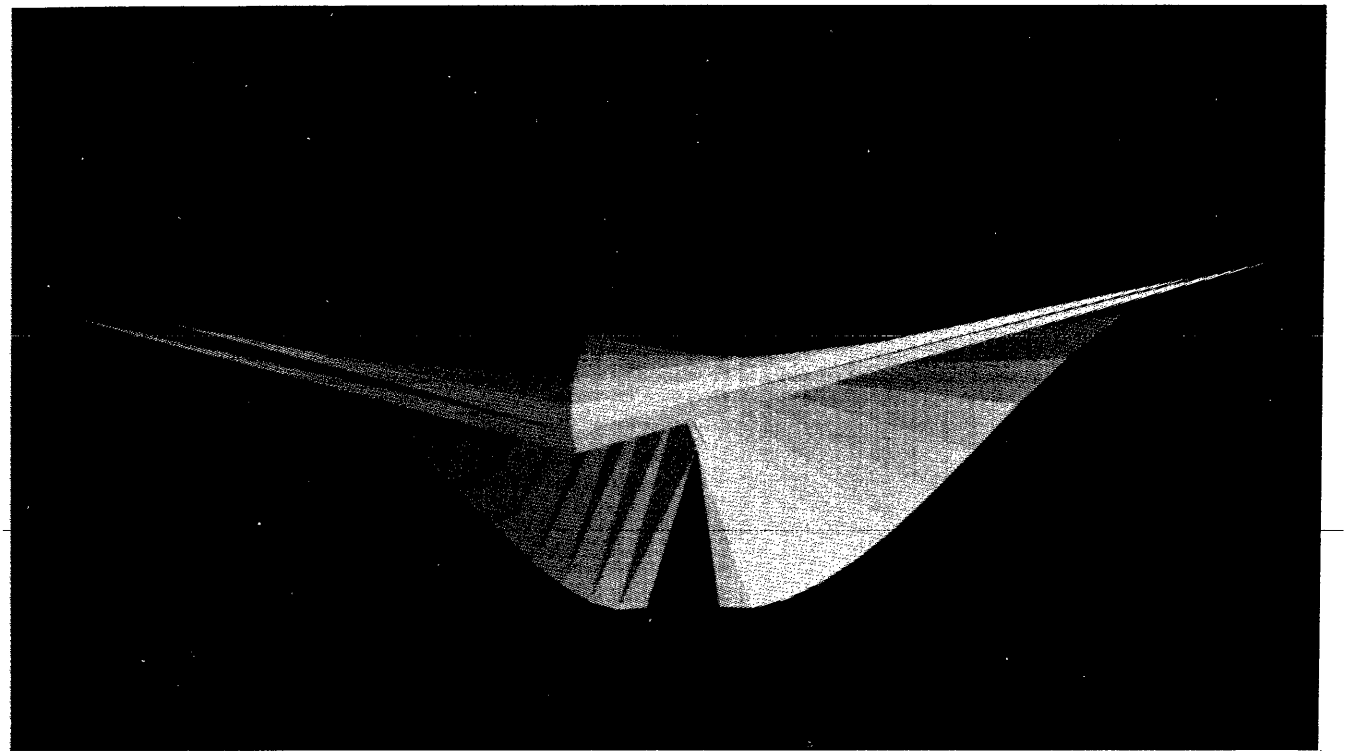


Figure 6

Figure 7

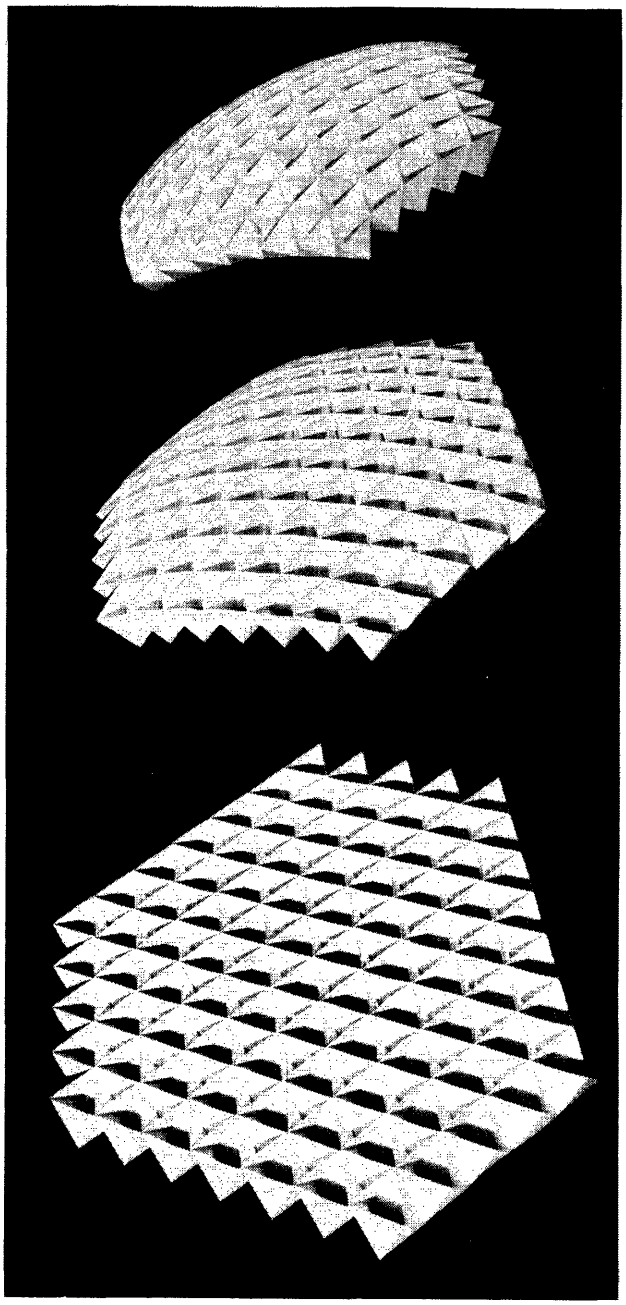


Figure 8

lines. This artwork was transferred to an aluminum plate and the pattern chemically engraved. The sheet was then folded by hand. The 'bird form' on the cover of the "Communications of the ACM" was produced in this manner.

CONCLUSION

My work in the area of structure design, having been coupled with the computer as a medium of design and production, seems to suggest its possibility as a communications medium. The obstacles of achieving a facile relationship of people and things seems to inhere not so much in the structure of things themselves as the structure of our ideas and values. I believe from present experience that it is possible to reintroduce a 'soft prototype'; to pay attention to the subtle variations of user needs; to conceive of objects as a continuously varying class of solutions to a continuously varying set of needs; and to use these needs as input to a transformation upon this topologically conceived object class such that it determines a specific set of instructions that will work within the variations made possible by automatic machines and process.

I believe that a large number of specific demands on a production process can result in an equally large number of individual objects, of a class of objects, at no substantial increase in total cost, if the system is so designed. This total concept I have called topological design.

An informal graphics system based on the LOGO language*

by WILLIAM M. NEWMAN

Queen Mary College,
London, England

INTRODUCTION

This paper describes an attempt to create an *informal* graphics system: a system simple, flexible and inexpensive enough to be used by non-professional programmers for day-to-day problems. Informal computer systems of a nongraphical nature, most of them based on the APL or BASIC languages or on the RAND Corporation's JOSS system, are now quite commonplace. They are invaluable for teaching programming and for use by 'occasional' programmers in research and industry. In the future, as graphics equipment costs fall, the use of informal graphics systems is likely to become increasingly widespread.

Simple line-drawing functions can be added to any conversational language without much difficulty; this is one way of creating informal graphics systems that has become quite popular. There is a danger, however, that if one tries to add to the system more powerful facilities, such as functions for transforming or structuring pictures, the system will become unwieldy and demanding on the programmer. The aim on this occasion has been to provide all these facilities, and at the same time to keep the system simple and easy to use. The result is a system that enables even beginning programmers to write interactive graphical programs with ease.

The system has been designed around the LOGO langauge,[1] LOGO is the sole language available to the user, who therefore uses it both to write programs and to control and edit them. The language has been extended by adding a number of graphical functions. Picture parts and symbols are defined as procedures, and a special DRAW function allows these symbols to be placed in any rectangle on the screen; this provides a very simple means of scaling and rotation, and is a technique that may be used to several levels of depth in defining structured pictures. Symbols need not be predefined, since procedures may be created or modified by the program itself. An interesting feature of the system is the absence of conventional global variables or data structures; instead sequential files are used whenever data must be stored nonlocally. These aspects of the system, and their relevance to future informal graphics systems, are discussed at some length in this paper.

* This research was supported by the Science Research Council.

THE CHOICE OF LANGUAGE

LOGO is essentially a string processing language. It therefore contrasts sharply with the arithmetic processing languages generally used in graphics systems. It is possible to perform arithmetic in LOGO programs, but the functions to do so have the relatively unfamiliar prefix form, and the numbers they manipulate are stored as strings. Arithmetic expressions are therefore less compact than when written in normal infix notation, and are evaluated somewhat less rapidly.

Nevertheless LOGO has much to recommend it as an informal graphics language. It combines to an unusual degree the simplicity needed by the beginner or occasional user with the powerful structure needed by more serious programmers. It lends itself very conveniently to the use of procedures for picture definition. The simplicity of the language makes it feasible to implement a very compact interpreter. LOGO's emphasis on string processing, although it reduces the efficiency of arithmetic processing, makes the system more versatile, and hence increases its range of applications. A system based on APL or BASIC would in contrast have few non-numeric uses.

GRAPHICAL FUNCTIONS IN LOGO

A number of criteria influenced the choice of graphical functions to add to LOGO. In the first place, it was desirable to maintain the characteristic simplicity of LOGO by adding as few functions as possible, and by choosing function names that would clearly indicate each function's use. The inclusion of a large number of functions for specifying transformations, as is common in general-purpose graphics systems, would have been out of character with LOGO.

A second problem was the choice of numbering system. For the sake of speed and simplicity, previous versions of LOGO have always restricted numbers to be signed inte-

gers. Unfortunately it is difficult to define scaled pictures if only integer scale factors are permitted. Consideration was given to extending LOGO to handle real numbers, but this was found to complicate the arithmetic and graphics functions excessively. Instead a scaling notation was adopted that uses rectangular instance positions, defined in integer coordinates.

For the same reason, integer coordinates are used to define basic graphical entities. These may be placed anywhere on the screen's 1000×1000 grid, using the following functions:

MOVETO *x y*	move beam to (*x,y*)
MOVE *dx dy*	move beam through distance (*dx,dy*) from current position
LINETO *x y*	draw line to (*x,y*) from current position
LINE *dx dy*	draw line of length (*dx,dy*)
DISPLAY *s*	display string *s*, starting at current position.

These functions have the effect of adding to the information being displayed on the screen. The user may type these functions as direct commands to the system, and hence may create a picture on-line.

Alternatively these functions may be used within LOGO procedures. A rectangular box could be defined as follows:

```
TO BOX/LEFT/ /BOTTOM/ /RIGHT/ /TOP/
10 MOVE TO /LEFT/ /BOTTOM/
20 LINETO /RIGHT/ /BOTTOM/
30LINETO /RIGHT/ /TOP/
40 LINETO /LEFT/ /TOP/
50 LINETO /LEFT/ /BOTTOM/
END
```

This procedure could then be used to display a rectangle of any size at any position on the screen. For example, the command BOX 0 0 1000 1000 would draw a square border around the screen.

THE DRAW FUNCTION

The technique just described for defining symbols has one clear disadvantage: the entire symbol must be defined in terms of the variables denoting its size and position. The DRAW function removes this difficulty. It allows the programmer to define a symbol at a fixed size in its own arbitrary coordinate system, and to specify the size and position of each *instance* of the symbol separately by defining the corners of the enclosing rectangle. For example, the BOX symbol could be redefined as follows:

```
TO BOX
10 MOVETO 0 0
20 LINETO 1000 0
30 LINETO 1000 1000
40 LINETO 0 1000
50 LINETO 0 0
END
```

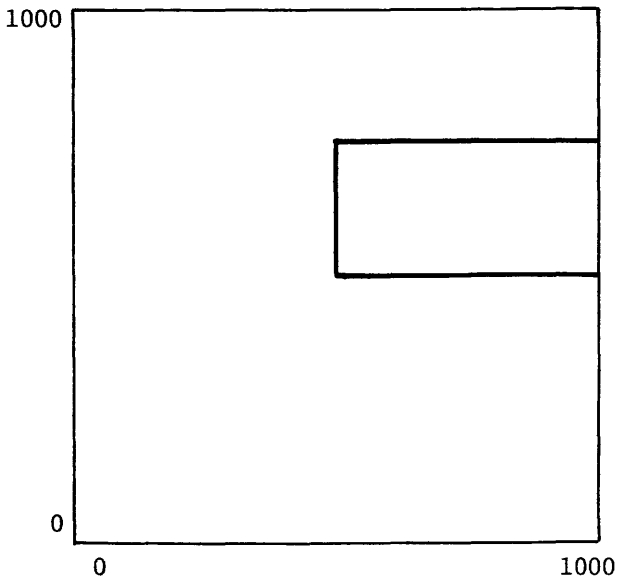


Figure 1—Draw box in "500 500 1000 750"

A rectangle could then be drawn as shown in Figure 1 by means of the following command:

```
DRAW BOX IN "500 500 1000 750"
```

Rotated instances are specified by adding a fifth element to the instance rectangle definition, denoting clockwise rotation in degrees. For example, the command DRAW BOX IN "500 500 1000 750 30" has the effect shown in Figure 2. The symbol is first positioned as speci-

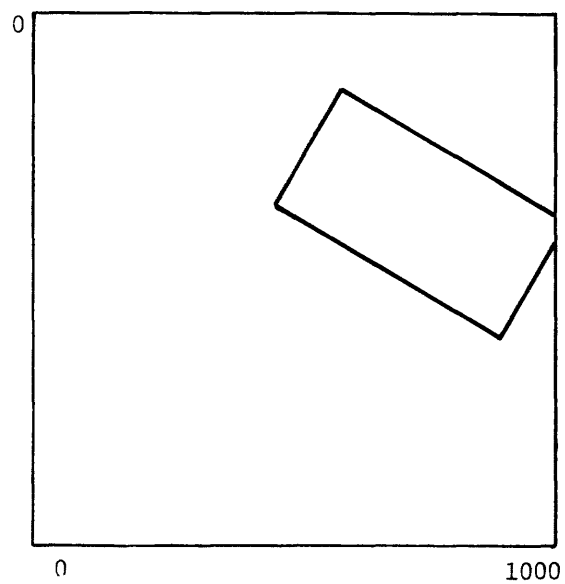


Figure 2—Draw box in "500 500 1000 750 30"

fied by the first four parameters, and is then rotated through the angle denoted by the fifth. An ARCTANGENT function is provided for calculating rotation angles when only the slope is known.

The DRAW function may itself be used in symbol definitions:

```
TO HOUSE
10 MOVETO 0 500
20 LINETO 500 1000
30 LINETO 1000 500
40 DRAW BOX IN "0 0 1000 500"
END
DRAW HOUSE IN "300 300 700 800"
```

The result is shown in Figure 3.

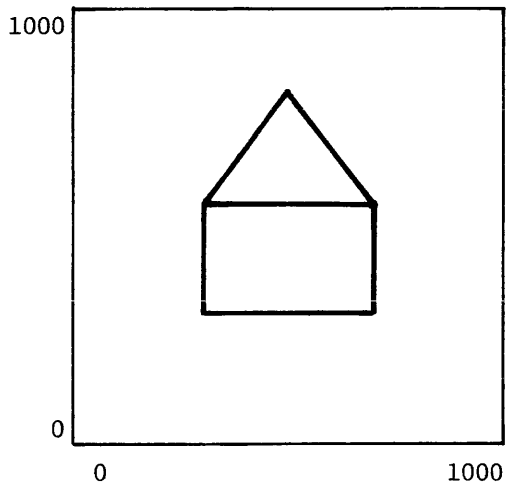


Figure 3.—Draw house in "300 300 700 800"

THE SIZE FUNCTION

The reader will notice that symbols are implicitly defined within a 1000 unit×1000 unit definition space. This in fact a default size that may be overridden by using the SIZE function. For example, the HOUSE symbol could be defined to a size of 10×10 units, without affecting any picture in which it is employed, as follows:

```
TO HOUSE
10 SIZE "0 0 10 10"
20 MOVETO 0 5
30 LINETO 5 10
40 LINETO 10 5
50 DRAW BOX IN "0 0 10 5"
END
```

The inclusion of a SIZE function call in a symbol definition serves two purposes. Firstly, its arguments are used as denominators in determining scale factors for each instance of the symbol. Secondly, it defines a rectangular *window* onto the symbol: all information lying outside this window is clipped from the picture on the screen. By using variable SIZE arguments it is possible to view any rectangular region of the picture defined by a procedure.

As the window onto a picture is varied, certain symbols may be excluded altogether from the screen. The clipping routine can detect this before the symbol procedure is called, and the call is instead by-passed. This can lead to considerable savings in the time taken to display a small section of a complex picture.

As mentioned earlier, one advantage of defining instances by rectangular boundaries is that integer coordinates can be used throughout. The main advantage, however, is the convenience of being able to use a symbol without knowing about the coordinate system in which it is defined. One can define symbols as an entirely separate phase of programming, much as one writes utility routines separately from the main program. Symbols may be redefined without the danger that their instances will no longer fit. The technique also makes it extremely easy to use a "library" of pre-defined symbols.

SEGMENTED PICTURES

Each time the user types a command such as DRAW HOUSE IN"300 300 700 800", a logically separate segment is added to the picture, identified by the name HOUSE. If another DRAW HOUSE command is typed, the segment is replaced. The user may create a picture out of a number of separate segments, any one of which may be redrawn without affecting the rest. An ADD function, otherwise identical to DRAW, can be used to add further instances to the same segment, and a REMOVE function will erase segments.

This segmenting mechanism does not depend on the user to type the DRAW command, but on the fact that the DRAW function is not being called by another DRAW operation. DRAW operations that occur within a symbol, as for example in the HOUSE symbol, do not create separate segments. If we consider these as branches from the node represented by the DRAW operation that calls the symbol, then it is only the DRAW operation at the root of the resulting tree that generates a separate segment.

INPUT FUNCTIONS

The language includes only two input functions: REQUEST, which reads a string from the keyboard, and PENPOINT, which reads a coordinate pair from the tablet with which the display is equipped. The PENPOINT function returns a value each time the stylus is pressed down on the tablet surface.

Since the system is designed to run in a single-user environment, it would be feasible to read the stylus position continuously and to execute a procedure every time a fresh position is received. This would permit the picture to move dynamically with the stylus. Unfortunately the slow speed of the input connection from the display to the computer made such a form of interaction infeasible in this case.

FILE HANDLING IN LOGO

The principal feature that raises the system from the status of a graphical plaything to a useful system is the file system. Sequential files, consisting of records in the form of lines of text, may be created and accessed by means of the following functions:

START*f*	open file for reading, name *f*
READ *f*	read a record from *f*
WRITE *f r*	write record *r* on file *f*
LOOKUP*f*	look up *f* in directory, return TRUE or FALSE
FINISHEDP f	test for end of file
DELETE *f*	delete file *f*

The READ and WRITE functions perform most of the *open* and *close* operations that are normally involved in file handling. For example, WRITE opens the file named, adds one more record to it and closes it; if no file exists with the given name, a new file is first created. These conventions simplify programming and ensure that no file is lost due to failure to close it.

The ease with which interactive programs may be written using these functions is illustrated by the next example. It allows the user to position HOUSE symbols on the screen, using the stylus. Pairs of points are stored as instance positions in a file called HOUSES; this file is periodically read to create the picture on the screen.

```
TO RUN
10 WRITE "HOUSES" SENTENCE OF PENPOINT
  AND PENPOINT
20 DRAW TOWN IN "0 0 1000 1000
30 RUN
END
TO TOWN
10 START "HOUSES"
20 DRAWHOUSES
END
TO DRAWHOUSES
10 TEST FINISHEDP "HOUSES"
20 IF TRUE STOP
30 DRAW HOUSE IN READ "HOUSES"
40 DRAWHOUSES
END
RUN
```

IMPLEMENTATION

Previous versions of LOGO have generally been designed for a time-sharing environment. In this case, however, the system has been designed to run on a small machine serving a single user. This appears to be a more economical and effective basis for an inexpensive, responsive graphics system. The particular computer used was an Interdata Model 4, equipped with a writable control memory, a drum and a refresh display. An advantage of this machine was that it offered, by virtue of its writable control memory, the opportunity to microcode certain sections of the interpreter in order to achieve better performance.

The choice of a single-user environment greatly simplified the design of the most critical part of the system, its memory allocation system. Blocks of memory are allocated from one end of memory, while a single execution stack unfolds from the other; this stack contains both parameters and procedure return addresses.

In most other respects the system follows quite closely the design of earlier LOGO systems.[1] An EXECUTE function has been added to permit procedures to modify each other. This function takes as its argument a string, which is parsed and then executed. Strings representing procedure editing commands can be constructed and then executed in order to modify procedures on-line. For example, the following is the kernel of a procedure to add lines to the procedure PICTURE; each line added consists of a LINETO function whose parameters are the coordinates of a point indicated with the stylus:

```
TO ADDALINE /N/
10 EDIT PICTURE
20 EXECUTE SENTENCE OF /N/ AND SENTENCE
OF "LINETO" AND PENPOINT
30 END
40 ADDALINE SUM OF /N/ AND 1
END
```

The processes to handle graphical functions are organized as shown diagramatically in Figure 4. Lines and text are clipped and transformed into screen coordinates, and are then formed into display instructions and added to the display file. The IN and SIZE functions, which specify transformation parameters, are passed to a *concatenation routine* which combines the current transformation with the one specified, in order to generate a new transformation; the old one is saved on the stack. The IN function, although it lexically follows the procedure call to which it applies, is in fact processed first. This permits the new transformation to be set up in readiness, and allows the procedure call to be by-passed if the symbol it represents lies off the edge of the screen. After the procedure has been executed the old transformation parameters are restored from the stack.

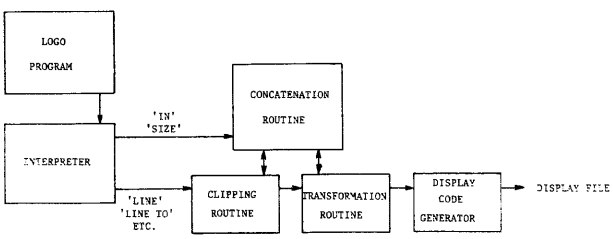


Figure 4.—Organization of the transformation process

This process differs in only one major respect from earlier transformation systems for handling display procedures:[2,3] it includes a clipping routine capable of clipping to an arbitrary polygonal clipping region. Nonrectangular clipping regions are likely to arise with the use of rotated instances, since the rectangle defining the instance may intersect with the edge of the screen. This problem has been avoided in the past by specifying the position and scale of an instance, rather than its dimensions, and by clipping after transformation. In this case it was necessary to choose between three different solutions to the problem, each involving a different sequence of clipping and transformation:

(a) Transform the symbol to screen coordinates, then clip to the non-rectangular intersection of instance and screen (see Figure 5a);
(b) Transform the screen boundary back into the symbol coordinate system, compute the intersection of symbol and screen boundaries, clip the symbol and then transform it to screen coordinates (Figure 5b);
(c) Clip the symbol against its own boundary, transform it to screen coordinates, and clip it against the screen boundary (Figure 5c).

The first and second methods both involve clipping to non-rectangular boundaries. The first has the added disadvantage that large numbers of lines may be transformed unnecessarily, since they are eventually clipped out of the picture. Method (c) suffers less from this drawback, but involves a separate transformation for each successive rotation applied to each symbol; in a highly structured picture this could lead to unacceptable amounts of computation. Method (b) is the only technique that avoids unnecessary clipping and transformation, and has for this reason been adopted.

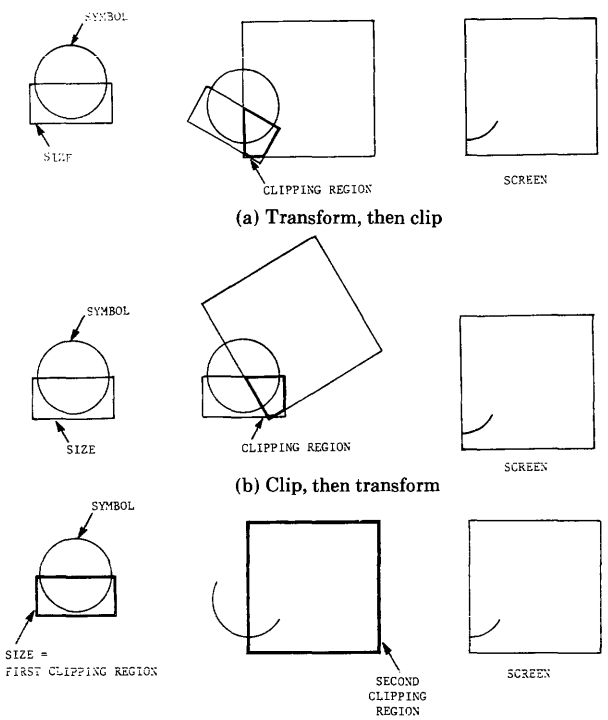


Figure 5.—The three clipping and transformation sequences

Clipping is performed by a version of the algorithm devised by Sutherland and Hodgman.[4] This algorithm performs both types of clipping required by the system: it functions as a *polygon clipper* to compute the intersection of symbol and screen boundaries, and as a *line clipper* to clip the symbol itself. In both cases the algorithm will handle arbitrary convex polygonal regions. The same algorithm is used whether or not rotation is involved; this results in some loss of speed in clipping, but greatly simplifies the design of the transformation software.

CONCLUSION

The system described in this paper was built with two aims in mind. One aim was to construct a system that offered a flexible high-level language with powerful graphics facilities, that was easy to use and that could be run on a small, inexpensive computer. The other aim was to explore techniques that might be applicable to future informal graphics systems. In some respects these aims were in conflict: it was not economically feasible to provide certain features, such as the capability to handle real numbers, that would certainly be provided in future systems.

One feature that was included largely for experimental purposes was the file system. As small computers become cheaper, it is likely that many graphics systems will use these computers as their main source of processing power, and multiaccess systems will exist mainly to support shared file systems. The LOGO system was therefore designed to simulate the combination of a single-user graphics processor and a remote file system. Although only sequential files are provided, LOGO gives the user the capability to construct procedures that search for data in a non-sequential fashion: in effect he can superimpose a non-sequential structure of his own choosing on the sequential file. It is likely that experience with the system will generate useful information about the types of file structure needed by remote interactive users.

REFERENCES

1. Fuerzeig, W., Papert, S., et al., *Programming Languages as a Conceptual Framework for Teaching Mathematics*, Report No. 1889, Bolt, Beranek and Newman Inc., November 1969.
2. Newman, W. M., "Display Procedures", *Comm. ACM*, Vol. 14, 9, October 1971.
3. Newman, W. M., Sproull, R. F., *Principles of Interactive Computer Graphics*, pp. 137-167, McGraw Hill Book Co., 1973.
4. Sutherland, I. E. Hodgman, G. W., *Reentrant Polygon Clipping*, in publication.

Graphics and interactive systems—Design considerations of a software system

by DR. ROBERT C. GAMMILL

University of Colorado
Boulder, Colorado

and

DAVID ROBERTSON

*National Center for Atmospheric Research**
Boulder, Colorado

INTRODUCTION

The National Center for Atmospheric Research (NCAR) maintains a CDC 6600 and a CDC 7600 computer for doing numerical studies related to the atmosphere. These computers are operated in a batched multi-programming mode by a unique operating system implemented by the system's development staff (seven programmers). Hard copy output from the computers is handled either by the line printers or by the two Computer Output Microfilm Recorders (COM). A large library of graphic routines is frequently used by the applications programmers to produce outputs ranging from contour maps to movie titles. Around 600,000 frames of graphic output are produced each month, and nearly 80 percent of all jobs generate some graphic output.

The obvious success and importance of microfilm graphics at NCAR led to the rental of a CDC GRID (Graphical Interactive Display) to investigate the potential of interactive graphics in atmospheric research. The primary question was whether interaction would prove as important to the atmospheric scientist as does graphical output. The answer to that question has not yet become clear, with user reactions ranging from enthusiasm to disinterest. The purpose of this paper is to describe a software system aimed at making the GRID as useful as possible to the atmospheric scientists and programmers at NCAR within a context of carefully limited allocation of system and manpower resources.

Three topics will be examined. First, the context and limitations imposed by choice, by the existing facilities and by the orientation toward atmospheric research will be described. Second, a set of goals which grew out of the context will be itemized. Finally, the form of the implementation will be briefly described, related to the goals, and evaluated as to success.

* The National Center for Atmospheric Research is sponsored by the National Science Foundation.

SYSTEM CONTEXT

Perhaps the most important features of a context are the set of constraints under which any new system must exist. At the time the GRID arrived, the NCAR operating system was undergoing major revision and change brought on by the arrival of a CDC 7600. Little help could be provided by the systems staff. Any implementation had to mesh with existing system structures. The NCAR operating system uses quite different I/O and supervisory linkages than CDC systems. Therefore the GRID system implementation had to be complete, not just an extension of a CDC package. At least as important were a set of philosophical constraints imposed by NCAR's computing environment.

The CDC 6600 to which the GRID was to be connected is used primarily for fast turnaround batched Fortran jobs. Fast turnaround, for all but very long running jobs, has been one of NCAR's most cherished traditions for which the operating system as well as human procedures have been specifically tuned. It was therefore decided that under no circumstances was the GRID to cause significant degradation of that fast turnaround. This led to the decision to treat GRID jobs the same as any other batched job for system resource allocation. It also implied that host computer-GRID communication should be held to as low a frequency as possible.

The last constraint was that the GRID was on trial to see if it was in fact useful. This made it mandatory to develop software that could be fruitfully applied to the class of problems that currently existed at NCAR, rather than develop a complete tool and hope that given the new sophisticated tool, a new class of problems would be posed to use the new tool.

Atmospheric scientists have depended heavily upon all kinds of graphics since long before the computer existed. They also have made extensive use of computers since their inception. What a scientist does on the next computer run is a function of what the graphics, say contour

maps, show him about the last one. In this sense, NCAR scientists have been doing interactive graphics for many years with a response time on the order of minutes rather than seconds. What we needed were interactive programs with graphic output (display), not interactive *graphics*. Therefore it was possible to ignore a large number of the subjects currently in fashion among interactive graphics systems developers.

The last context element was the very large library of available graphics routines at NCAR, their extensive documentation and the great familiarity of users with them. Harnessing the power of these without requiring extensive program changes by the user was a must. Since the COM units had the same 1024×1024 address space as the GRID, we decided upon the straightforward expedient of direct translation of COM instructions to GRID instructions.

DESIGN GOALS

The context, just described, led to the adoption of a number of carefully chosen goals. The most important considerations involved deciding what *not* to do, in order that a useful system could be produced under our particular limitations. Some of these goals are quite different than those chosen in other graphics systems. Despite their limited and simple character, the design decisions they caused have provided a surprisingly versatile and powerful system.

Simplicity, compatability and usability

It was important that the whole effort be kept small, as only four people, each working part time, were available to work on it. Compatability of the software with existing operating system conventions was necessary because there was not systems programmer manpower available for any extensive modification. The graphics generation portion needed to be compatible with existing graphics generation procedures both to utilize that power and to avoid large manpower expenditures on a parallel effort. Since the GRID had only the status of visitor on trial, it was absolutely essential to make it easy for the user to understand how it worked and how to use it. Lack of manpower precluded voluminous manuals and extensive training programs. Therefore the more the GRID could be made to look like a collection of familiar I/O devices the more the user could capitalize on his previous knowledge.

Device independence

Making the GRID look like a combination of reader, printer, and COM unit was important for other reasons. We were to have only one GRID and experience told us that it was apt to be down for extended periods. It might also be removed permanently at any time. Research activities at NCAR had to continue no matter what the current state of the GRID, so making alternative non-interactive devices available as a surrogate interactive terminal made program running always possible, if not interactive. This problem is not unique to NCAR, and greater emphasis upon device *and interaction* independence in the user interface would make interactive graphics a much more usable and dependable tool for the average computer user.

Minimum degradation of operating system.

Besides the decision to treat an interactive job as a batch job for resource allocation, already mentioned, this goal implied that communication between the host computer and the GRID be kept to a minimum. Keeping the number of communications to a minimum simply means that each transmission must contain the maximum amount of information. This in turn forces the interactive graphics terminal resident program to be as self-sufficient as possible, and capable of collecting a fairly large fund of information before communication becomes necessary.

DESIGN DECISIONS

The preceding goals coupled with the initial constraints led to three crucial design decisions:

(a) A high degree of modularity was to be used.
(b) All graphics were to be generated by translation of COM graphic commands.
(c) All interaction was to be through the use of card image transmission as line by line text in an extended form of the NAMELIST I/O input language.

Modularity and staged implementation

The system was to be designed as a set of modules embodying some of the principles of software engineering as expounded by Liskov[1]. In particular, *levels of abstraction* would be identified, and modules designed to implement each level. However, in this case the primary motivation was not correctness, but the ability to stage the implementation. The most essential (bottom most) levels of abstraction would be implemented first, leaving the less essential top levels till later. An advantage of a very structured approach was that part-time personnel could work on different levels of abstraction without necessity for constant interaction. Note that the bottom-up approach appears to be contrary to the top-down method recommended by Liskov[1] and Dijkstra.[2] A schematic diagram of this scheme is shown in Figure 1.

To further ease the development task, it was decided to use a high level language (Fortran), wherever possible, with subprogram calls and named common blocks as the primary mechanism of communication between and within levels of abstraction.

Graphics generation

The advantage of having the entire existing graphical software library available for generating graphical output was self-evident. User familiarity with and good documentation of that library provided a valuable bonus. The need to provide archival graphical output was also solved by translation of the COM instructions since they would be available not only as input to a translation routine but also for direct use.

A decision on how to use the light-pen was deferred until the main body of the package was functional.

Interaction through NAMELIST I/O

Card images are a rather limited form of communication, but the advantage of allowing the GRID to be used as if it were a combined card reader, line printer and COM let the inexperienced user begin writing programs for it without going very deeply into the methods being used. Furthermore, this approach took account of the fact that the user might desire his card image input from one of several possible locations (e.g., card reader, permanent file, etc.) besides the GRID. This not only provided flexibility for interactive runs, when the source of the next input card could be specified interactively, but allowed non-interactive use of the programs when the GRID was out of order.

An extremely important decision was to use NAMELIST I/O as the primary method of communication. A feature of NAMELIST input which made it extremely attractive was what might be termed "mode independence." By this is meant the fact that data-directed I/O is easy to use in either interactive or batched mode. Each statement is self-describing and self-contained, and the ordering makes little difference (unlike list directed I/O).

Since the NAMELIST I/O service routines at NCAR are implemented in Fortran, it was quite easy to modify them to add new features which made NAMELIST input an especially effective tool for interaction. One of these additions was to allow specification of output as well as input values, in a NAMELIST input statement. An example of this is $A=?$ (a statement meaning, "print the value of A").

These extensions to NAMELIST input resulted from the realization that NAMELIST serves to allow interpretive execution of simple assignment statements which are for all practical purposes being inserted in the code at the point of the NAMELIST read. Examples of extensions which proved particularly useful were:

(a) implied DO loop
(b) print statements
(c) assignment statements with variables and expressions on the right hand side
(d) definition of a special array called CORE beginning at location one (allowing patching and dumps).

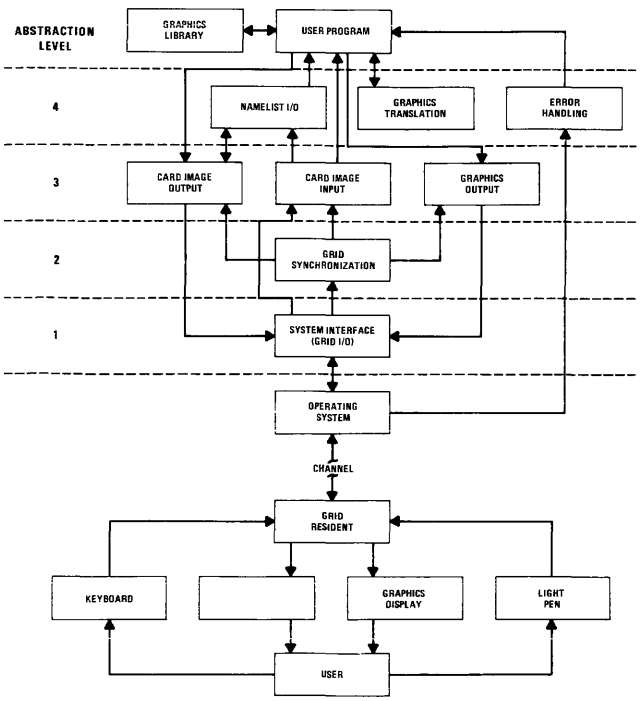


Figure 1—Diagram of GRID software system showing flow of information and levels of abstraction

IMPLEMENTATION

The discussion of the implementation must necessarily be brief. Further descriptive material can be found in the GRID User Manual.[3]

The GRID resident software

The hardware of the GRID is composed of:

(a) a refreshed CRT tube
(b) a light-pen
(c) a typewriter keyboard with supplementary function keys
(d) a mini-computer for operating the device
(e) an operator's console for the mini

Even though the user is not concerned with the mini-computer software, we will briefly discuss how it was designed to match our overall context and constraints. Here again, restriction of the capabilities has allowed simplicity. The primary mechanism for transmission of information from the user of the GRID to the host computer is through the line of text (card image) typed in at the keyboard, and transmitted after possible backspacing, line clearing and retyping, by a SEND key. Text may also be generated by positioning the light-pen tracking cross at a desired position and pushing a special function key, causing text of the form "$MX=132$, $MY=427$," to be generated. This means, that for the majority of informa-

tion transmitted from the GRID to the host, there is no essential difference between the GRID and a card reader or a file of card images.

The only other kind of GRID input allowed is item selection by the use of the light-pen. The result of such a selection is the transmission of the number associated with the graphical object at the time it was generated by the host computer.

Graphical displays and card images are handled by separate parts of the GRID resident software. This means that text and graphics can overlap, distinctly reducing readability. As a practical matter, overlap is normally not a great problem, since the screen is completely cleared preceding each new graphical display transmission.

Host computer software

Abstraction level 1

The lowest level of abstraction of the host computer software, and the first to be implemented, is an assembly language routine named GRIDIO. This routine allows the host computer system to execute op-codes which control or interrogate the GRID. Arguments of the routine include: the op-code, a word for return of status bits, a buffer area and a count of words of information in the buffer area. The op-codes, status word and buffer areas are kept in named common blocks of the Fortran programs which call GRIDIO. GRIDIO has no knowledge of or access to the routines which call it. It simply serves as the sole access mechanism for the Fortran routines at the next level of abstraction, which must communicate with the GRID.

Abstraction level 2

This level includes a single Fortran routine named WAIT. The host computer routines are able to request a number of different kinds of services from the GRID, such as clearing its screen or putting the next line of text on the screen. When the GRID is in the process of providing such service, it must not be interrupted. WAIT provides the synchronization of such service requests.

Abstraction level 3

This level is concerned with card image communication both ways, and graphical communication (complete pictures) to the GRID.

Two Fortran routines, named RDCARD and WRCARD, read and write card images to and from the GRID (and other devices as well). Each of these routines has a single argument, the core location of the input or output card image. Besides calling GRIDIO to execute the appropriate op-codes for communication with the GRID, these subprograms also take responsibility for conversion of the character codes.

The RDCARD and WRCARD routines each use a specified location in a named common block which tells the subroutine the source or destination of the card image. This location contains a logical device name. Changing that logical device name during execution allows the sources and destinations of card images to vary. That change can be effected from within the stream of card images itself, by using the NAMELIST input mechanism to assign a new name to the specified location. An especially important point is that any new interactive device which becomes available will only require software implementation for text from this level down.

Graphical entity transmission takes place at this level through a routine called FRAMEG. The products of COM instruction translation are kept in a work space classified by an entity number. A user indicates which entities he wishes to be displayed by this entity number. FRAMEG moves and combines these various entity display instructions into a single display file and calls the lower level routine to accomplish the transmission. Hard copy can be produced on the COM unit at the same time.

It is important to have the provision for many separate entities which may be combined, revised or recreated. For instance, a common occurrence is a weather map displayed on the background of a world map. By creating the world map once as an entity, any number of weather maps may be displayed upon it without the computer time involved in creating the instructions for the background map each time it is wanted. In the GRID, as in most refresh type display units, flicker and display file space are always a problem. It often happens that a complicated weather map uses so many instructions that the background map won't fit along with it. At this point the user may simply request that the background be omitted for this particular display.

Abstraction level 4

A user may communicate directly with the third level routines by creating and breaking down card images using ENCODE and DECODE statements. These are formatted core-to-core write and read respectively, and are provided on most Fortran compilers used on CDC machines. It would have been more desirable, in some ways, to hook the card image reads and writes directly into Fortran list directed I/O, but this would have required substantial changes to the Fortran I/O service package, which was undesirable due to the context of the project. The result is that a user desiring to put a simple message, with some formatted data, on the screen of the GRID must execute an ENCODE and then call WRCARD to transmit the card image.

This level contains the extended NAMELIST I/O, COM to GRID translation with associated work space management, and provision for error recovery.

NAMELIST input is accomplished through a routine called READLX. It is totally isolated from the GRID, and has no dependence upon specific GRID characteris-

tics. READLX acquires its card images through RDCARD, and prints through WRCARD, the third level routines.

The basic tool of READLX is associating a memory location with a name. This is accomplished by the user calling a routine LEXCON (NAME, IWHERE, ITYPE). NAME is the hollerith character string defining what he is going to call this particular variable, IWHERE is the variable name itself, (an address) which READLX will use in locating it, and ITYPE is the type of the variable (integer, real, double, complex, or character). If ITYPE is zero or the argument not provided, LEXCON assigns a type of integer or real depending on the first character of NAME according to Fortran conventions. LEXCON must be called for each variable name one wishes to reference during a run. Each variable is assumed to be the first address of a singly dimensioned array, so that any variable may have a subscript during READLX input.

Extensions have been added to the input form to allow printing and multiple element definition, as well as full arithmetic assignment statement capability. In each case the extension has been borrowed from a familiar Fortran usage in order to seem as natural as possible.

Several variables are automatically defined by the initial call to LEXCON. These are variables the user may use to help define the behavior of the interactive package, such as ECHO, a variable that controls auxiliary printer and COM output of what goes on during an interactive run.

In the early stages of system development the light-pen was completely ignored. The light-pen is rather difficult to handle in a device independent fashion, as it does not act in a way that is similar to other I/O devices. Upon completion of the early development stages, the 80 character card image and NAMELIST I/O had clearly established themselves as simple yet powerful abstractions. As a result, it was decided to experiment with techniques for hiding the light-pen under the card-image, as this would allow the card reader to substitute for the light-pen. It proved quite simple, using NAMELIST input language, to represent the present position of the light-pen tracking cross by a pair of statements "$MX=1014$, $MY=523$." This statement pair is generated by the GRID when a certain function key is struck. The NAMELIST I/O facility allowed long sequences of such data pairs to be transmitted and saved in a special array where they could be retrieved by the user program after completion of the NAMELIST read. A particular advantage of this approach was that pen positions selected by the function key but later recognized as undesirable, could be edited out of the card image before transmission to the 6600. Furthermore, this method tended to collect a sequence of pen positions before action was requested by the user from the host computer. It must be recognized that this method of handling positions of the light-pen tracking cross would be unsuitable in any application where drawing of general curves or objects was done on a regular basis, as a function key strike must be made for each X-Y position to be transmitted. However, in applications where the user wishes to pick accurately selected positions on the screen (e.g., when making corrections to graphically presented data) the technique is very successful.

In the final stages of system development a capability was added for light-pen selection of a menu item or graphical object. The result of such a selection is that a number assigned to that object is immediately returned to the host computer. Those numbers are assigned to the graphical objects at the time they are generated by the 6600. The number returned to the 6600 is not coded into the form of a card image. This is the *one* exception to the axiom of this system, that all information transmitted from the GRID to a user program is in the form of a card image. The only justification is that it seemed unesthetic. Note that device independence has not been lost, because the routine which returns the number of the object selected to the user program can read a number from a formatted card image instead.

Error recovery is a feature not mentioned thus far but important nonetheless. While GRID jobs are treated the same as batch jobs for resource allocation, it is necessary to avoid run termination in the event of a normally fatal error. Errors frequently occur in typing NAMELIST statements such as $A=B/C$ where C is inadvertently zero. IERPROC is a machine language routine that causes control to be returned to the user when a fatal error occurs. The facility for this was already in the operating system for use with system simulation.

When such an error occurs, a message is sent to the screen containing the error message and a request to the user to specify what to do next. His options are to go back to the NAMELIST routine, return to a specific part of his program, terminate the run, or get more CPU time. CPU time is doled out in one minute units, with a maximum of 10 minutes. To avoid infinite loops, the program is automatically terminated when IERPROC is entered more than 30 times.

EVALUATION

The software system for the GRID has achieved considerable success within the limited goals set for it. The design decisions, to translate display commands, use of the card image as the message unit for interaction, and the use of data directed input as the primary method of interaction, have all proved fortunate. The system is relatively easy to understand and modify. This is attested by the number of different programmers who at one time or another, participated in the development of a module. Users of the GRID found it easy to use, because only a few new calls on system subroutines were needed, beyond those graphics calls they were already making to generate microfilm. The data directed input facility provided users with great flexibility and power for examining the contents of different locations, either by name or by address, and modifying those contents. On many occasions the GRID proved to be especially useful as a debugging tool.

On the negative side, the reader should notice that the only facility available in the final design which requires a refreshed graphic display with light-pen is the menuing and object selection. This capability has not been terribly important in any success the GRID has enjoyed. Furthermore, the refreshed display is often incapable of displaying all the graphic information that a scientist desires, both because of excessive flicker and insufficient storage space. This lack, which is inherent in refreshed displays, has been the most detrimental to the usefulness of the GRID for the atmospheric scientist. Currently the use of storage scope technology (with cursor) is being investigated. Costs of such devices are much lower, and there is no flicker or storage space problem. Somewhat higher demands may be placed on the host computer if it must deal with characters instead of card images. The interactive portion of the software system, and the user programs which depend upon it, would continue to be valid down to the level of RDCARD and WRCARD on the text side, and down to the translator on the graphics side, no matter what device may replace the GRID.

SUMMARY

A software system for a graphical interactive display terminal, and the motivation for the decisions involved, has been described. Several abstractions proved useful in achieving the goals set. The most important of those were, the use of card images for interactive communication, the use of NAMELIST input as the primary user tool, and the direct translation of display commands. Most important system features were a high degree of modularity and device independence.

REFERENCES

1. Liskov, B. H., "A Design Methodology for Reliable Software Systems," *Proceedings of FJCC*, Vol. 41, pp. 191-199, 1972.
2. Dijkstra, E. W., *Notes on Structured Programming*, Technische Hogeschool, Endhoven, the Netherlands, 1969.
3. Walker, J., Robertson, D., Gammill, R., *Grid User Manual*, National Center for Atmospheric Research, Boulder, Colorado, December, 1971.
4. Adams, J. C., Rotar, P. A., *Library Routines Manual*, National Center for Atmospheric Research, Boulder, Colorado, NCAR-TN/IA-67, August 1972.

Recent advances in sketch recognition*

by NICHOLAS NEGROPONTE

Massachusetts Institute of Technology
Cambridge, Massachusetts

In a shocking and almost silly interview with Max Jacobson, Christopher Alexander[1] recounted the following story:

> *"There was a conference which I was invited to a few months ago where computer graphics was being discussed as one item and I was arguing very strongly against computer graphics simply because of the frame of mind that you need to be in to create a good building. Are you at peace with yourself? Are you thinking about smell and touch, and what happens when people are walking about in a place? But particularly, are you at peace with yourself? All of that is completely disturbed by the pretentiousness, insistence and complicatedness of computer graphics and all the allied techniques. So my final objection to that and to other types of methodology is that they actually prevent you from being in the right state of mind to do the design, quite apart from the question of whether they help in a sort of technical sense, which, as I said, I don't think they do."*

While we find notions of a "frame of mind . . . to create a good building" extremely distasteful (and paternalistic), we wholeheartedly admit that computer graphics is guilty of great complication and noise. In general, computer graphics research has been totally self-serving, aptly fitting Weizenbaum's[2] analogy: "It is rather like an island economy in which the natives make a living by taking in each other's laundry."

The following paper describes a specific experiment in computer graphics, one with which Alexander might someday be at ease: sketch recognition. The effort is particularly exciting (to us) because it allows for a wide variety of approaches (some contradictory), modestly executable, with the acknowledgment that the limiting case—a computer that can recognize a hand-drawn sketch with the same reliability as an onlooking human—will require a machine intelligence. The following pages report upon the salient characteristics of an actual computer program, but most of the major issues are far broader than the experience can admit. The reader should seriously wonder (as we continually do), if drawing is a two-dimensional language, does sketching have a syntax and semantics? Is any of HUNCH more than the syntactical processing of a hand drawing?

The founding work in computer graphics was called SKETCHPAD.[3] While this was an effective name, in some way it polluted the notion of "sketching" in any sense of the word. In contrast to SKETCHPAD, "We view the problem of sketch recognition as the step by step resolution of the mismatch between the user's intentions (of which he himself may not be aware) and his graphical articulations. In a design context, the convergence to a match between the meaning and the graphical statement of that meaning is complicated by continually changing intentions that result from the user's viewing his own graphical statements."[4] Sketching can be considered both as a form of introspection, communicating with oneself, and as a form of presentation, communication with others. In the first case, the machine is holding the same pencil, eavesdropping so to speak. In the second case you are sharing a piece of paper with the machine, and both of you are drawing on the same sheet with his (its) own stylus. In both instances memory is the drawing medium (for the human at least) and the design vehicle for looping into the physical world.

We are not suggesting that the heart magically tells the wrist something that embellishes a concept passing from mind to medium. We are proposing that a nebulous idea is characterized by not knowing when you begin a sentence exactly what you are going to say at the end. Furthermore, the final "phrases" are in fact flavored (for better or for worse) by your initial tack and your, our, or the computer's reaction to it. Consequently, in an act like sketching, the graphical nature of the drawing or doodle (that is, the wobbliness of lines, the collections of overtracings, and the darkness of inscriptions) have important meanings, meanings that must not be, but are for the most part, overlooked in computer graphics. "A straight line 'sketch' on a cathode ray tube could trigger an aura of completeness injurious to the designer as well as antagonistic to the design."[5]

* This work has been supported by The Office of Computing Activities The National Science Foundation

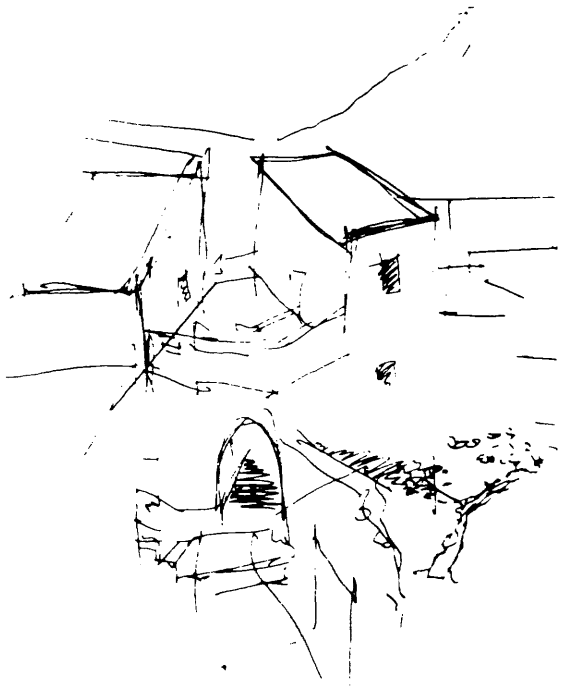


Figure 1

In contrast to most graphical systems, we have built a sketch recognition system called HUNCH that faithfully records wobbly lines and crooked corners in anticipation of drawing high-level inferences about...! The goal of HUNCH is to allow a user to be as graphically freewheeling and inaccurate as he would be with a human partner; thus the system is compatible with any degree of formalization of the user's own thoughts. Unlike the SKETCHPAD paradigm, which is a rubber-band pointing-and-tracking vernacular, HUNCH takes in every nick and bump, storing a voluminous history of your tracings on both magnetic tape and storage tube. HUNCH is not looking at the sketch as much as it is looking at you sketching; it is dealing with the verb rather than the noun. It behaves like a person watching you sketch, seeing lines grow, and saying nothing until asked or triggered by a conflict recognized at a higher level of application.

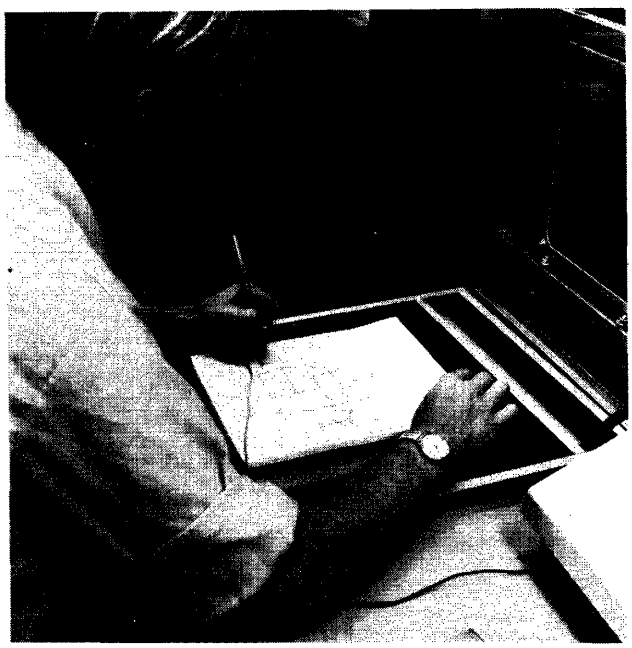


Figure 2

Unlike a completed sketch, that is, a two-dimensional representation, what we have just described is so far one dimensional. The information is recorded serially at the rate of 200 X, Y, and limited Z coordinates per second. This coordinate information is augmented by measurements of pressure upon the stylus, from zero to fifty ounces. At this writing, position and pressure are the only recorded data; one can imagine measuring how hard the sketcher is squeezing the pen or taking his galvonic skin resistance. In addition to position and pressure the method of reporting X, Y, Z (that is, a continuous updating 200 times per second) is in fact a built-in form of clock, which provides the added and crucial features of speed and acceleration.

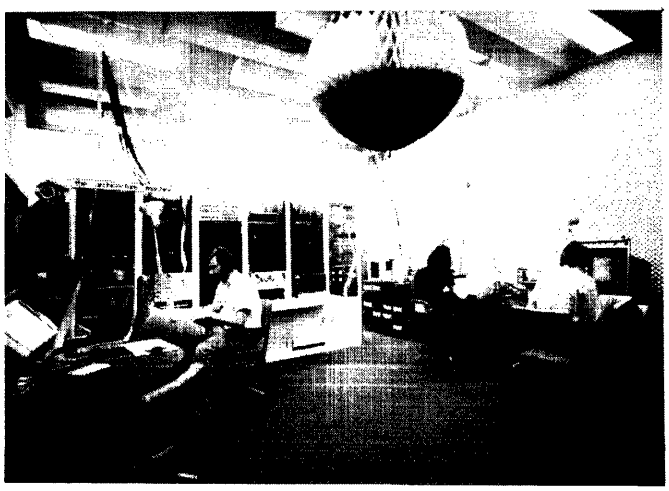


Figure 3

Either on-line or upon command, HUNCH performs certain transformations on the stream of data and then examines it for the purpose of recognizing your intentions at three levels: (1) what you meant graphically, in two dimensions; (2) what you meant physically, in three dimensions; and (3) what you meant architecturally. Each catagory is progressively more difficult. They range from recognizing a square, to a cube, to your being a new brutalist.

GRAPHICAL INTENTIONS

This section describes the most primitive level of recognition, which involves graphical intentions at the level of finding lines, corners, and two-dimensional geometric properties. For humans to "see" these intentions is so

easy and apparently uncontrived that it is difficult to convey the enormity of the computing task without embarking on a technical memorandum of programming techniques. One major difference between the computer's problem and ours is that the computer is given the graphical information as a stream of points (indeed closely spaced but discrete) and does not "see" them as lines without some initial assumption making. A revealing game is to take any line drawing and ask somebody to recognize what is depicted by viewing the drawing only through a small hole in an overlayed sheet that can be freely moved about (thus always hiding the whole picture except for what is seen through the hole). This is how a computer treats the image.

In a similarly sequential manner, HUNCH constructs two representations of the sketch while the user is drawing it, a one-dimensional data structure and a two-dimensional data structure. The first is a faithful record of how the drawing was created in terms of speeds, accelerations, pressures upon the pen. The second is a two-dimensional bit map that is, in effect, a surrogate piece of paper (see Figures 19 through 23). The two structures represent (redundantly) the original sketch, and they are kept intact at all times. All subsequent structures, either sequential or positional, are maintained above and beyond these original descriptions. They may be transient, manipulated, destroyed, updated, or reproduced ad nauseum. In contrast, the original sketch, as represented sequentially and positionally, is maintained as a faithful icon acting like the "real world" to which we can always return for another look. The bit map will soon be replaced by a vision system that looks at the sheet of paper, avoiding the need for surrogate paper.

The process of recognizing graphical intentions includes seven kinds of operations, each of which relies to different degrees on the two structures. The following paragraphs describe specific transformations in their most usual, but not necessary, order. Even though they are described as specific transformations with known inputs, it is usually the case that several guesses must be made and that several candidate resolutions must be carried through, building up evidence for and against. All the transformations are ridden with contingencies that cannot be handled in a rote fashion that puts all of one's faith in one guess.

Diagrams

When one sketches, it is natural to intermingle elements that have a projective geometry interpretation (intersections, limiting contours, demarcations of patterns, etc.) with those that have a diagrammatic intent (symbols, arrows, letters of the alphabet, figures, etc.). Consequently, one of the intial passes at recognition is to separate the diagrammatic elements from the projective elements. There is no foolproof way of distinguishing, for example, arrowheads from rooftops. In some cases it is necessary to leave the ambiguity for a future operation to stumble upon and untangle with "higher order" evidence.

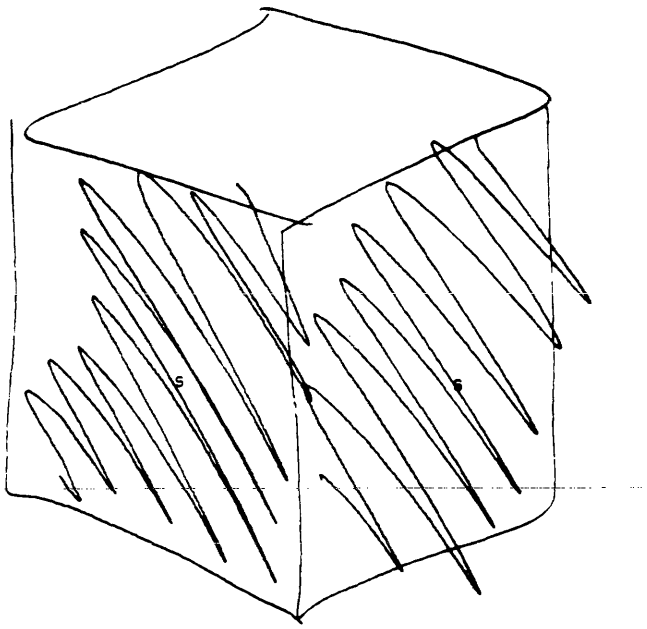


Figure 4

Diagrams fall into two classes: those recognizable by shape and those distinguishable by gesture. An arrow, for example, has a distinctive topology and can be defined in the jargon of line types and joints. A squiggle, on the other hand, is a hand movement, meaning, for example, either shading or "to be erased." The recognition of the arrow is achieved primarily with positional data, whereas the squiggle is more easily found in the sequential stream, in terms of jerking hand motions. The adjacent illustrations depict the sort of weeding out that takes place at this stage. Note that the squiggles are interpreted as "S's."

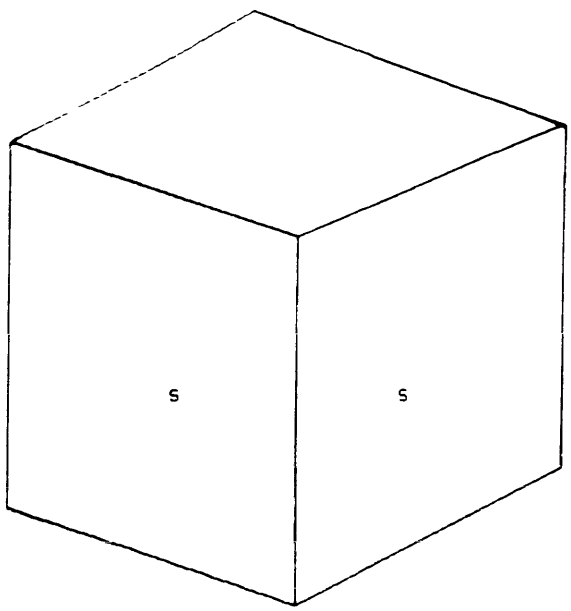


Figure 5

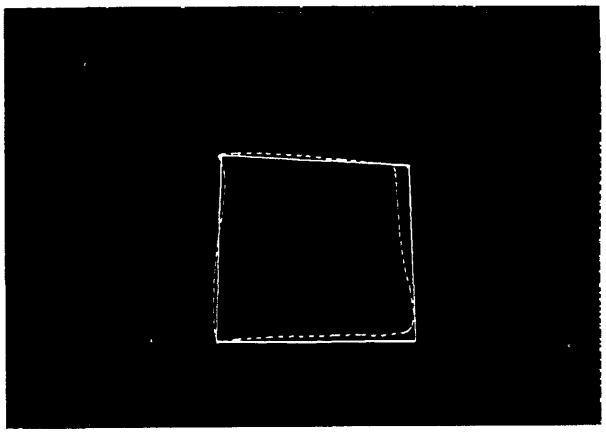


Figure 6

Data compression

Consider that at a 200 coordinates per second ten-minute sketch of a dog results in 3,600,000 bits of sequential data. A major role for any sketch-recognizing system is to compress this data for the purpose of transmitting it to other procedures or other machines. An ultimate case of data compression would be to take the 3,600,000 bits and transform them into: "short-haired poodle that looks like Spiro." A more modest transformation, in the context of architectural drawing, is to reduce the projective geometry elements to a list of nodes and linkages of straight lines and curves.

HUNCH performs this operation with uncanny success, guessing at the intended straight lines, curves, and corners. It achieves this transformation with two simple but powerful parameters of intentionality: speed and pressure. Figures 6 and 7 illustrate the measures of intention in that the first square was drawn rapidly (and sloppily) and interpreted as a square, whereas the second was drawn slowly, hence with apparent caution and intent, and interpreted as an irregular figure with rounded corners. The correlation of speed and pressure to simple intentions yields a powerful measure of graphical

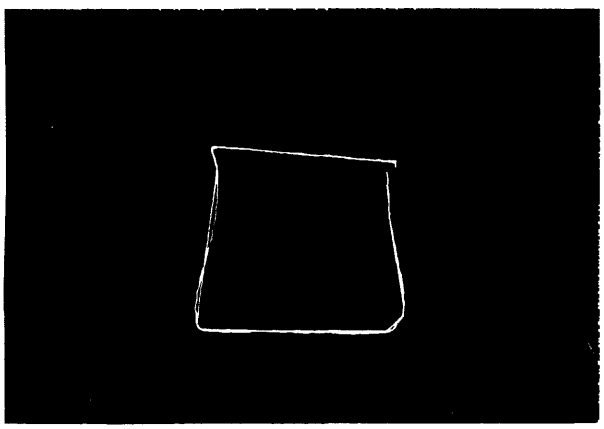


Figure 7

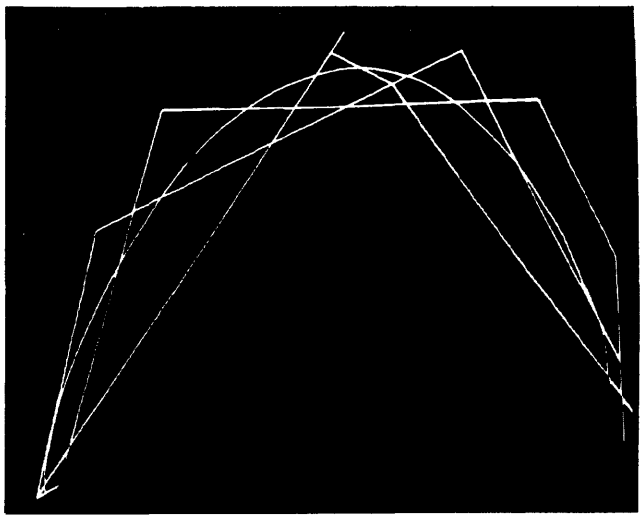


Figure 8

"purpose." Nonetheless, it should be noted that these parameters are very sensitive to the hand of the individual designer and thus must be delicately tuned and tailored. This is achieved at first encounter by a simple scenario of: "draw me a this . . . or that . . . faster . . . slower" and later is revised on-line, ultimately (wishfully) in context.

Curve recognition

A myth of computer-aided design has been that computer graphics can liberate architects from the parallel rule syndrome and hence afford the opportunity to design

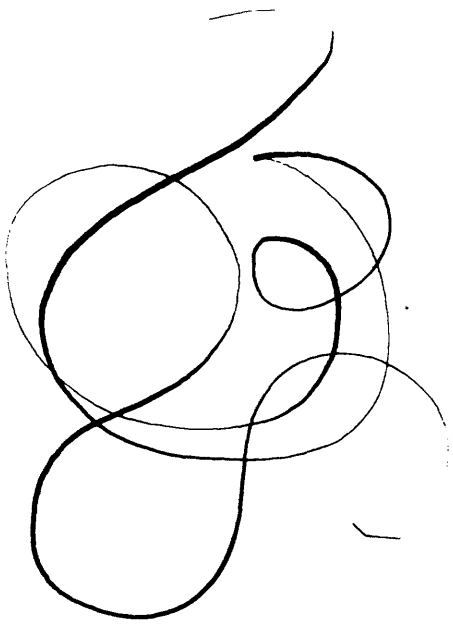


Figure 9

and live in globular, glandular, freeform habitats. We do not subscribe to this attitude. We believe that orthogonal and planar prevalencies result from much deeper physiological, psychological, and cultural determinants than the T-square. Partly as a consequence of this posture, The Architecture Machine Group initially and purposely ignored curves, feeling that straight lines and planar geometries could account for most graphical intentions. However, it is the case that in demonstrating HUNCH, the sketcher invariably incorporates curves in his second sketch, if for no other reason than to see what the machine will do.

Recently we have incorporated curve recognition as a subset of data compressing. The problem is twofold: to recognize and to fit. The recognition is simply a matter of distinguishing a hastily drawn straight line from a purposeful curve. As with the previous examples, speed and pressure provide the most telling evidence and form the basis for most neuristics. However, unlike finding corners and straight lines, recognizing curves requires a greater interplay between the two data structures, because taking derivatives of irregularly spaced points (without interpolation) can be very misleading.

Two approaches have been employed for curve recognition. The first approach (see Figure 8) is to try arbitrarily to straighten all lines with minor variations in parameter weighting. This causes minor variation in the straight line interpretations and wide variations in the curves because of the programming technique. The second approach is to concentrate on the derivatives (second and third) in the assumption that curves are less speed dependent and, by their nature, require more cautious application.

Curves are cumbersome graphical elements in the sense that neat ways for fitting and describing them in a simple "compressed" manner do not exist. Presently we represent them with a B-Spline technique, a method that allows for a high level of curvature continuity and for a compressed representation that employs points that conveniently are few in number and do not lie on the curve. Figures 10 and 11 show the effects of varying the order of the spline. For a more complete account of this technique the reader should refer to the thorough and definitive work of Richard Reisenfeld who is presently working with Ivan Sutherland at the University of Utah.

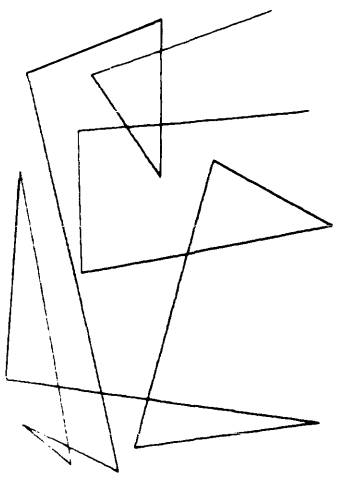


Figure 10

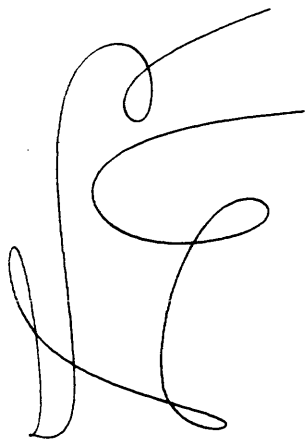


Figure 11

Latching

It is necessary to perform this task of latching, the process of guessing when a line is meant to be connected to a point, with as high a level of reliability as possible because a single unlatched line can make the simplest figure topologically impossible (or implausible) in a planar or volumetric representation. In the early HUNCH days, we assumed that latching could give relatively consistent success when treated in a manner similar to finding corners; that is, we relied on speed and pressure to vary the range in which one would venture a latch. In fact, it worked quite well until a user drew small pictures or incorporated detail like a window in a wall. In these cases the latching routines would be over enthusiastic and latch lines to all the nearby end points, making mullions look like starfish. This was because latching was initially achieved in a very narrow context. More recently, latching procedures have been redesigned to look for patterns in the positional data. Heuristics employ features like repetition, closures, and density to provide evidence that a certain endpoint probably is meant to be attached to a certain other endpoint.

Intersections

When a line is drawn that crosses or abuts another, the initial procedures do not locate the point of intersection. Finding intersections is a straightforward operation that has both significant and misleading results. It is often necessary to carry multiple representations, guessing when intersections are or are not important. For example, in the case of a five-pointed star, fifteen line segments

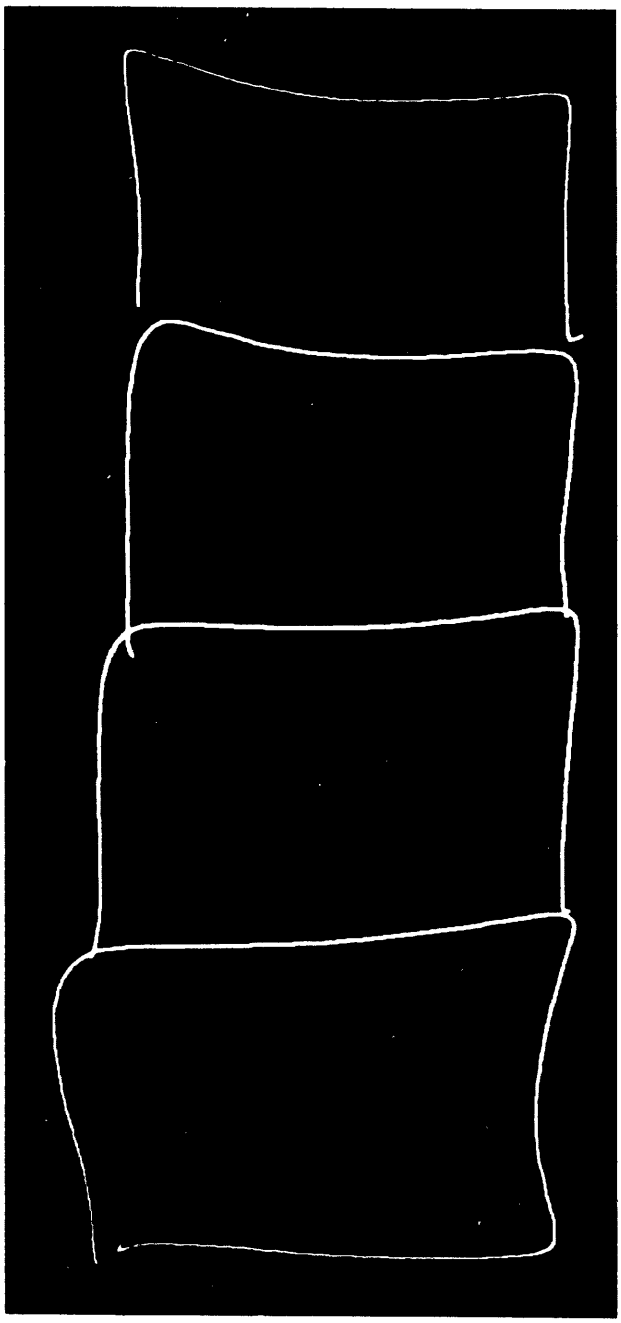


Figure 12

and ten endpoints are returned. This distorts the concept of "starness," intrinsically a five-sided design.

Nonetheless, in most instances, intersections are invaluable for the recognition of higher-order features. One case is an intersection that contains one line that does not "pass through," for example, a T joint. This form of intersection will often be unlatched at a later time inasmuch as T's provide very strong evidence that one plane or body lines behind another. In that sense they represent a more serious passing, not in the same plane.

Implicit Constraints

Early SKETCHPAD experiments included constraint application and resolution such that you could draw two skew lines and apply the constraints of parallelism and similarity in length, and observe the lines meander to equilibrium (if geometrically possible). Similarly,

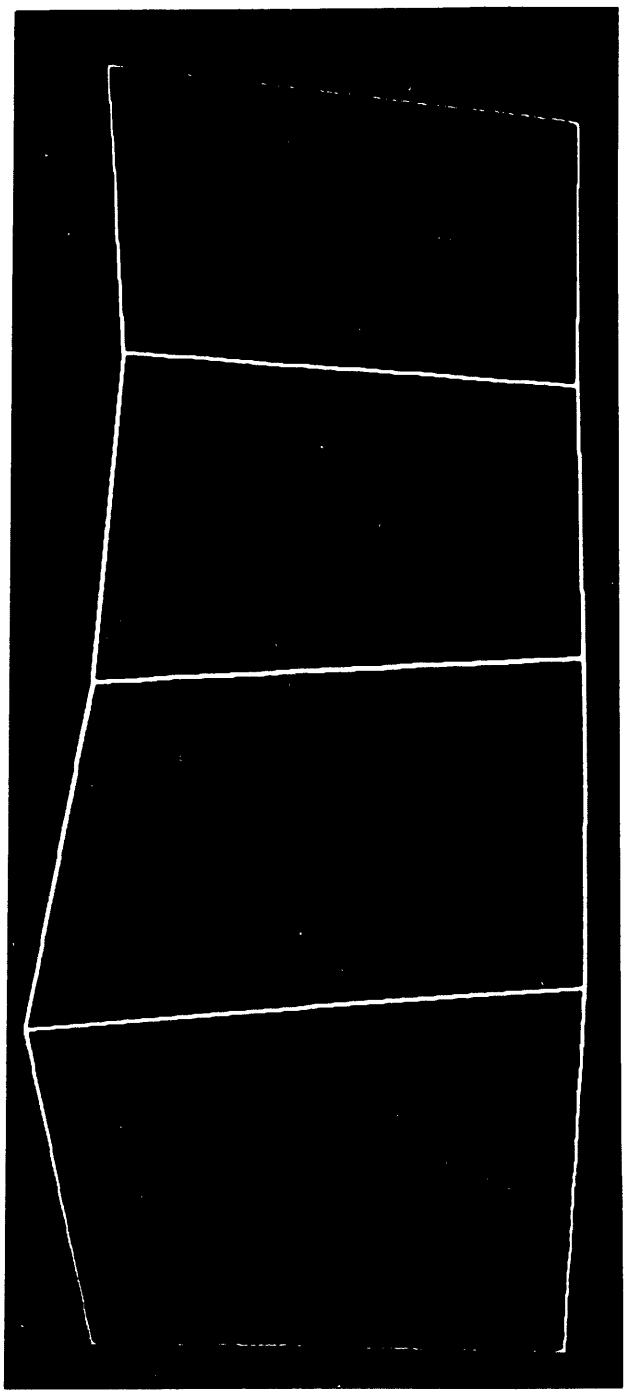


Figure 13

HUNCH supplies four constraints; the only difference is they are initiated implicitly. They include horizontal/vertical, parallel/perpendicular, continuous, and overtraced. They are relatively straightforward computations (described in the adjacent figures); some involve local consideration, and some require a search of the entire image. One can imagine many more implicit constraints, and one can also imagine an evolving set of constraints

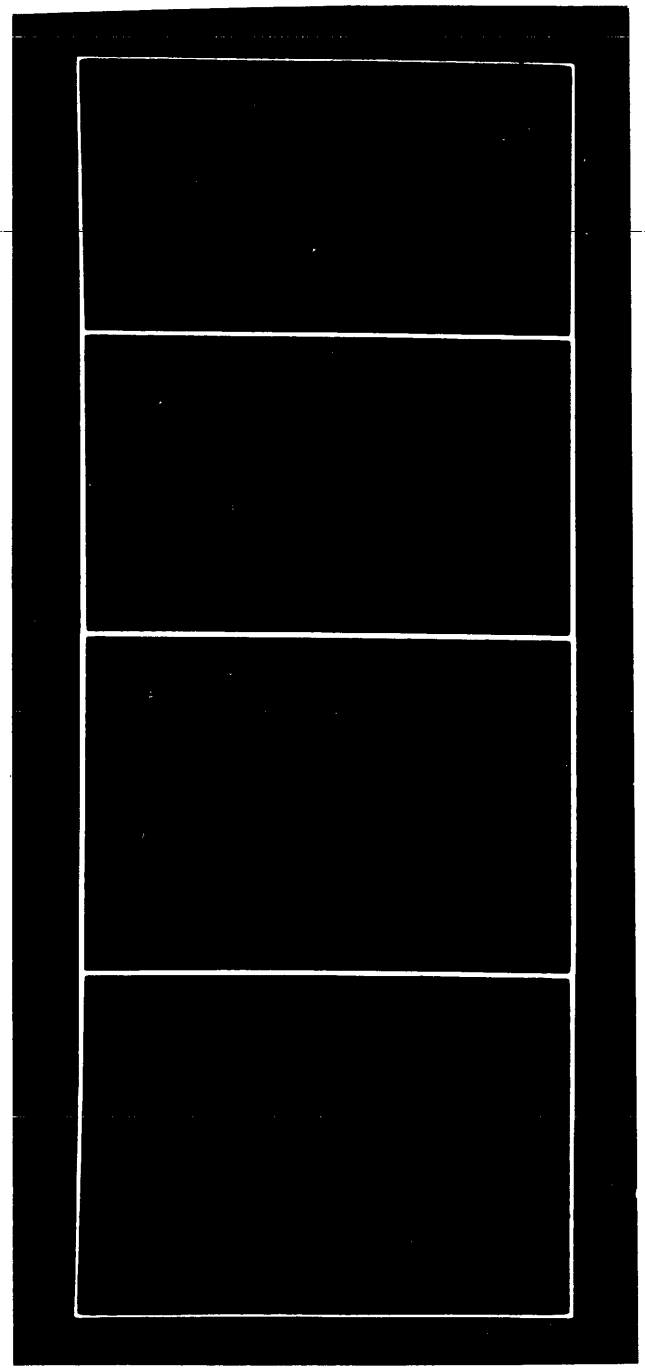


Figure 14

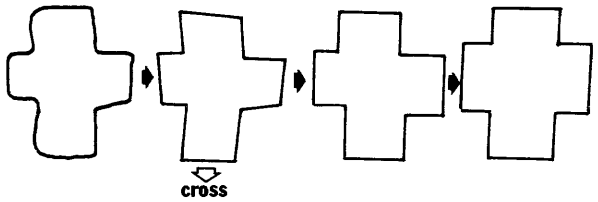


Figure 15

resulting from a particular user's idiosyncracies and habits. These, too, would be a function of speed and pressure.

Overtracing, however, warrants special attention because it is a fascinating drawing behavior that can imply two very contradictory intentions: reinforcement and correction. In "yellow tracing paper operations," so familiar to students and practitioners of architecture, one tends to consider and execute contradictory but exploratory lines, with the result that the representation, if viewed in its entirety, would be a "nonsense artifact." It is also usually the case that, prior to overlaying more yellow paper, the most salient and ambiguous features are overtraced so that the translucency will cover the "noise." On opaque paper, the sketch often starts as light scribbles and construction lines and evolves into a black hodgepodge of many light lines accompanied by studied, purposeful dark lines.

A simple way to handle overtracing is to consider it as a form of implicit erasure of the lines beneath. Or, equally simplistically, one could read the magnetic tape (that is, sequential data) backwards, automatically giving higher credence to the most recently sketched features. Both methods work with surprising success (especially when accompanied by factors of speed and pressure). However, they overlook some of the important implications of overtracing. For example, highly reworded lines "may represent important (perhaps semantic) dispositions toward a design such as being 'concerned about,' 'sure of,' 'puzzled by,' and so on."[6,7]

Shape Recognition

At this point the reader should be discouraged by the disparity between seeking an artificial intelligence and enumerating simple geometric transformations. Nowhere has learning been involved. All previous operations are as syntactical as parsing a sentence or separating words in a speech. Shape recognition begins to raise more challenging questions, for example, at what point is a shape recognized?

Figure 15 depicts the transformations of a cross-like figure achieved in the order in which I have described them. Note that the last representation remains irregular (let's assume I meant a regular cross) in that the four wings are of different proportions. A first thought might be to append the additional implicit constraint of repetition of line length. This in turn could be mapped into the concluding transformation: CROSS (as defined

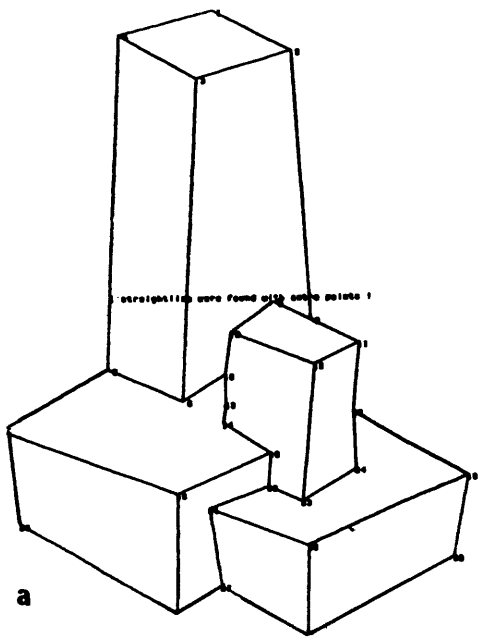
rigorously by a figure with four equal . . . etc.) However, is it not more rewarding to look for "crossness" much sooner? "The very concept of 'cross' furnishes many of the graphical inferences that until now have been handled in some sense brutally."[8,9]

INFERRING A THIRD DIMENSION

How many people are aware that the general attitude of a cube is such that its silhouette forms a hexagon? Do we use such information to understand or to recognize the three-dimensional aspects of cubeness?

The retinal image is a two-dimensional representation that we constantly map into three dimensions with no overt intellectual effort. The psychology of perception is a voluminous field with classic works like those of Gibson[10,11] and Gregory[12] that has provided some clues as to how we see. However, the traditional views of psychologists have been of very little help in making machines that can see or that can infer a third dimension. The reader interested in machine vision per se should refer to the founding works by Oliver Selfridge[13] and his colleagues, the works of Minsky and Papert,[14] Guzman,[15] and a great body of papers emanating from the three centers of robotics: MIT, Stanford, and Edinborough.

Our own interest in machine vision has oscillated between low resolution and high resolution, between geometries and behaviors. One specific experiment is described in an early (1969) NSF proposal in "Machine Vision of Models of the Physical Environment." More recently our interest in vision has settled specifically on the inference making necessary to achieve three-dimensional information from a two-dimensional representation, such as a drawing. Notice that in the case of sketching, making inferences about the third dimension is



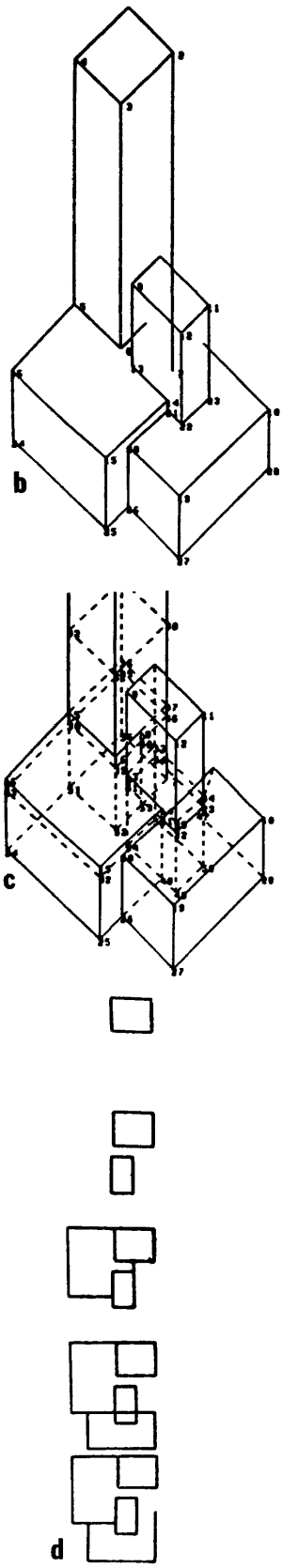


Figure 16

somewhat easier than looking after the fact at a scene of, let's say, a pile of blocks. This is because one has the additional information of "construction sequence," which can be employed in heuristics that make speculations like: this is connected to that, this is behind that, and so on.

The first task of inferring the third dimension in a drawing is to recognize the kind of projection. Is it a plan or a section? Is it a perspective or an axonometric? The two alternatives are distinctly different even though a section and an axonometric, for example, belong to the same family of orthographic projections, and a plan can be considered as a perspective from above with the observer at infinity. They are different because the one group supports the illusion of three dimensions whereas the other requires conventions, consistencies, and a combination of views or the additional cues of shading.

Let's consider axonometrics and perspectives first. They have fascinated researchers in computer graphics, in particular with respect to the removal of all lines and line segments that would be invisible from a given vantage point. The so-called hidden line problem has been exhaustively studied by: Roberts;[16] Kubert, Szabo, and Giulieri;[17] Gilimberti, Montanari;[18] Loutrel;[19] Ricci;[20] and, in a survey that proposed a new solution, Encarnacao.[21] But it is not an interesting problem, because it is deterministic and blatantly solvable though complicated. It is much more interesting to consider the opposite problem: given a perspective, fill in the hidden lines. We say it is more interesting because: (1) it is riddled with ambiguities; (2) there exists no algorithm that will work for all cases; and (3) it can only be handled with a knowledge about the physical world.

Figure 16 illustrates the operations of a program that takes HUNCH input, constructs an axonometric, and maps it into three dimensions with modest accuracy, using limiting assumptions. The primary operations include: (1) estimate the families of parallel lines . . . an N point perspective; (2) find redundant points, stray lines, that is, HUNCH oversights in working in a two-dimensional frame; (3) axonometricize the figure, if necessary; (4) break all T joints; (5) project T's until they intersect a plane as defined by any two parallel lines that each belong to a different family but neither to the family of the projected T; (6) look for parallelograms; (7) furnish guesses at a third coordinate as a function of length and angle away from verticality; (8) project all horizontal planes to intersect any element that protrudes above.

Notice that the eight steps and functions are quite arbitrary; they represent an interpretation of desired results, not an interpretation of how we see. Each operation assumes a model of the world (it can be as simple as orthogonal) that imparts arbitrary legitimacy to the computer program in that it behaves with a nice precision. However, no matter how hard we try, we embed simplifying assumptions, and we can never be assured that handling the abstracted set of arbitrary three dimensional figures will lead to handling the entire set. For example, we can limit the class of sketch to the extent of making this mapping just about deterministic. Similarly, we can broaden it to handle any collections of irregular polyhedra. In the latter case we find that we make implicit assumptions (as opposed to built in limitations).

In contrast to axonometrics and perspective, plans and sections through conventions afford more unambiguous descriptions. They require, however, the additional task of piecing together sections and matching different views. Furthermore, an additional step of recognition is necessary: Is the slice horizontal (a plan) or vertical (a section)? Once again this is usually so obvious to the onlooking human that it behooves us to understand the essence of plan and section. We do not agree, for example, with the often-stated position that a plan and a section should be indistinguishable. Our physiology is such that we tend to witness the world in section but, interestingly enough, to remember it predominantly in plan. In addition, our sense of balance plays a major, unexplored role in the primarily orthogonal structure of human concepts about the physical world, as described by terms like above, in front, right, left, etc.

Unlike mapping perspectives into three dimensions, most energies in the recognition of plans and sections are devoted to the basic determination of which is a plan and which a section. A computer program must draw upon clues like steps, trees, sloping roofs, and take advantage of such facts as: floors are usually horizontal. There will be cases where it will be unclear to even the most experienced architect whether the drawing is a plan or section. It would be wrong to expect a machine to do much better, but it would be right to expect it to ask.

The reader familiar with projective geometry techniques will understand that formats like those employed in mechanical engineering are quite a bit easier to correlate than the typical architectural set of drawing. Unlike with mechanical engineers, architects do not share a general consensus of conventions for dotted lines, auxiliary views, and the like. This difference is less exaggerated in sketching. At this writing, little work has been done on this problem by The Architecture Machine Group. The reader should be referred to the recent work of Ejiri, Masakazu, Takeshi, Hauro, Tatsuo, and Kiyou.[22]

ARCHITECTURAL INFERENCES

An architectural inference can range from recognizing the propensity to use cheap materials to assuming a lifestyle. "When we recall that the process will generally be concerned with finding a satisfactory design, rather than an optimum design, we see that the sequence and division of labor between generators and tests can affect not only the efficiency with which resources for designing are used but also the nature of the final design as well. What we ordinarily call 'style' may stem just as much from these decisions about the design process as from alternate emphasis on the goals to be realized through the final design."[23] "If we see a building with a symmetric facade, we can be reasonably sure that that facade was generated

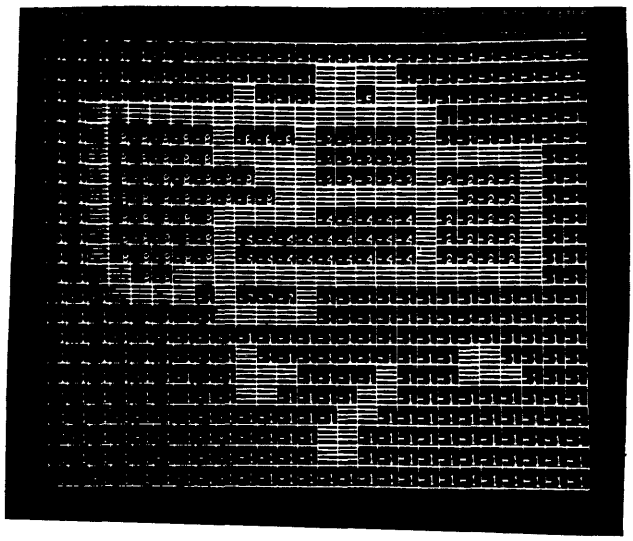


Figure 17

at an early point in the design. If, on the other hand, we see one with many asymmetries, we will conjecture with some confidence that these asymmetries are the external expression of decisions about how to meet internal requirements" (Simon, 1970).

These two quotes may offend the professional architect; the notion of "style" belongs only to history and to a posteriori observation. However, if we replace the word style with intent and suggest that intentions are both implicitly and explicitly manifest in the method of work of the designer, the idea of looking for architectural inferences is more palatable; the problem is to infer what was meant versus what was done. By recognizing architectural implications, one can begin to say something about the past experience of the designer. This is because a large number of decisions are made through prejudice and preconception.

One example of drawing inferences as a function of method of work can be found in an experiment associated with "plan recognition" (see illustrations). The "user" is asked to draw a plan of his house. We find two general methods of drawing such a plan. The first entails describing the external envelope and then subdividing it into rooms. The second involves "walking a line around," space to space, tracing out interior compartments as cells that interconnect. With some confidence we can make the assumption that the first method indicates living in a detached house, for example, where one has the opportunity to witness the "whole" as set upon a plot of land. The second method is symptomatic of living in a high-rise apartment building, where one does not have the occasion

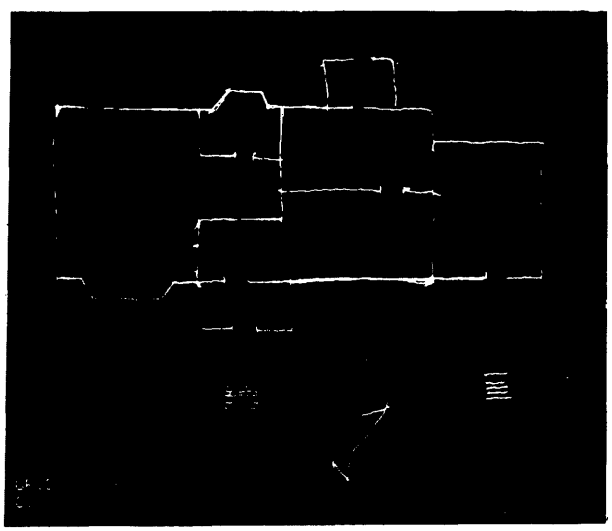


Figure 18

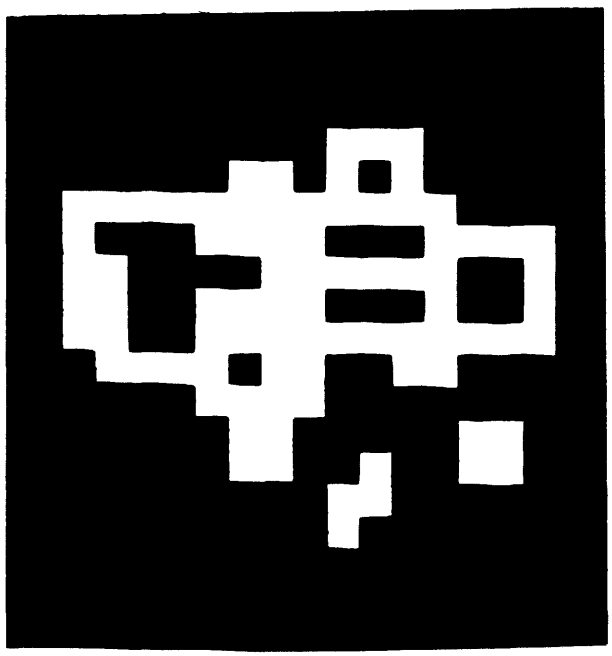


Figure 19

to inspect the external envelope of one's own living space. Another (wilder) interpretation might be that the first person owns his house (has observed it drawn in plan) and the second rents it and is therefore unconcerned with global matters of insurance or heat loss.

More formal examples of looking for architectural intentions can be found in hunting for tendencies to repeat elements, in recognizing a propensity to align boundaries, or in searching for playful and whimsical uses of angles and penetrations. These tend to be symptomatic of superficial constructs especially when viewed as ends unto themselves. A deeper level of intentionality can be achieved in what Gordon Pask calls the "cybernetic

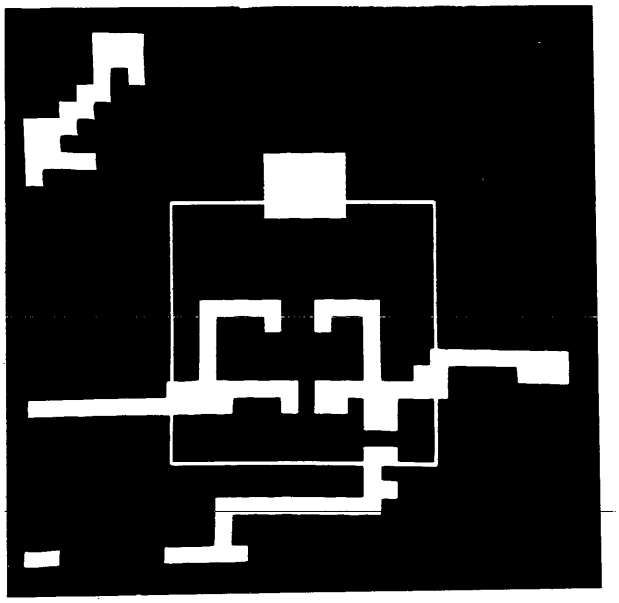


Figure 20

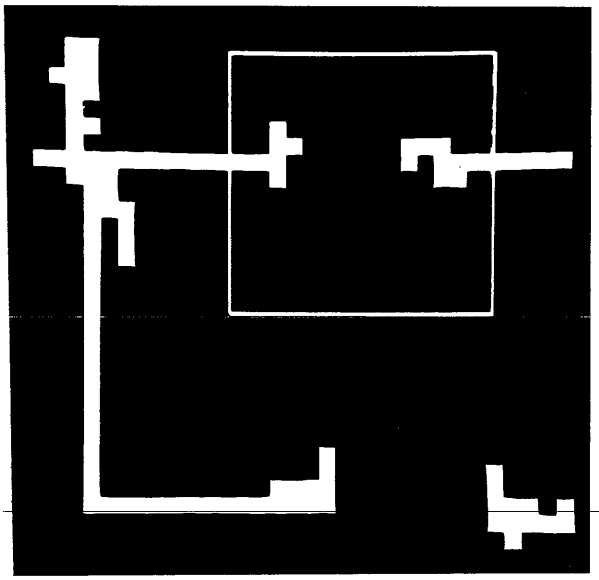


Figure 22

design paradigm" by looking for unstated goals: "It should be emphasized that the goal may be and nearly always will be underspecified, i.e.: the architect will no more know the purpose of the system than he really knows the purpose of a conventional house. His aim is to provide a set of constraints that allow for certain, presumably desirable, modes of evolution."[24]

A principal means of recognizing architectural intentions will be to look for architectural attributes, rather than architectural properties, the physically measurable properties. Architectural attributes are measured in terms of our own experiences and are recognized in discourse by knowing something about the person with whom you are talking. To be sure, they are described by metaphors and analogies; they do not surface in the geometries of a sketch. To emphasize this point, I refer to Thomas Evans's early work[25] on the program ANALOGY as an example of one kind of difference.

The ANALOGY program tackles the so-called "geometry analogy" intelligence test: Figure A is to Figure 3 as Figure C is to which of the following? The Evans program goes through four major steps: (1) the figures are decomposed into subfigures; (2) properties are ascribed, such as inside of, to the right of, above, etc.; (3) "similarity" calculations are determined to successfully map A into B (4) the appropriate similarity is used to map C into

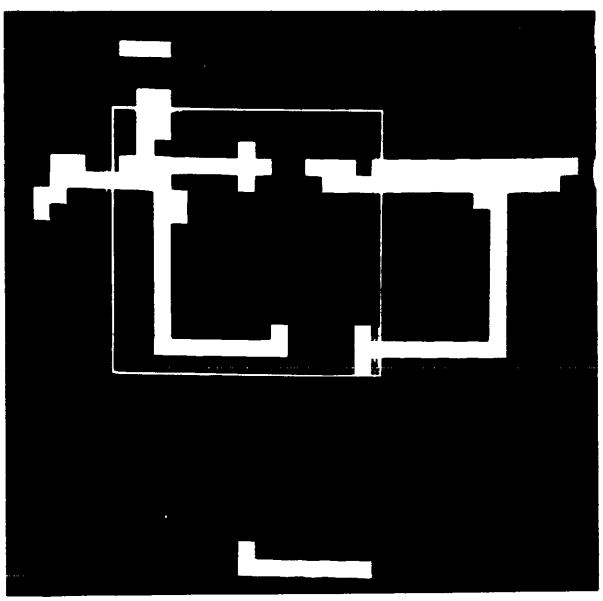


Figure 21

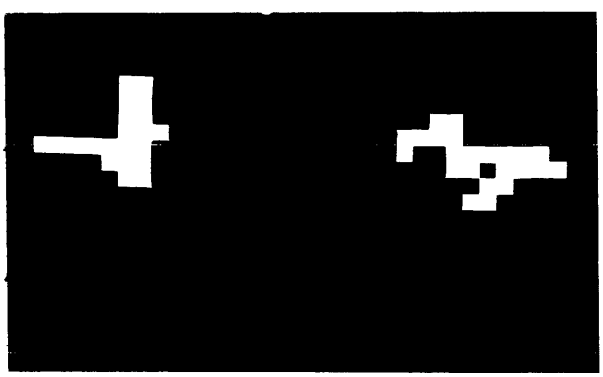


Figure 23

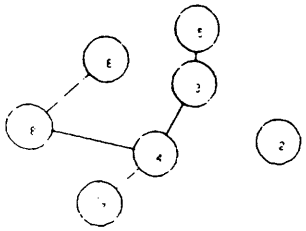


Figure 24

whichever. The procedures are extremely complex; the program represents an historical landmark in the development of artificial intelligence. However, consider the example of: pyramid is to figs as as chaletis to... This should alert us to a major difference between the geometric analogy and the "meaning" analogy between properties and attributes. It behooves us to ignore sometimes the formal counterparts and to recognize the simplest architectural intention, a tiny step beyond geometry. But we really do not know how to do it in baby steps. It is exemplary of the desperate problem of arriving at simple frontiers in artificial intelligence that appear to be extendable only in their most consummate form.

WHY BOTHER?

In contrast to the unenlightening, recursive argument of "so what," "why bother" can be a particularly instructive question in the context of computers and, in particular, in the light of their continuously dropping costs. Historically, a well-supervised parsimony with computing power has forced us to bend our manner of conversation and warp it into a man-machine communication characterized by trumped up, unnecessary level of consistency, completeness and precision. One is expected to be explicit and unequivocal with a computer.

Consider the previous example of recognizing whether a sketch is a plan or a section. The amount of code necessary to perform that task and the amount of ensuing computation is enormous. It might make a good programmer's doctoral's thesis and require five to ten seconds of fast computing to arrive at a reasonable conclusion. Would it not be easier to insist that the sketcher be required to exert the trivial additional effort of typing an "S" or "P" after completing his drawing? The answer is surely, Yes, it would be easier. The issue, however, is where to draw the line, even in the most timid, master-slave applications.

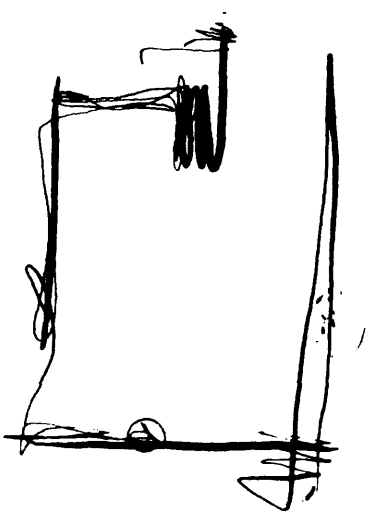


Figure 25

Figure 26

One extreme position is to adopt the SKETCHPAD explicitness: this is a line, this is its end, these two are parallel, this is an arc, and so forth. The other extreme is to consider all levels of communication as potentially as smooth, congenial, and free of explication as a conversation with a very intelligent, very good friend. We opt for the latter in toto on the following counts: (1) it is crippling to force an explicitness in contexts where the participant's equivocations are part of the function of design; (2) the tedium of overt, categorical exchange is counterproductive, unfulfilling for the speaker, and boring; (3) constructive and exciting responses are often generated by twists in meaning that result from the personal interpretation of intentions and implications; (4) finally, we view computer

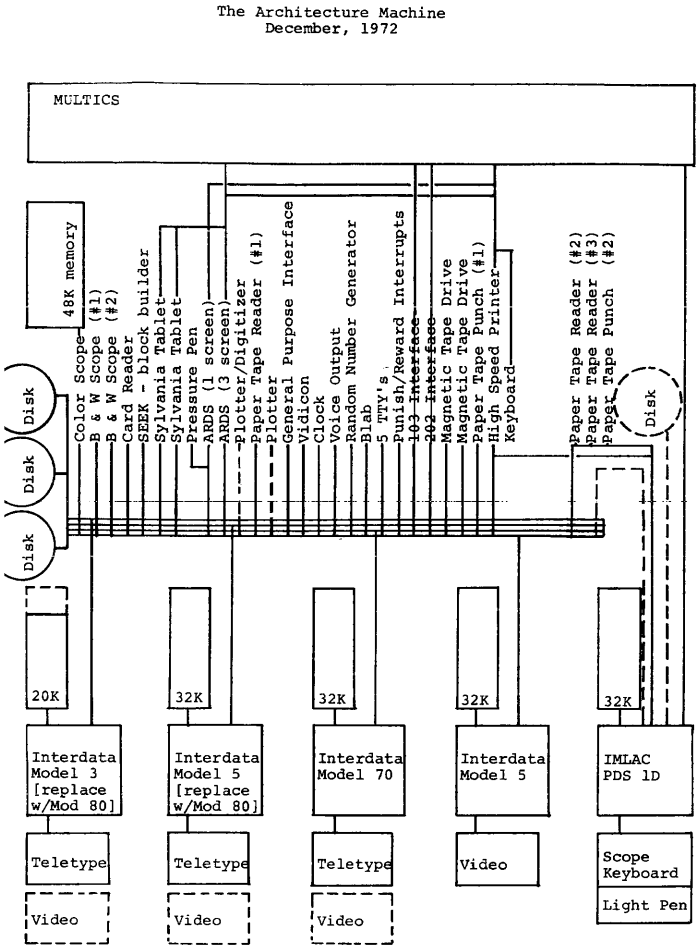


Figure 27

time as a free commodity to be allocated in the abundance necessary to make a rich dialogue, perhaps richer than we have ever had with another human.

REFERENCES

1. Alexander, Christopher, through personal conversation and reported by the DMG newsletter interview.
2. Weizenbaum, Joseph, through personal conversation and to be incorporated in a forthcoming volume in preparation at Stanford University.
3. Johnson, Timothy, "Sketchpad III: A Computer Program for Drawing in Three Dimensions," *Proceedings of AFIPS,* Spring 1963.
4. Negroponte, Nicholas, Taggart, James, "HUNCH—An Experiment in Sketch Recognition," in *Computer Graphics*, edited by W. Giloi, Berlif 1971.
5. Negroponte, Nicholas, *The Architecture Machine*, MIT Press, 1970.
6. Alexander, Christopher, through personal conversation and reported by the DMG newsletter interview.
7. Weizenbaum, Joseph, through personal conversation and to be incorporated in a forthcoming volume in preparation at Stanford University.
8. Negroponte, Nicholas, Taggart, James, "HUNCH—An Experiment in Sketch Recognition," in *EDRA Proceedings*, edited by William Mitchel, January 1972.
9. Negroponte, Nicholas, Taggart, James, "HUNCH—An Experiment in Sketch Recognition," in *Computer Graphics*, edited by W. Giloi, Berlif 1971.
10. Gibson, J. J., *The Perception of the Visual World*, Houghton Mifflin, Boston 1951.
11. Gibson, J. J., *The Senses Considered as Perceptual Systems*, Houghton Mifflin, Boston 1966.
12. Gregory, R. L., Manuscript presently before MIT Press. The manuscript is a collection of all his essays and articles since graduation.
13. Selfridge, Oliver, Neisser, U., "Pattern Recognition by Machine," in *Computers and Thought*, edited by E. A. Feigenbaum and J. Feldman, McGraw-Hill, New York 1963.
14. Minsky, Marvin, Papert, Seymour, *Perceptions*, MIT Press, 1970.
15. Guzman-Arenas, Adolfo, *Computer Recognition of Three-Dimensional Objects in a Visual Scene*, Project MAC Memorandum MAC-TR-59, MIT 1968.
16. Roberts, L. G., "Machine Perception of Three-Dimensional Solids," in *Optical and Electro-optical Information Processing*, edited by J. T. Tippett et al., MIT Press 1965.
17. Kubert, B., J. Szabo, Guilieri, S., "The Perspective Representation of Functions of Two Variables," *Journal of the ACM*, April 1968.
18. Galimberti, R., Montanari, U., "An Algorithm for Hidden Line Elimination," *Communications of the ACM*, April 1969.
19. Loutrel, P., "A Solution to the Hidden Line Problem for Computer-Drawn Polyhedra," *IEEE Transaction on Computers*, March 1970.
20. Ricci, A., *An Algorithm for the Removal of Hidden Lines in 3D Scenes*, Comitato Nazionale Energia Nucleare Centrodi Calcolo, Bologia, Italy, November 1970.
21. Encarnacao, J. L., "A Survey of and New Solutions to the Hidden Line Problem," in the *Proceedings of the Interactive Computer Graphics Conference*, Delft, Holland, October 1970.
22. Ejiri, Masakazu, Uno, Takeshi, Yoda, Hauro, Goto, Tatsuo, Takeyasu, Kiyoo, "An Intelligent Robot with Cognition and Decision-Making Ability," in the *Proceedings of the Second International Conference on Artificial Intelligence*, London, England, 1971.
23. Simon, Herbert, *The Sciences of The Artificial*, MIT Press, 1969.
24. Pask, Gordon, "The Architectural Relevance of Cybernetics," *Architectural Design*, September 1969.
25. Evans, Thomas, "Analogy" described in a chapter of *Semantic Information Processing*, edited by Marvin Minsky, MIT Press, 1968.

Interactive graphics for computer aided network design

BY J. L. FRANKLIN and E. B. DEAN

Naval Ordnance Laboratory
White Oak, Silver Spring, Maryland

INTRODUCTION

Computer analysis codes for electronic networks have been in existence since about 1961. In spite of this, they have failed to generate much impact on electronics design and analysis. One of the main reasons for this is that most engineers do not have either a background in computer coding or the time (approximately two concentrated weeks) to learn the languages associated with each code.

Another drawback of computer aided circuit analysis is that it is limited by the turn-around time of the computer facility. This, coupled with the fact that a large number of coding errors are made, slows down a potentially very fast analysis technique.

As an illustration of the problems associated with batch operation of the codes, these steps are associated with their usage:

(1) A schematic is coded into the computer language associated with the code being used.
(2) Cards are punched.
(3) The job is submitted to be run by the computer.
(4) Keypunch errors listed in the output are corrected.
(5) If there are coding errors listed in the output, a schematic must be drawn from the coding and checked against the original schematic.

Steps (2) through (5) are then repeated until all the errors are removed and the code gives the circuit response requested. From this point on, the job is repeatedly resubmitted with different output requests, different values of the circuit elements, or an altered circuit topology until a complete analysis of the circuit is obtained.

As a result of this process and of the low acceptance of the codes, it was decided that a prototype computer graphics input and output system should be developed. The objective of graphics would be to divorce the use of the analysis codes from traditional language programming. The only operation the user would have to perform would be to draw the circuit on the face of a cathode ray tube (crt). This would eliminate the excessive time required to learn the circuit languages, to acquire programming skills, to become proficient at keypunching, and to redraw the circuit from the code. The output from the circuit would be displayed on the crt, i.e., there would be essentially zero turn-around time for a job. Thus, the user could perform any changes he desires and see the results immediately. Whereas with a batch job a complete analysis would take days or weeks; with a computer graphics analysis it would take only hours.

DEVELOPMENT OF THE SOFTWARE

All the circuit analysis codes have their own input languages to describe the topology and values of the circuit. In general, inputs to these languages are the circuit element names, node numbers for nodal connection points, and element values. Once the circuit has been described by the circuit analysis language, the analysis program will give the response of the circuit. SCEPTRE, NET-2, CIRCUS, AEDCAP, SPICE, and ECAP are among the main codes in use today for this purpose.

The graphics approach is to use the terminal as a translator for the circuit analysis language, i.e., to use a graphical symbolic language the engineer is familiar with in place of an alphanumeric language.

The first system was developed with the philosophy of maximum user flexibility. Thus, the user had control of the names, nodes, and values of each circuit element (see Figure 1). This proved to be too difficult to learn to use for people with no familiarity with the circuit codes. Also, it was time consuming and not that great a benefit for the sophisticated user. In the next version, user flexibility was traded off for simplified user operation. The only inputs required for this version were the values of the circuit element (Figure 2).

This improved the system's usability, but it still had other drawbacks. The most serious of these was the inability to rotate circuit elements. Element rotation had not been included in the first version because it was reasoned that the circuit could be redrawn for the orientation of the graphics elements. However, in practice, it was found that: (1) circuit diagrams were usually optimally oriented and redrawing made them overly complicated; (2) mistakes were made in redrawing the diagram to agree with the fixed graphic element orientation,

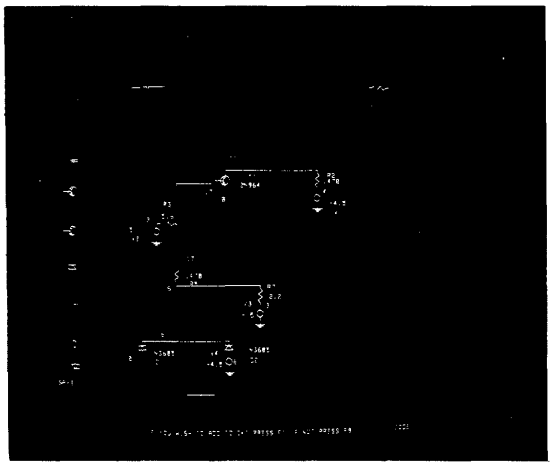


Figure 1—First version display

and one could never be confident that the redrawn circuit was 100 percent correct; (3) redrawing accurately was a time-consuming operation. Besides nonrotating elements, the original version had other drawbacks which were more of an inconvenience than a source of error. These were the inability to edit (make corrections) on the circuit via a graphical operation, and the inability to request the circuit output via a graphical operation. In this version, editing and output requests were implemented via the circuit analysis code. This method of operation necessitated the user to know some circuit analysis coding. Another drawback was that learning to use the system was difficult because of the user instructions. These instructions told the user which operations he could perform next. The problem with the set used with the second version was that they made the operator choose between more than two options. This turned out to be too complicated for the novice to resolve quickly.

Further use indicated the need for "fail-safe" software. By this is meant that if the graphics user chooses an unlisted option or happens to accidentally hit the keyboard or light pen, the program would not "clobber" itself. Also, and probably more important, we began to have the unhappy experience of putting up a large circuit and having the main computer crash. This sometimes resulted in the loss of one or two hours worth of circuit drawing.

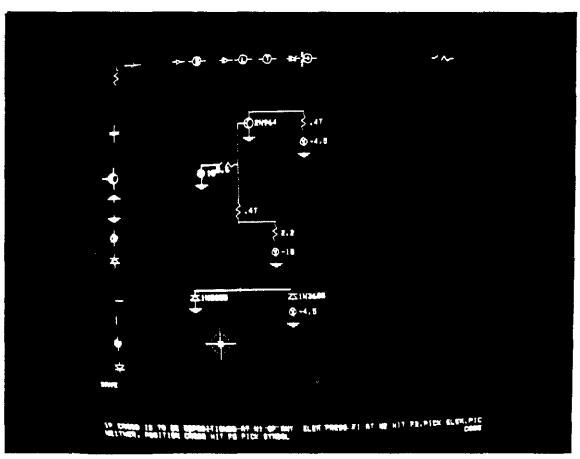


Figure 2—Second version display

The current graphics version (Figure 3) has graphical editing, rotation of elements, and graphically requested circuit output. In addition, alphanumerics (such as element values) are not displayed on the screen unless requested. This latter feature was implemented because it was found that the alphanumerics associated with each circuit element take up more space than the circuit diagram itself. The third version is written from a "fail-safe" viewpoint, and in order to minimize the loss due to a main frame computer crash, the circuit and its associated information are saved on tape with every five elements drawn. Thus a main frame crash can only cost you, at most, a five element loss while drawing the circuit.

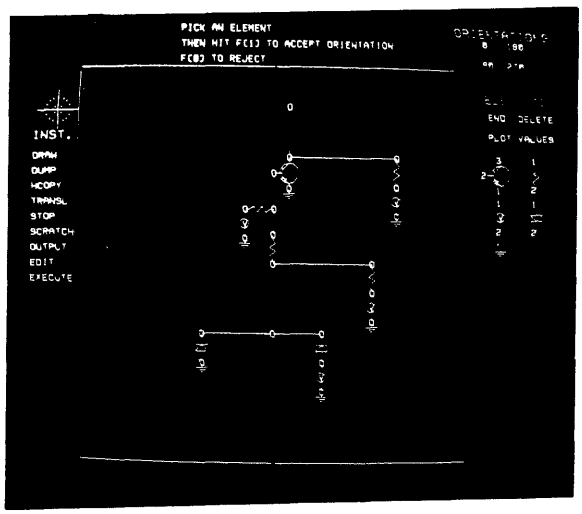


Figure 3—Third version display

Even though we have strived to maximize the graphical operations in this version, we still have retained a circuit analysis (alphanumeric language) text editor. Experience has taught us that there will always be times when the advanced circuit analysis code user will want to augment his work with the code.

At this point a five-minute film showing the operation of the system will be shown. In lieu of the film Figures a through l are provided. On the right side of the screen, the menu which contains the choices of elements from which the user may build his circuit is shown. To build the circuit, the user positions the tracking cross and with the light pen picks the appropriate menu symbol to bring up the element at the position of the tracking cross. To pick up succeeding elements the process is repeated. Elements can be joined by lines of arbitrary length. At any time, the user can change the topology of the circuit drawn (i.e., edit). A hardcopy can be made of what is on

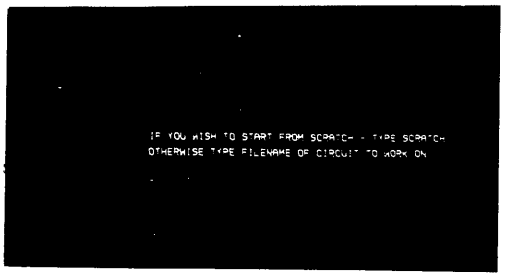


Figure a—Third version display

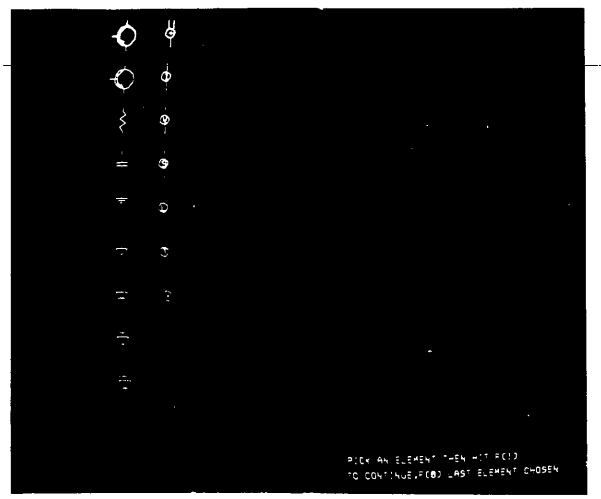


Figure b—Menu of available circuit elements

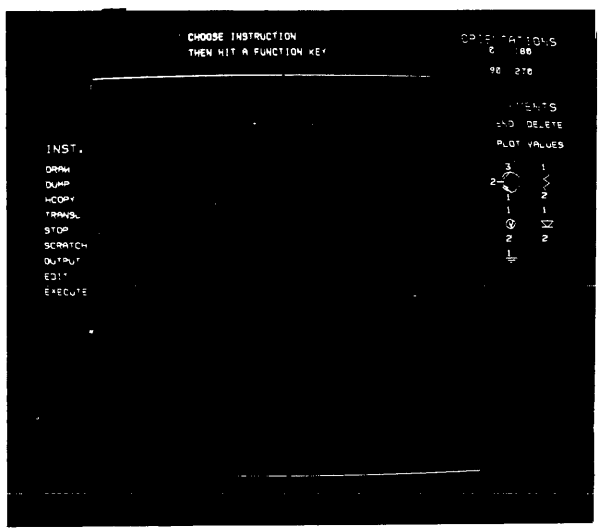


Figure c—Third version display

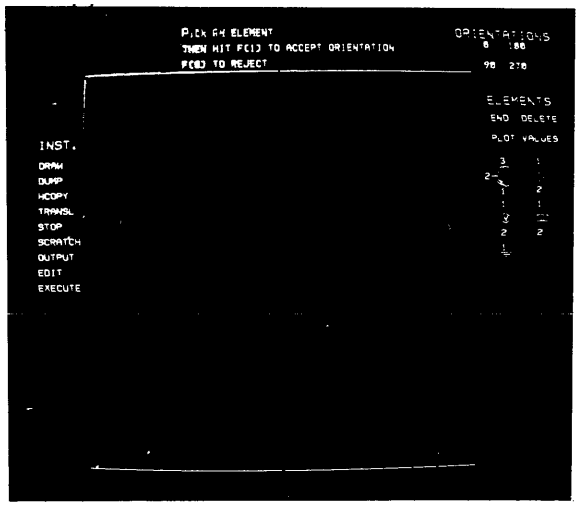


Figure d—Third version display

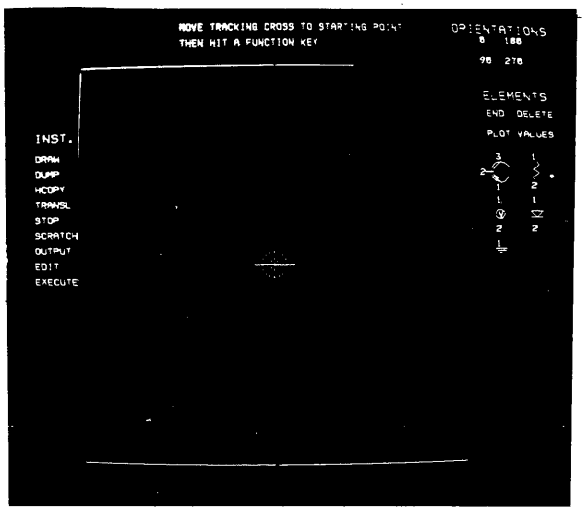


Figure e—Third version display

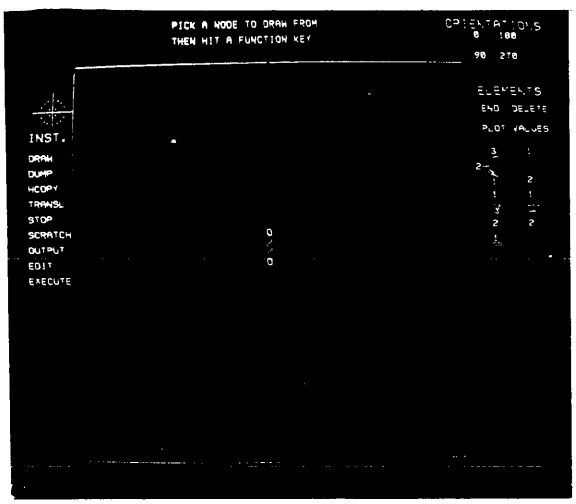


Figure f—Third version display

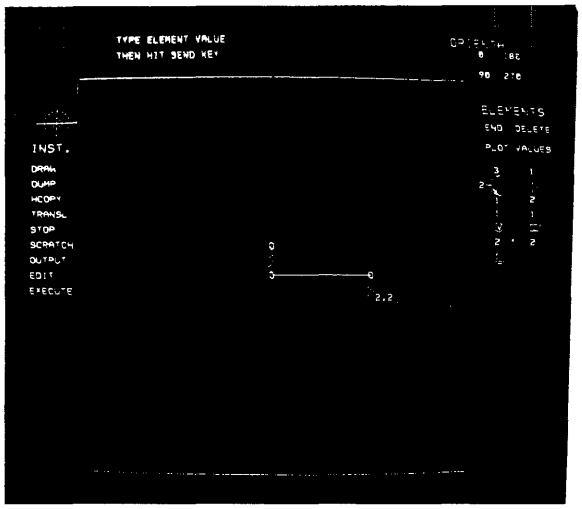


Figure g—Third version display

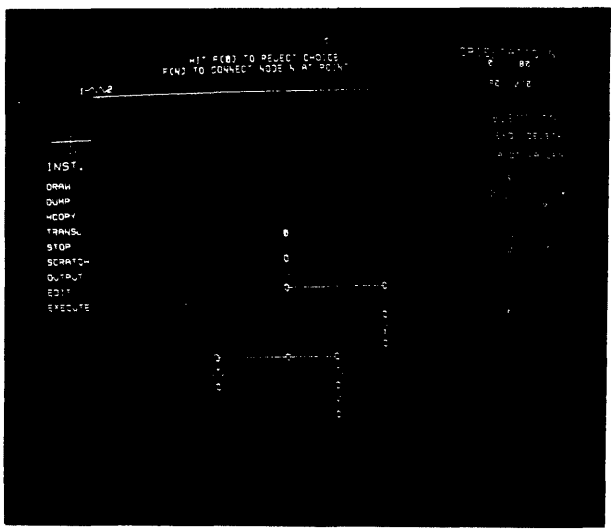


Figure h—Third version display

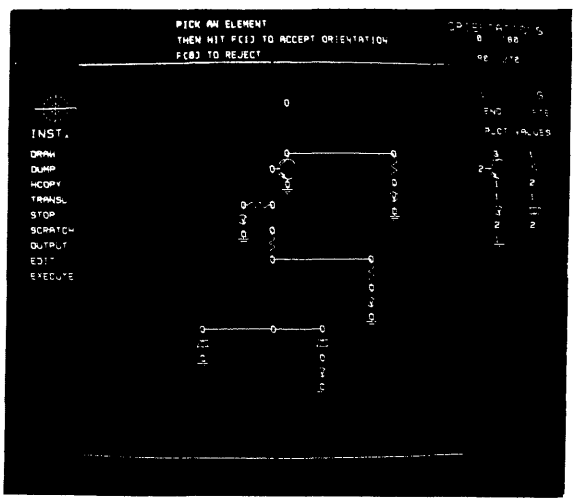


Figure i—Third version display

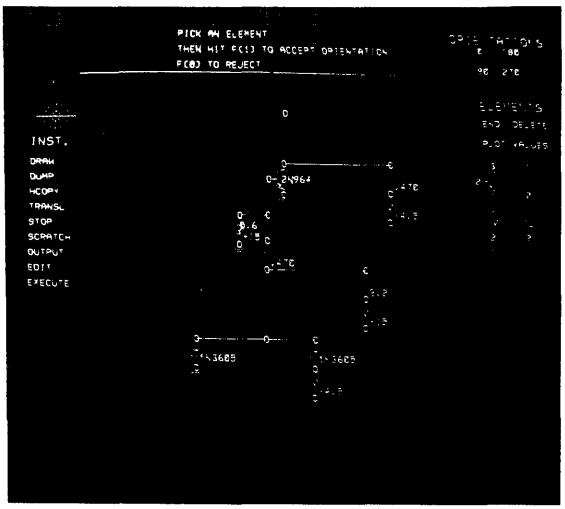


Figure j—Third version display

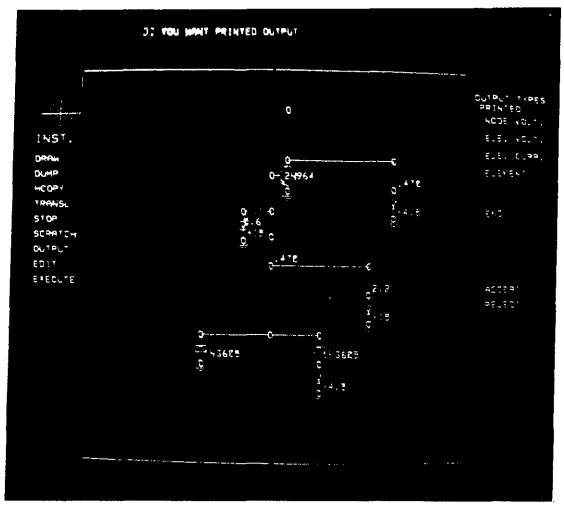


Figure k—Output request display

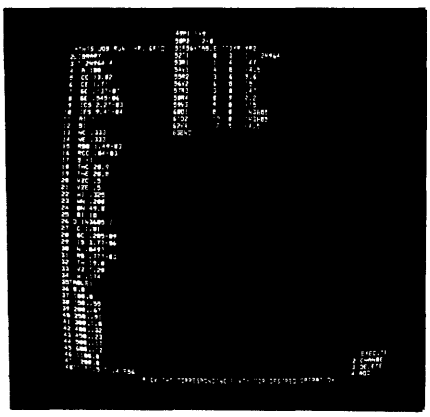


Figure l—Alphanumeric code editor

the crt by picking "HCOPY" with the light pen. Figure *m* shows a hardcopy of the circuit drawn. Once the circuit has been drawn, the output desired by the user is requested by picking the circuit nodes for node voltages or the circuit elements for element currents. The response of the circuit is returned to the graphics (Figure *n*). The user can redisplay his circuit for further changes i.e., to continue his analysis.

IMPLICATIONS AND EXTENSIONS OF COMPUTER AIDED NETWORK DESIGN GRAPHICS SYSTEMS

It became evident as development proceeded that the graphics system was more than a visual input-output device. In fact, it is a high level compiler of the natural language of the engineer. The half dozen circuit analysis codes currently in use could all be handled with almost identical graphics software. The exceptions would be

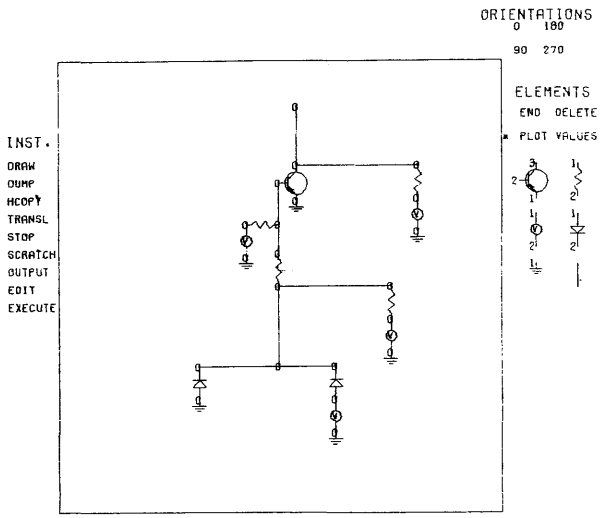


Figure m—Hardcopy of third version display

FORTRAN format statement changes to compensate for the idiosyncracies of the different languages. The user-engineer does not have to worry about learning the languages of the various programs. Instead of having to train people in a new language, only the person who maintains the graphics software need learn it. Thus the compiler aspect of graphics affects a reduction in the training cost, training time, and the trauma of having to learn many new languages.

But the graphics is more than a compiler, it is a natural data manager. This became evident when it was decided to implement a feature called "nesting" which was available in one of the circuit analysis codes. Nesting begins by representing a subcircuit by a symbol. Whenever this symbol is used in another circuit it is like connecting the subcircuit into the new circuit. This new circuit can be defined by another new symbol and used in another main circuit. Figure 4 shows the nesting process. A nest of level one is commonly referred to as a model.

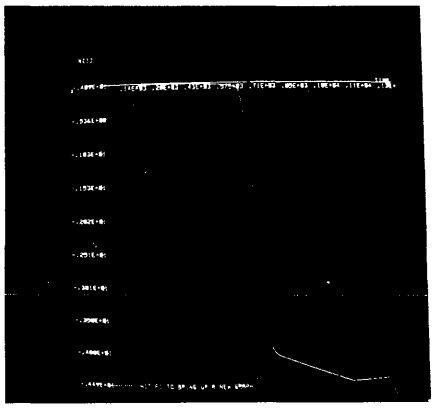


Figure n—Response of circuit displayed on graphics

The graphical implementation of nesting is merely to draw a circuit and define a menu symbol for it. This menu symbol then becomes available to be used in drawing other circuits. The user can exercise an option to see and/or alter the circuit represented by his symbol. It is

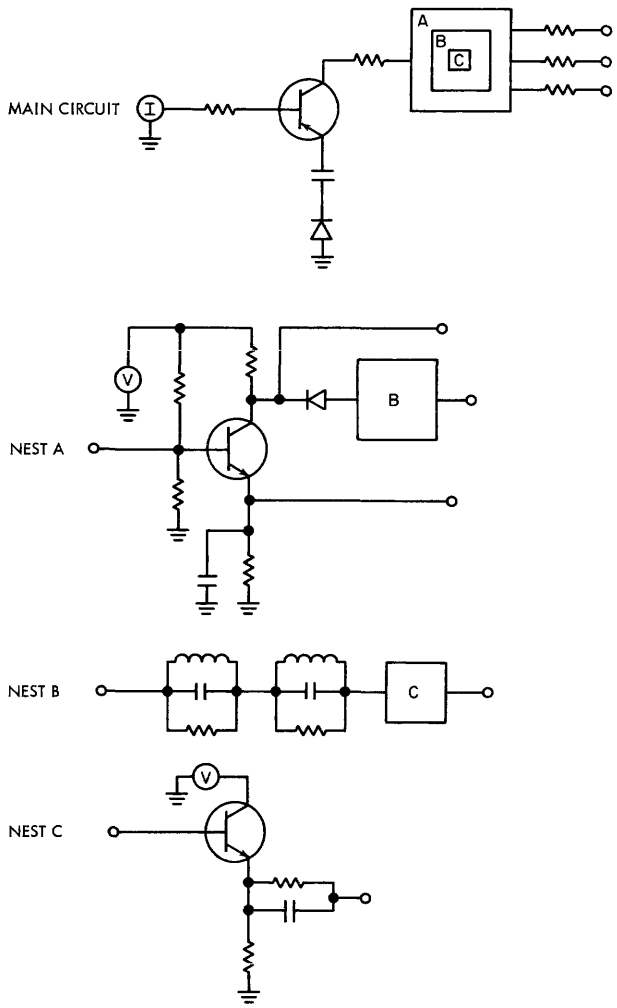


Figure 4—Nesting

```
DCAMRSS,CM150000,T100,P2.54301770,624,SHAKLAN.
ATTACH(NETTWO,NET199)
NETTWO.
'
*TEST
DEFINE MET VIN RMS
   MULT1 1 VIN VIN
   FILT1 1 3
   SUM2 4 3 -6
   GAIN1 4 6 -1+7
   ABV1 7 RMS
   MULT2 6 RMS RMS
DEFINE FILT IN OUT
   LPAS1 IN 1
   LPAS2 1 2
   LPAS3 2 3
   LPAS4 3 4
   LPAS5 4 5
   LPAS6 5 6
   LPAS7 6 7
   LPAS8 7 8
   LPAS9 8 9
   LPAS10 9 10
   LPAS11 10 11
   LPAS12 11 12
   LPAS13 12 13
   LPAS14 13 14
   LPAS15 14 15
   LPAS16 15 16
   LPAS17 16 17
   LPAS18 17 18
   LPAS19 18 19
   LPAS20 19 20
   LPAS21 20 OUT
DEFINE LOWP IN OUT
   LPAS1 IN 1
   LPAS2 1 2
   LPAS3 2 3
   LPAS4 3 4
   LPAS5 4 5
   LPAS6 5 6
   LPAS7 6 7
   LPAS8 7 8
   LPAS9 8 9
   LPAS10 9 10
   LPAS11 10 OUT
DEFINE LPAS IN OUT
   SUM1 1 IN =OUT
DEFINE ABV IN OUT
   TABF1 TABLE1 OUT IN
   TABLE1
      -1.+7 1.+7
      0. 0.
      1.+7 1.+7
PARAMETER
   MET3.FILT1.LPAS1.INT1 .628
   MET3.FILT1.LPAS2.INT1 .628
   MET3.FILT1.LPAS3.INT1 .628
   MET3.FILT1.LPAS4.INT1 .628
   MET3.FILT1.LPAS6.INT1 .628
   MET3.FILT1.LPAS7.INT1 .628
SUM1 5 4 -6
INT1 5 6 P2
P2 .0314
RNGEN1 3 1
GAIN1 3 4 P1
P1 .001
MET2 4 7
MET3 6 9
LOWP1 7 10
LOWP2 9 11
STATE1
   TIME 0 (500) 10
   PLOT N(4) N(7)
   PLOT N(6) N(9)
END
```

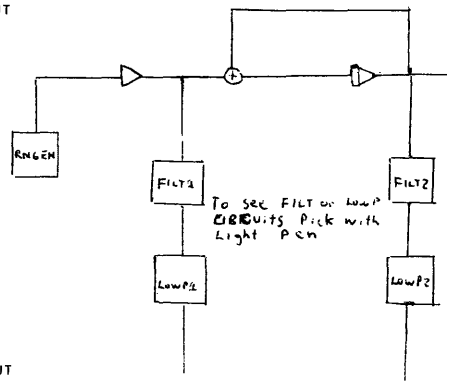


Figure 5—Graphical versus alphanumeric formulation of the same nesting problem

obvious that by using the graphics to keep track of the nesting level rather than tracing it back through some symbolic circuit analysis language is a great aid with large circuits. Figure 5 shows the difference in complexity using graphics versus the symbolic language for the same nesting problem.

It was stated that a nesting of level one is generally referred to as a model. Modeling is where the symbol is related to the actual electronic device. The device is measured for certain parameters that represent the physics of the model. Here at NOL a computer controlled model parameter measurement system is being developed. To get the parameters, curves are fitted to the measured data and displayed on the graphics. If the fit is not good, the user interacts with the data via the graphics to produce a good fit. These model data go into a library on disk and can be recalled to the graphics terminal at any time.

The complete system is shown in Figure 6. As can be seen, the graphics acts as the center of the measurement system, the libraries, and the analysis program. In effect, Figure 6 demonstrated that the graphics-man interface is the data manager of the complete system. As an example consider the operations performed by the graphics from management of raw data to the final analytical results of the circuit simulation.

The engineer receives a new circuit. First he displays a list of the devices on which data have been stored. For the devices in the circuit that have not been characterized, he measures the devices, displays the raw data, and fits curves via the graphics. Through a series of iterations, by adjusting the fitting routines and viewing the results, he gets satisfactory fits and thus acceptable model parameters. For the devices in the circuit that have been previously measured, he simply calls these parameters from the disk library. The engineer then draws the circuit on the graphics and gets the response of the circuit. Let us say that for some reason the circuit analysis code does not work. One solution to the problem would be to try another circuit analysis code. Since for the graphics circuit compiler the only difference between circuit codes is a change in FORTRAN format statements, it is a simple matter to utilize the circuit drawn as input to another code. Thus, the engineer merely calls up another circuit analysis code from the graphics and runs the problem again. If he wishes to change the topology of the models associated with his stored data, he simply calls up his models and

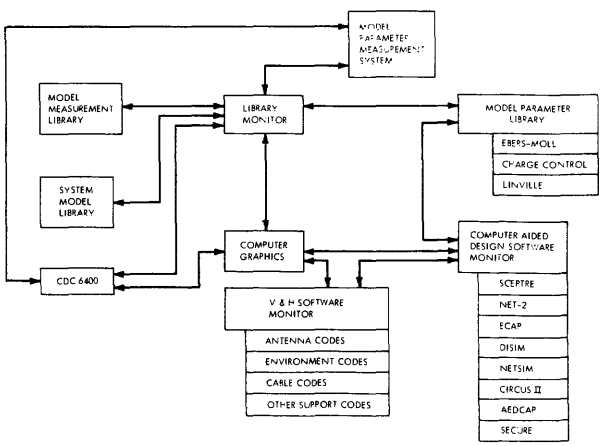


Figure 6—The Naval Ordnance Laboratory model measurement and data acquisition system monitored by graphics

redraws them on the graphics. After he is finished with the problem he stores his circuit. At a later date he can redisplay his circuit and immediately begin again where he left off.

Solving a circuit response is mathematically equivalent to solving a system of nonlinear differential equations of N independent variables. Since there are many physical systems described by such a set of equations, the applicability of the graphics as a general compiler goes far beyond circuit analysis. For example, substituting the mechanical analogs for their electrical equivalents such as mass resistance, force for voltage, etc., gives a graphical mechanical language for a minimum amount of software change.

At NOL, we are moving toward this goal by utilizing the system, or analog, capabilities of the NET-2 circuit code to solve regular and partial differential equations of mechanical, electrical, chemical, and mathematical systems. Although we have used the analog symbols in our graphics solutions, it would be a simple matter to have substituted the appropriate physical or mathematical symbols as part of the menu. If these symbols had been added instead of formulating the problem in electrical schematic, it could have been formulated in the natural symbols of the physical system. Figure 7 is an example of a mechanical problem that was run using NET-2. It shows the mechanical, electrical, and mathematical equivalent formulations. Also, it can be seen from Figure 7 that to formulate a spring, the graphics would need only write the code for a gain and a summer which is a simple format exercise. Similar analogies hold for the dashpot and the masses.

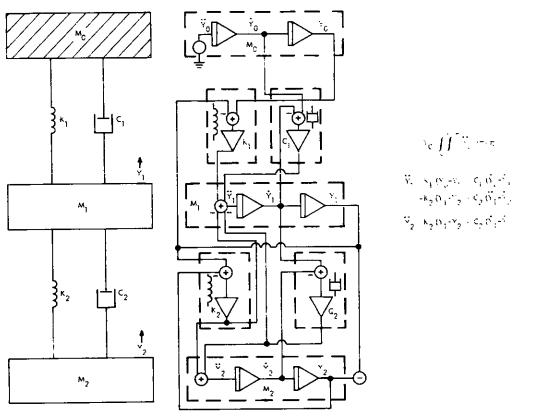


Figure 7—Equivalent formulations of problems that can be run on graphics

A facit of graphics implicit in our program but not yet developed to its fullest is self-instruction. The instructions that tell the user which optimal he can exercise next essentially teach the user how to operate the graphics program and the circuit analysis codes. In fact, graphics systems have inherent self-instruction capabilities.

The final and possibly the most important advantage of a graphics controled system is that it allows the man to interact with the computer in a natural and efficient way. This interaction allows the level of software complexity to be kept to a minimum. For example, with Newton Raphson convergence algorithms, it is much easier for the user to pick trial solutions for a more optional convergence than to have to write software to do the same job. The interactive capability cuts the cost and frustration of the analysis. Instead of having to run a long involved parametric study to see the way a system is responding, the user merely watches the results to see which analysis variables he need adjust (via the graphics). In short, graphics reduces the number of iterations on a trial-and-error type analysis problem by having the user act as part of the feedback loop.

It has been our experience that this feedback loop property which improves temporal efficiency also improves psychological efficiency. The user sees a result, acts on it, and sees the result of that, etc. He finds his analysis going faster because he can maintain a continuity of thought. This, coupled with being able to manipulate a naturally familar language (the schematic, etc.), acts as a catalyst for insight and efficiency.

In conclusion, at NOL, our original goal was to write a graphics front and rear end to our circuit analysis programs. However, in the process of accomplishing this we came to the realization that we had found more than just a new input/output medium. Maybe it is as one contemporary thinker has said, "and the medium really is the message."

Sorting and the hidden-surface problem

by IVAN E. SUTHERLAND,* ROBERT F. SPROULL,** and ROBERT A. SCHUMACKER*

Evans & Sutherland Computer Corp.
Salt Lake City, Utah

INTRODUCTION

Ten years ago the task of producing hidden-surface pictures by computer seemed untractable. Courageous men pressed forward nonetheless, and today we can have quite beautiful renderings of solid objects generated by computer in remarkably short times. The pictures being produced today and the speed of the programs producing them are beyond any but the wildest dreams of ten years ago.

In December 1972 we embarked on a formal study of ten hidden-surface algorithms, hoping to develop a systematic understanding of how they worked. We felt that if we could find a framework into which to fit the various programs, we could identify empty branches of that framework and thus discover approaches to the task which might be new. If we could understand some basic principle which all of the algorithms apply, we might gain some fundamental understanding of the inherent cost of rendering solid objects, and thus some measure of the "ultimate" performance one might expect in such an algorithm.

Two underlying principles have emerged from our study. The first of these we had earlier supposed to be important,[14] namely sorting. All of the algorithms *sort* through collections of surfaces, edges, or objects according to various criteria, finally discovering the one visible item and displaying it. Although the order and kind of sorting used differ, our original supposition that sorting is the key to the task seems amply justified.

The second underlying principle that emerges from our study is coherence. The environments rendered by the hidden-surface algorithms consist of objects with more or less flat surfaces and straight edges rather than random discontinuities. This *coherence* of the environments being rendered limits how different the picture can be from place to place or from time to time. All of the algorithms capitalize on various forms of coherence to reduce the work of sorting to manageable proportions. The kinds of coherence most helpful to particular algorithms are easily identified; to what extent useable coherence exists in a particular solid object seems to determine to a great extent the speed with which the algorithm will render it.

By applying these two principles we have constructed a framework into which to fit the algorithms. The framework is expressed as a tree whose branch points represent different choices of sorting order, and thus different estimates of the importance of various kinds of coherence (see Figure 1). The framework seems comfortable in the sense that we have been able to fit everything into it nicely; programs that seem similar in some intuitive sense appear on nearby nodes of the tree; and the framework seems to have room for as yet uninvented algorithms. Indeed, the framework suggests some promising new algorithms that should be tried.

SORTING

Sorting is made up of a sequence of simpler operations which we call "searching," "culling," and "merging." A *search* operation identifies exactly one element of an input set which has some special property. A *culling* operation removes elements from an input set which fail to have some special property. A *merge* adds a new element to an input set to create an output set.

A *sort* permutes an input set to produce an output set ordered according to some property. The output set may be an ordered table, an ordered list, or a tree. Although the techniques of sorting are well understood, it is important to distinguish a few special types of sorting particularly well suited to some of the operations which a hidden-surface algorithm must perform.

A *bubble* sort interchanges adjacent elements of the unsorted list to achieve ordering. The cost of a bubble sort increases with the square of the number of elements in the list to be sorted unless the elements are already nearly in order. In this case, a single pass through the list and a few interchanges will produce a correct ordering. Bubble sorting is particularly useful for sorting the X intercept positions of edges in the picture as a TV scan progresses because the order in which such edges cross a scan-line is nearly the same as for the previous scan-line.

A *bucket* sort is a sort in which a large number of "buckets" are used to collect elements with similar prop-

* Evans & Sutherland Computer Corporation—Salt Lake City
** Stanford University; formerly with Evans & Sutherland

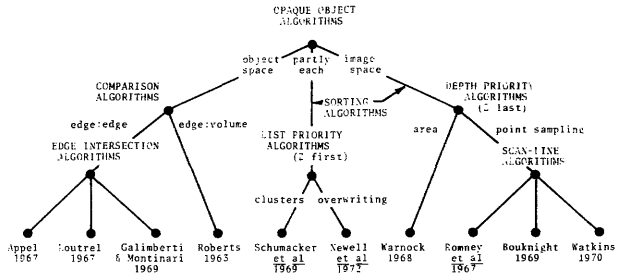


Figure 1—A comparison of ten opaque-object algorithms

erties. For example, there could be one bucket for each scan-line in a TV picture, and edges might be placed in these buckets according to the scan-line in which they first appear. A bucket sort requires only a single pass through the data to put each element in its correct bucket.

A *quicksort* may be constructed in any one of a number of ways which depend on successive division of the input set into more and more nearly sorted outputs. Quicksorts require somewhat more time than bucket sorts and somewhat less than bubble sorts. Randomly distributed data will require $n*\log_r m$ tests, where n is the length of the list to be sorted, m is the range of the key-field, and r is the number of sublists into which the input list is divided during each pass. The bucket sort can be thought of as a quicksort where $r=m$ and thus $n*\log_r m=n$.

COORDINATE SYSTEMS

We are treating all of the algorithms as if they used the same coordinate systems even though some of them do not. Our approach is based on the notion of perspective projection as a transformation of the original three-dimensional coordinate system (see References 10, 11, and 15.

The transformation from object to screen coordinates preserves straight lines, computes "perspective" so that the X and Y positions of each point are correct on the screen, and preserves depth ordering in the Z coordinate. The use of three, rather than two-dimensional screen coordinates simplifies the otherwise difficult visibility computations which appear in every hidden-surface algorithm.

Because the transformation from real space into a perspective picture must necessarily include division, there is always the possibility of overflow. Because of this we uniformly think in terms of a pre-processing cull which we call *clipping* which will remove from consideration those faces and edges which lie outside of the field of view.[15] It is also common to eliminate all faces which face away from the observer and are thus obviously not visible. These preliminary culling operations require computations which grow linearly with the complexity of the environment.

THE ENVIRONMENT

A hidden-surface algorithm makes a two-dimensional picture of a collection of three-dimensional objects and surfaces which we call the environment. Different algorithms require different forms of internal description for the environment, and place different restrictions on it. We have translated the terms actually used by various authors to relate how they define environments into standard terms defined in this section.

One should be alert to differences in the topological properties of the different environment descriptions. Some algorithms need to know which surfaces meet at a particular edge, while others make no use of such information. Similarly, some authors make use of groupings of faces into objects or clusters while others simply treat faces independently. The difficulty of building environment models increases with the amount of such topological information required by the algorithm, but the algorithm may profit immensely from the availability of such additional information.

The algorithms considered in this paper deal only with plane-faced objects; we have not considered algorithms for dealing with curved surfaces. Thus by the *face* of an object we mean a closed polygon composed of straight *edges* which connect a number of *vertices*. We assume that the vertices of each face lie in a plane, although some of the algorithms are able to handle non-planar or *skew* faces.

A *cluster* is a collection of faces that can be treated as a group for some special reason. Schumacker,[6] who introduced the concept, associates faces as a cluster provided that a preset priority order of visibility can be established for them independent of the viewpoint. Two clusters are *linearly separable* if a plane can be passed between them.

The statistical behavior of the following picture quantities is important in evaluating the sorting strategies used

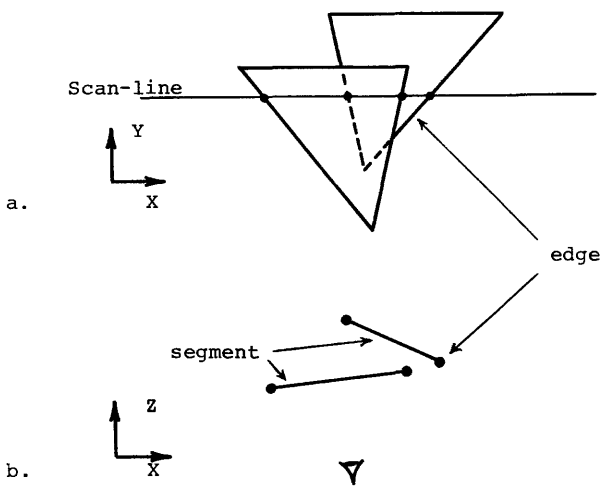


Figure 2—A scan-line intersects faces in segments. (a) The view on the screen, (b) The view in $X=Z$ space, showing the depth relationship between the segments.

in the algorithms. A face or edge is *relevant* if it survives an initial clipping or back facing cull. The *depth complexity* of an environment is a measure of how many relevant faces are pierced, on the average, by an arbitrary ray from the viewpoint. Watkins[18] measured the depth complexity for various environments and found it to be remarkably low, between 1 and 3. A *segment* is the straight line segment defined by the intersection of a face and the plane which contains both the scan line and the viewpoint. Segments are often illustrated in the *XZ* plane as shown in Figure 2b so that their depth relations may be seen clearly.

DESCRIPTION OF THE ALGORITHMS

In this section we shall briefly describe the major differences between the opaque-object algorithms included in Figure 1. A more complete summary of the algorithms may be found in Reference 17 and, of course, the published works of the authors listed in the bibliography.

We have chosen to categorize algorithms into two basic classes: object-space algorithms and image-space algorithms, although some algorithms fit partly in each class. This classification, quite coincidentally, separates the hidden-*line* from the hidden-*surface* algorithms. The object-space algorithms seek to compute "exactly" what the image should be by discovering what parts of the object are hidden by other parts. The image space algorithms, on the other hand, seek to compute what the image will be only at each of the resolvable dots on the display screen.

Object-space algorithms

Among the object-space algorithms, we can identify a further division. Although all of these algorithms test relevant edges to determine what parts of the edges are visible, the invisibility criteria are quite different. In the Roberts algorithm, an edge may be obscured by an object that lies between the edge and the viewpoint. The algorithm thus capitalizes on the spatial coherence of objects: it tests edges against objects. Roberts observed that a convex object could, at most, break an edge into two visible segments. By computing the locations along the line at which the edge is first obscured by an object and last obscured by an object, he discovers which segments of the edge are visible.

The algorithms of Appel, Galimberti & Montinari, and Loutrel, on the other hand, use quite another form of coherence. As Appel first observed, the invisibility of an edge can change only at places in the picture where another edge crosses it. Moreover, edges which share a common vertex have a common invisibility at the vertex. As shown in Figure 3, these algorithms work outward along a network of edges from some initial vertex whose invisibility has previously been computed, incrementally

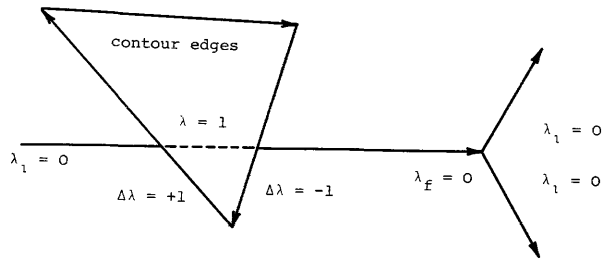


Figure 3—Computation of the quantitative invisibility ( ). The invisibility of the initial vertex, λ_l, is computed. The quantitative invisibility on the edge changes only where the images of the relevant edge and contour edges intersect and the contour edges are closer to the viewpoint. The final invisibility, λ_f, is used as the initial invisibility of other edges emanting from the vertex

changing the invisibility of each edge each time it crosses another edge. Appel coined the term "quantitative invisibility" to describe the property which is incrementally changed. Quantitative invisibility is essentially a count of the number of surfaces between a particular point and the observer. The quantitative invisibility of the initial vertex is calculated by an exhaustive search of all relevant faces to count how many faces hide the vertex.

Although some minor distinctions can be made between the Appel, Galimberti & Montinari, and the Loutrel algorithms, they all share with Roberts the requirement that each object be compared against all other objects in the environment. Thus the cost of these computations grows as the square of the number of objects in the environment, and these algorithms become impractical for very large environments.

Image-space algorithms

Historically, the pure image-space algorithms follow from work started in 1967 at the University of Utah by D.C. Evans. Evans understood clearly from the start the importance of the limited resolution of image space and the need for incremental computation during TV scan. The Utah efforts produced a series of interesting algorithms, the most recent of which is a real-time algorithm by G. S. Watkins that is now commercially available in hardware.

The partly image-space algorithms of Schumacker and Newell, on the other hand, appear to be independent starts. The earlier work of Schumacker and collaborators was begun at General Electric in 1965. Their goal was to develop a high-quality image-presentation system for use in visual flight simulation. Their work culminated in the delivery and later enhancement of a system for NASA's Manned Spacecraft Center. This was the first real-time solution to the hidden-surface problem and has been operational since 1968.

The more recent algorithm of Newell *et al.* is similar to the Schumaker algorithm in basic principle, but both are based on quite different notions from those used in the

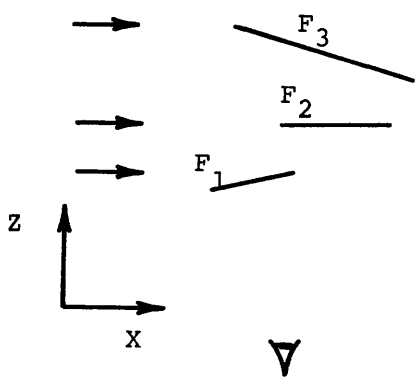


Figure 4—Z sort to determine priority order. If the faces are sorted by furthest vertex from the viewpoint (arrows), the correct order F_1, F_2, F_3 is produced.

pure image-space algorithms. In the image-space algorithms the visibility test is postponed until last and comes about as a computation of the depth of the various surfaces that would be penetrated by a viewing ray at a particular point in the image. Thus these algorithms can capitalize on the lateral separation of the image to reduce the number of depth computations required. The *list priority* algorithms, on the other hand, precompute in object space a linear visibility ordering or "priority" for all surfaces before generating the picture in image space; if ever two surfaces conflict, the one with the higher priority is the visible one.

List priority algorithms

The most significant difference between the algorithms of Schumacker *et al.* and that of Newell *et al.* concerns the way in which the priority list is computed and used. Newell makes use of a priority sorting algorithm to place the faces into priority orders as shown in Figures 4 and 5. Newell's sort will sometimes have to divide faces as shown in Figure 6. Newell next writes the surfaces into a picture buffer memory in inverse order of priority. Because of the priority order, each surface will be written

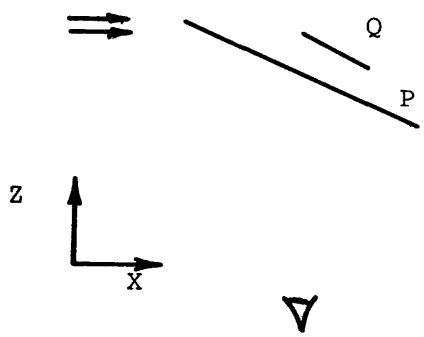


Figure 5—The Z will sort faces Q and P into the incorrect order Q, P. However, the Newell special sort will interchange the order and discover that the order P, Q is acceptable.

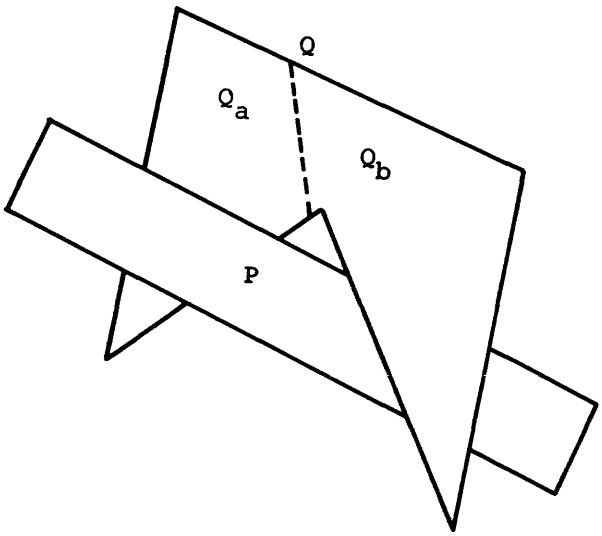


Figure 6—Faces P and Q cannot be placed in priority order because they conflict. However, if Q is divided into two faces Q_a and Q_b by the plane of P, then the order Q_b, P, Q_a is acceptable.

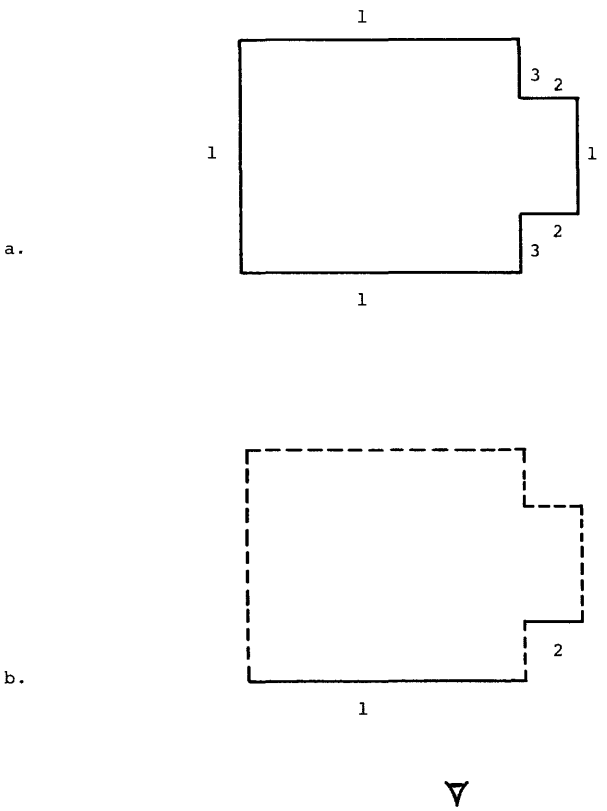


Figure 7—Face Priority. (a) Top view of an object with face priority numbers (lower numbers are higher priority). (b) Same object with a specific viewpoint located. Dashed lines show back faces. FAce '1' takes priority over face '2'.

into the buffer after any other surfaces that it may obscure, and will thus overwrite them. By allowing certain surfaces to partially (rather than completely) overwrite, Newell achieved some interesting transparency effects.

Schumacker, on the other hand, computed as much of the priority ordering information as possible for the environment before ever making any pictures. Schumacker observed that within certain *clusters* of faces a priority order could be discovered which is independent of viewpoint (see Figure 7). He divided his environment into such clusters so that for each frame of output picture his algorithm need only compute the relationship among the clusters rather than among all of the faces. Schumacker restricted his environment to ensure that clusters would be *linearly separable*, that is that they could always be separated by planes, as shown in Figure 8a, and stored the rules for such separation in a precomputed tree, as shown in Figure 8b. For each new viewpoint a prefix walk of this tree would produce the correct priority order of clusters. Schumacker's algorithm is thus very well matched to the essentially static environments found in flight simulation, but not at all well suited to environments in which everything moves independently.

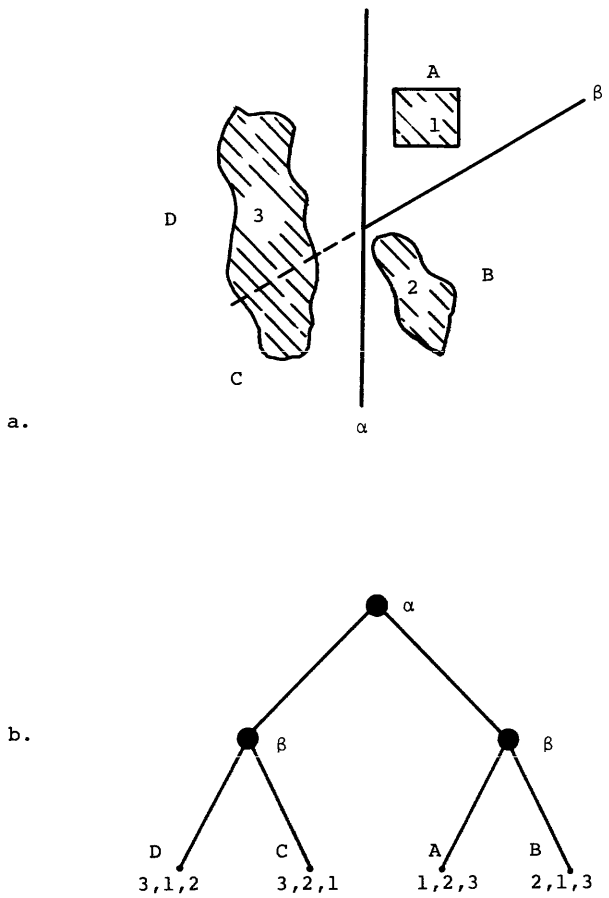


Figure 8—Cluster priority. (a) Three clusters (1,2,3) are separated by two planes (α,β). The viewpoint may be located in one of four areas (A,B,C,D). (b) A tree structure for finding the cluster priority. At nodes labeled with planes, we take a branch depending on which side of the plane the viewpoint lies. The result is to sort the clusters into priority order.

Depth priority algorithms

The depth priority algorithms divide neatly into two different categories: those that sample areas of the screen (Warnock), and those that sample infinitesimal points on the screen (scan-line algorithms). We shall call these two approaches *area sampling*, and *point sampling*.

The aim of the area-sampling approach is to compute an appropriate intensity for every area of the screen. If much of the screen is homogeneous, such as sky or background intensity, the area-sampling approach need only perform one computation for each such homogeneous area. In other words, the algorithm capitalizes on area coherence.

The point-sampling or scan-line algorithms are all designed to compute answers to the hidden-surface problem in a form and order suitable for a raster-scan display, such as a television monitor. These algorithms compute the intersection of the plane of a scan-line and each face in the environment; the line segments resulting from these intersections are called *segments*. As we shall see, the scan-line algorithms capitalize on the coherence properties of segments: the relations among segments change only slightly from one scan-line to the next.

The creation of segments simplifies the hidden-surface problem to an analogous problem on segments in two dimensions: segments are measured by X and Z coordinates only. The reduction of the problem from three to two dimensions makes many common computations, such as those that test segments for overlap or depth, simpler than the corresponding tests in three dimensions used in the area algorithms.

This reduction has one serious drawback: the intensity calculated for a raster element cannot be an *average* intensity corresponding to all visible items that fall within the square raster element. Instead, the intensity is based solely on one computation at a discreet point. As a result, objects can "disappear" between scan-lines or between raster elements as shown in Figure 9. Even though the lateral extent of these objects is below the lateral resolution of the screen, it is important that illumination of these objects be included when calculating intensities at surrounding raster elements. Similarly, raster elements near edges of large objects must have intensities that represent the average intensity within the raster element; if this average is not computed, ugly "sawtooth" patterns are displayed at object boundaries.

J. E. Warnock (1968)

The Warnock algorithm hypothesizes that sample areas on the screen, called *windows*, can be declared to be

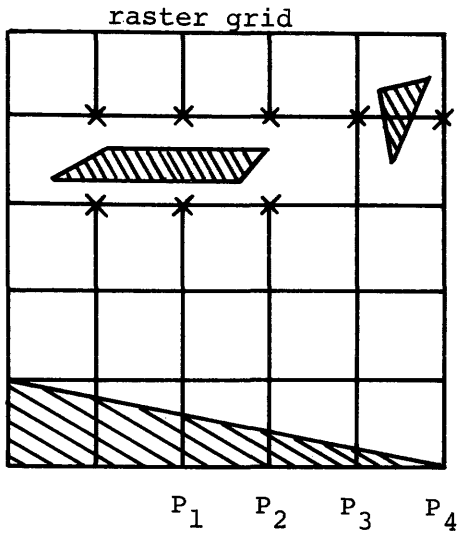


Figure 9—Incorrect shading intensities result unless locations of objects within a raster element are measured. The two small objects should contribute to the intensity at the points marked with x's. Similarly, the points P_1, P_2, etc. should have different intensities because the object does not fill the raster element.

homogeneous, and hence can be displayed after a simple shade calculation. The hypothesis is considered correct if (1) no faces fall within the sample window at all, or (2) one face completely covers the window and is nearer the viewpoint than every other face that falls in the window. If the hypothesis cannot be proven true, the sample window is divided into four smaller sample windows, and each of these is examined analogously. When the size of the sample windows reaches the size of the raster element, the subdivision process is terminated.

The procedure for dividing a sample window is a cull. A set of faces is compared to the window to see whether the face (1) surrounds the window, (2) intersects the window or (3) is completely disjoint from the window as shown in Figure 10. This cull for a sample window need not test *all* faces in the environment. The cull which produces new face lists for the four subwindows need only test faces which intersected the original window; faces disjoint from the large window will certainly be disjoint from the four small windows and faces which surrounded the original

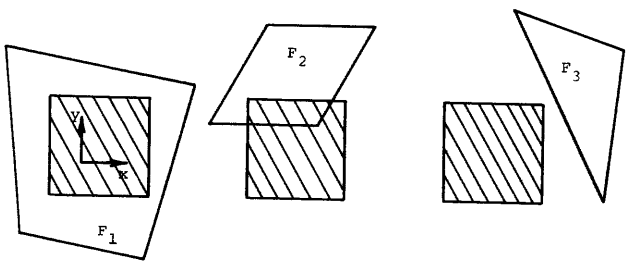


Figure 10—The relationship between a face and a sample window. F_1 surrounds the window; F_2 intersects the window, and F_3 is disjoint from the window. Note that these properties depend only on X-Y relationships, not Z.

window will surround its descendent windows. Warnock saves "ancestral information" with each face to avoid needless computation: the surrounder and disjoint properties can both be passed down to subwindows.

The cull operation is turned into a legitimate sort (the "Warnock Special") by the subdivision operation. In fact, the algorithm bears a striking resemblance to a quicksort: the faces are culled into two groups; those that are disjoint from this window, and those that are relevant to this window. The relevant faces are then passed down to subwindows, where the faces are again culled according to a new criterion (aspect to the smaller window), and so forth. The process terminates when a window is verified to be homogeneous, just as quicksort terminates when the lists contain indistinguishable elements. The Warnock cull and subdivision thus become a radix 4 quicksort.

One difficulty with the Warnock algorithm is that its output cannot conveniently be passed to a raster-scan device like a television. The decisions about windows are reached in a seemingly random order, rather than in a top-to-bottom left-to-right order. D. Cohen has devised a scheme for driving a raster display from window computations, but it involves a massive sort of the windows by Y and X coordinates (4).

*Scan-line algorithms**

The three point-sampling scan-line algorithms are remarkably similar. For lack of space, we shall describe only the general philosophy used by all three, knowing the differences among them.

All three algorithms perform a Y sort, then an S sort, and finally, a Z depth search to establish the visible face. The purpose of the Y sort is to limit the attention of the algorithm, on each scan-line, to only those edges or faces that intersect the scan-line. As processing for each scan-line begins, the Y-sorted structure is examined to find any new edges that enter on this scan-line: they are added to those already entered. Any edges that terminate on this scan-line are likewise discarded.

Sorting edges by Y already takes advantage of one kind of scan-line coherence: the edges that intersect one scan-line are very likely to intersect the next scan-line. It is therefore quite sensible to keep a list of "active" edges and merely make incremental changes to this list as new edges enter or as old edges terminate.

Given a list of active edges, the algorithms proceed to examine them in order to compute which faces are visible in which portions of the scan-line. This process involves dividing the scan-line into smaller sections, called sample spans, within which the same face is visible. Here, the algorithms are capitalizing on another form of coherence: point-to-point coherence along the scan-line.

The processing of each sample span requires comparing the faces that fall within that span to determine which

* C. Wylie, G. W. Romney, D. C. Evans, A. C. Erdahl (1967). W. J. Bouknight (1969). G. S. Watkins (1970).

one is closest. The exact details of this comparison depend on the method of selecting sample spans. For example, if sample spans go from edge-crossing to edge-crossing, then the comparison is quite straightforward: we merely compare the depths of the faces at the limits of the sample span. As Bouknight[2] showed, the procedure must be altered slightly if penetrating faces are allowed. This process is illustrated in Figure 11. At each X coordinate indicated with a caret, we compute the nearest face; that face is visible at least until the next edge crossing (i.e., next caret). A more aggressive selection of sample spans is shown in Figure 11b.

To summarize, the scan-line algorithms have four basic steps: (1) edges are sorted by Y so that only those edges intersecting the current scan-line need be examined; (2) on each scan-line, appropriate sample spans are determined (this usually involves sorting the edges on the scan-line by X coordinate); (3) within each sample span, we must cull out the segments which fall in the span and therefore must be examined; (4) the segments that fall within a span are searched to find which one is visible. These four operations are called Y sort, X sort, span cull, and Z depth search respectively.

OBSERVATIONS

Our avowed intent in this activity was to study systematically the existing hidden-surface algorithms to discover what principles they share in common and what distinctions exist between them. We had hoped that such a study might suggest new approaches to the hidden-surface problem. In order to find such algorithms we need to examine our categorization carefully for missing nodes and for other combinations of the basic operations found in the various algorithms.

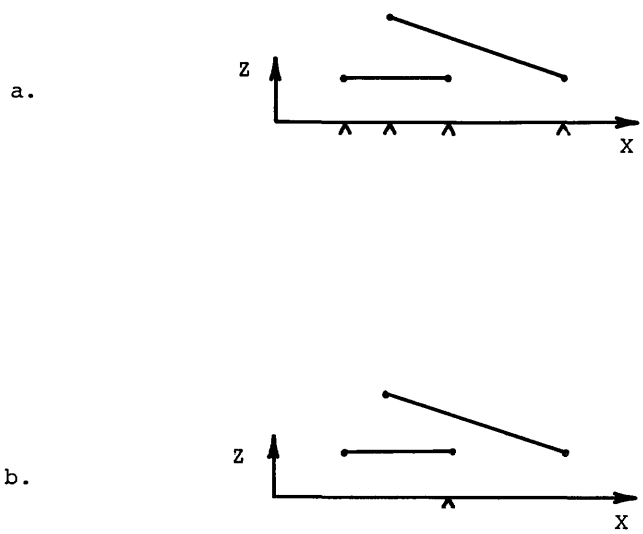


Figure 11—Sample span selection. (a) Each edge crossing starts a new span. (b) Aggressive sample spans. The caret divides the scan-line into two manageable spans.

Coherence

Each of the algorithms chooses some form of coherence as the basis for efficiently computing the rendering. In some cases, the use of coherence permits special performance gains in sorting operations; in others, coherence allows incremental calculations to replace more costly direct computations. Roberts used object coherence, noticing that each object can, at most, divide an edge into two pieces. Appel, Galimberti & Montanari and Loutrel all used edge coherence, progressing outward along the network of edges from some starting point in order to promote the known visibility of one vertex along edges to other vertices. The Schumacker and Newell algorithms both make use of depth coherence to precompute an order of priority for the faces, and Schumacker makes use of the additional coherence available in clusters to reduce the per-frame computing cost at some additional investment in environment preparation.

Finally, the four remaining algorithms make use of lateral coherence to reduce the number of surfaces under consideration at any position on the screen by eliminating from consideration those that are laterally displaced. Warnock used a kind of lateral coherence that is symmetric in the X and Y screen directions, whereas the other three, Watkins, Romney *et al.*, and Bouknight, made specific separation between the X and Y processes in order to capitalize on particularly favorable sorting techniques; bucket and bubble sorting.

We believe that the principal untapped source of help for hidden-surface algorithms lies in frame and object coherence. These types of coherence are closely interrelated because the objects presumably do not change between frames and thus, any computation done on objects may be preserved from one frame to the next. Only Schumacker[13] was able to make significant use of frame coherence. Frame and object coherence should be powerful helps to the hidden-surface problem because the objects rendered are usually well-behaved and the changes between frames are often minimal. For example:

- Newell *et al.* might make use of a frame coherence by saving the priority order of faces from one frame to the next. Thus, instead of an initial Z sort to get an approximately ordered list, one might use the list from the previous frame.

Area coherence is a basic and powerful tool for the hidden-surface algorithms. If Watkins' statistics[16] are anywhere near right, the problem of rendering hidden-surface objects is far more one of finding out which surfaces among the thousands in the environment are involved in any part of the screen than it is a task of finding which of the two or three that are involved is the frontmost one. We feel that the parts of a picture should be sorted according to the area of the screen in which they appear:

- Appel and the related algorithms could achieve a substantial gain in efficiency by sorting edges according to the boundaries of the smallest bound-

ing rectangle. An edge could then be compared with a collection of edges very much smaller than the total set of edges, namely those whose bounding boxes overlap.

Depth coherence might also be used in new ways:

- The Utah family might keep a depth sorted list of polygons "active" during each segment in the scan-line rather than merely remembering the single one which is frontmost. As edges were crossed, new polygons might be entered into this list in the same numerical position (e.g., third), in which they were found to lie in the previous scan-line. The correct position would then be confirmed by comparison with adjacent surfaces and corrected by bubbling if necessary.

Sorting order

In searching for a new combination of coherences to use, we are struck by the fact that one can consider the order of sorting as a measure of the types of coherence used. Sorting can occur along a specific dimension, *X*, *Y*, or *Z*, or along a combination of dimensions as in Warnock's area (*XY*) sort. Enumerating all possible orderings of such sorts, we find:

ZYX	Newell and Schumacker
ZXY	Uninteresting variant for TV output
YXZ	Romney, Watkins, Bouknight
XYZ	Uninteresting variant for TV output
YZX	Untried**********
XZY	Uninteresting variant for TV output
(XY)Z	Warnock
Z(XY)	This is what Newell would be were a frame buffer memory available to him. This scheme was implemented by Schumacker *et al.* (19)

After each sorting operation in the various algorithms the number of items left to consider is reduced, often by more than an order of magnitude. On the other hand, the number of times that the resulting smaller number of items must be considered is vastly increased. The selection of the types of sorts must account not only for the shortened lists but also for the increased number of times they must be sorted.

An Untried Sorting Order. It is interesting that there is an untried order of sorting. Let us consider what its properties might be. By the initial *Y* sort, we are making some use of lateral coherence to reduce the number of faces that need be considered at the next sorting step. Use of bucket sorting for the limited resolution required in this sort is particularly attractive.

The intermediate *Z* sort might be accomplished by something like Newell's method, and because there are fewer faces to sort, the effect of the square law involved will be reduced. The final *X* sorting step would make full use of the scan-line coherence properties familiar in the other algorighms. *X* intercepts would be incrementally computed and kept in *X* order by a bubble sort. One would make use of scan-line depth coherence by remembering which segments are visible from scan-line to scan-line and repeating a visible segment if no edge crossing has occurred involving one of its visible edges. Note that penetration conflicts will have been resolved for entire faces during the *Z* sort process.

Finally, one would capitalize on depth coherence by keeping an ordered list of involved segments during a scan. As scanning progressed, new segments would be entered in this list in the same position as was found appropriate during the last scan-line. From there the correct position would be found by bubble sort. This process would make available an ordered set of surfaces involved with the scanning ray at each point, and thus provide for the transparency effects Newell so attractively portrayed.

Conclusions

The conclusions we have reached from this paper are threefold. First, the hidden surface problem is one of sorting. Second, sorting methods which involve square-law computation growth with complexity are to be avoided. And third, the taxonomic approach taken in this research has amply justified the trouble.

Our conclusion that sorting is at the heart of the hidden-surface problem seems inescapable in view of the considerable light such an approach has shed on the various algorithms. In every algorithm examined the sorting steps have been clearly definable, clearly separable, and easily describable. Most important, taking this view has provided the basis for a framework within which to categorize the various algorithms and thus an approach toward seeking improved algorithms.

Although many approaches to the hidden-surface problem are applicable to simple situations, more complex environments rapidly eliminate the square-law approaches. Because the hidden-surface computation is difficult at best, great care must be taken in selecting sorting methods which will conserve computation by capitalizing on coherence.

Finally, our taxonomic approach seems to have provided substantial fuel for future research. Having suggested a framework, the framework itself suggests how to look for improved algorithms.

ACKNOWLEDGMENTS

We would like to thank Ed Catmull for his continuing interest in our activities and his many discussions with us. Thanks are also due to the several authors of algorithms who have spent time with us in person and by telephone helping us gather an understanding of what they have done. The work reported here was supported by the Office of Naval Research under contract N 00014-72-C-0346.

APOLOGIA

We would like to apologize in advance for any injustices we may have done to those authors whose programs and papers we may have overlooked, and to the authors of the papers we have used. We have associated certain key ideas with certain authors on the basis of the published works with which we are familiar. Other authors may have used these ideas, or even have priority on their invention; while we have tried to be careful about attribution, we are more interested in what is to be learned through examination of a set of algorithms than in the historical record of who invented what. Because algorithms change, moreover, later versions of the algorithms described may include features which we have omitted.

REFERENCES

1. Appel, A., "The Notion of Quantitative Invisibility and the Machine Rendering of Solids," *Proceedings 22nd National Conference*, ACM, Thompson Books, Washington, D.C., p. 387, 1967.
2. Bouknight, W. J., *An Improved Procedure for Generation of Halftone Computer Graphics Representations*, University of Illinois, Coordinated Science Laboratory, R-432, Sept. 1969.
3. Bouknight, W. J., "A Procedure for Generation of Three-Dimensional Half-toned Computer Graphics Representations," *CACM*, 13, 9, 527, September 1970.
4. Cohen, D., *Incremental Methods for Computer Graphics*, ESD-TR-69-193, Harvard University, April 1969.
5. Galimberti, R., Montinari, U., "An Algorithm for Hidden-Line Elimination," *CACM*, 12, 4, 206, April 1969.
6. *Electronic Scene Generator Expansion System*, General Electric Corp., Final Report, NASA contract NAS 9-11065, December 1971.
7. Loutrel, P. P., *A solution to the Hidden-Line Problem for Computer-Drawn Polyhedra*, Department of Electrical Engineering, New York University, Bronx, N. Y., Technical Report 400-167, September 1967. (Available from University Microfilm, Ann Arbor, Michigan).
8. Loutrel, P. P., "A Solution to the Hidden-Line Problem for Computer-Drawn Polyhedra," *IEEE Transactions on Electronic Computers*, EC-19, 3, 205, March 1970.
9. Newell, M. E., Newell, R. G., Sancha, T. L., "A New Approach to the Shaded Picture Problem," *Proceedings ACM National Conference*, 1972.
10. Newman, W. M., Sproull, R. F., *Principles of Interactive Computer Graphics*, Mc-Graw-Hill, 1973.
11. Roberts, L. G., *Machine Perception of Three-Dimensional Solids*, MIT Lincoln Laboratory, TR 315, May 1963. Also in *Optical and Electro-Optical Information Processing*, Tipper et al. eds. MIT. Press, 159.
12. Romney, G. W., *Computer Assisted Assembly and Rendering of Solids*, Department of Computer Science, University of Utah, TR-4-20, 1970.
13. Schumacker, R. A., Brand, B., Gilliland, M., Sharp, W., *Study for Applying Computer-Generated Images to Visual Simulation*, AFHRL-TR-69-14, U. S. Air Force Human Resources Laboratory, September 1969.
14. Sutherland, I. E., *A Characterization of Hidden-Surface Algorithms*, speech at Carnegie-Mellon University, 5 June 1972.
15. Sutherland, I. E., Hodgman, G. W., *Reentrant Polygon Clipping*, to appear.
16. Sutherland, I. E., Sproull, R. F., Schumacker, R. A., *A Characterization of Ten Hidden-Surface Algorithms*," to appear.
17. Warnock, J. E., *A Hidden-Surface Algorithm for Computer-Generated Halftone Pictures*, Computer Science Department, University of Utah, TR 4-15, June 1969.
18. Watkins, G. S., *A Real-Time Visible Surface Algorithm*, Computer Science Department, University of Utah, UTECH-CSc-70-101, June 1970.
19. Wild, E. C., Rougelot, R. S., Schumacker, R. A., *Computing Full Color Perspective Images*, General Electric TIS R71ELS-26, May 1971.
20. Wylie, C., Romney, G. W., Evans, D. C., Erdahl, A. C., "Halftone Perspective Drawings by Computer," *AFIPS Fall Joint Computer Conference* 1967, Thompson Books, Washington, D. C., 49.

Packet switching with satellites*

by NORMAN ABRAMSON

University of Hawaii
Honolulu, Hawaii

INTRODUCTION

History

The first computer-communication networks put into operation were designed around the communications provided by the existing worldwide telephone network. Lucky has given a convincing rationale for that decision.[1]

> "The voice telephone network is perhaps the most remarkable information processing system yet constructed by man. In 1970 it served 100,000,000 telephones in the United States. The number of possible interconnections is clearly enormous. The worth of this plant is approximately 50 billion dollars. Over one million people are employed by AT&T alone in the care and feeding of this huge network. Virtually every statistic associated with the telephone network can be phrased in some extraordinary manner. Its ready accessibility and virtual ubiquity make it the obvious first contender for handling data traffic."

As the limitations of this system for data communications became apparent, a number of methods were introduced to overcome the limitations of dial-up telephone and leased line systems. Data concentrators are used to increase the utilization of expensive long distance lines. High speed, wideband facilities are used to handle those situations where the burst data rate requirement of the network is larger than can be transmitted in a single voice channel. A few large systems use leased line data channels in a network with multiple paths between nodes for increased reliability. All of the systems built before 1970 however based the organization of their data communication channels on the circuit switching methods developed for voice signals during the latter part of the 19th century.

As the need for more powerful and more flexible computer-communication networks, distributed over large geographical areas, increased the basic limitations imposed by the organization of circuit switched systems was questioned.[2,3,4] By 1970 the ARPA Network,[5] the first computer-communication system to employ packet switching techniques suited to the peculiar statistics of digital data had gone into operation. The network is described in Reference 6:

> "The ARPA Network is a new kind of digital communication system employing wideband leased lines and message switching, wherein a path is not established in advance and instead each message carries an address. Messages normally traverse several nodes in going from source to destination, and the network is a store-and-forward system wherein, at each node, a copy of the message is stored until it is safely received at the following node. At each node a small processor (an *Interface Message Processor*, or *IMP*) acts as a nodal switching unit and also interconnects the research computer centers, or *Hosts*, with the high bandwidth leased lines."

By January 1973 the use of packet switching techniques in the ARPA Network had made possible a resource sharing computer network among more than 30 large machines; these machines represent an investment of more than $80,000,000, span a geographical region from Hawaii to Massachusetts and the network is still expanding at a rapid rate. At this time packet switched techniques are under consideration for other computer-communication networks in the USA, Canada, Japan and Western Europe.[7,8] But no common carrier has yet announced plans for a packet switched data service for the general user of data communications.

Although the basic packet switched method of organizing communication channels in the ARPA Network represents a significant step forward from the circuit switched methods of the voice oriented common carriers the communications medium of the ARPA Network (with the exception of a special satellite link to the University of Hawaii) is still the point-to-point wire (or microwave) channel.

* THE ALOHA SYSTEM is a research project at the University of Hawaii, supported by the Advanced Research Projects Agency under NASA Contract No. NAS2-6700 and by the U.S. Air Force Office of Aerospace Research under Contract No. F44620-69-C-0030. Part of the work reported in this paper was supported by Systems Research Corporation, Honolulu, under ONR Contract N00014-70-C-0414.

The medium is the multiplexor

In June 1971 the first remote terminal in THE ALOHA SYSTEM, an experimental UHF radio, packet switched network was put into operation at the University of Hawaii.[9] THE ALOHA SYSTEM is a packet switched computer communication network using many of the design concepts of the ARPA Network. The design of THE ALOHA SYSTEM departs from that of the ARPA Network in two major respects however. The first is in the use of a new form of burst random access method of employing a data communication channel. That method is particularly attractive for use with a broadcast radio channel such as in THE ALOHA SYSTEM; the characteristics of the ALOHA burst random access communication method are described in the next section.

The other respect in which the design of THE ALOHA SYSTEM departs from that of the ARPA Network, and indeed from the design of all other computer networks, is in the form of multiplexing which occurs in THE ALOHA SYSTEM. The network uses two 24,000 bits/second channels for all remote units—one of these channels is used by all remote units for data into a central machine (an IBM 360/65) and the other channel is used for data out of the central machine. Since data packets from all remote users access the same 24,000 bits/second radio channel in 30 millisecond bursts, each user automatically multiplexes their data onto that single channel at the time it transmits its packet. Thus the multiplexing is accomplished between the transmitting antenna at each user station and the receiving antenna at the central station. Steven Crocker of ARPA has characterized this effect by noting that in THE ALOHA SYSTEM, "the medium is the multiplexor".

A final point should be brought out about the lack of need for multiplexing equipment in THE ALOHA SYSTEM. The cost of communications for a network of terminals connected to a central time sharing system is often thought of as being composed of the line charges (lease cost or dial-up charges), the modem charges at either end of the link plus perhaps some portion of the cost of the communications processor. For long distance connections to a machine the line charges will usually dominate the cost of communications. Even for local connections however the real costs of simply connecting a terminal to a machine by common carrier communication facilities are hard to come by. A good portion of these costs can often be attributed to the front end communications processor and multiplexor. The need to sample telephone input lines on a frequent basis and to assemble characters, limits the number of input lines which can be handled by a single processor and the data rates at which these lines can operate. Some indication of the magnitude of the cost of performing these functions can be obtained from a survey of national time sharing services published in November, 1971.[10] The typical charge for connect time to one of these services (that is, the cost necessary for simply tying up communications resources, not CPU time) was about $10/hour.

Since multiplexing in THE ALOHA SYSTEM is accomplished automatically the channel now used in the system is capable of handling over 500 active terminals[9] each transmitting packets at a *burst* data rate of 24,000 bits/second. (Of course the *average* data rate of each user must be well below 24,000 bits/second.)

The ALOHA channel

Consider a number of widely separated users each wanting to transmit data packets over a single high speed communication channel. Assume that the rate at which the users generate packets is such that the average time between packets from a single user is much greater than the time needed to transmit a single packet. (In THE ALOHA SYSTEM the ratio of these times is about 2,000 to 1.)

Conventional time or frequency multiplexing methods or some kind of polling scheme could be employed to share the channel among the users. Some of the disadvantages of these methods are discussed by Roberts in a related paper in this session.[14] The method used by THE ALOHA SYSTEM is suggested by the statistical characteristics of the packets generated by remote users. Since each user will generate packets infrequently[11] and each packet can be transmitted in a time interval much less than the average time between packets the following scheme seems natural.

Each user station has a buffer which it uses to store one line of text. When the line is complete a header containing address, control and parity information for a cyclic error detecting code is appended to the text to form a packet and the packet is transmitted to the central station. Each user at a console transmits packets to the central station over the same high data rate channel in a completely unsynchronized (from one user to another) manner. If and only if a packet is received without error it is acknowledged by the central station. After transmitting a packet the transmitting station waits a given amount of time for an acknowledgment; if none is received the packet is automatically retransmitted. This process is repeated until a successful transmission and acknowledgment occurs or until some fixed number of unsuccessful transmissions has been attempted.

A transmitted packet can be received incorrectly because of two different types of errors; (1) random noise errors and (2) errors caused by interference with a packet transmitted by another console. The first type of error has not been a serious problem on the UHF channels employed. The second type of error, that caused by interference, will be of importance only when a large number of users are trying to use the channel at the same time. Interference errors will limit the number of users and the amount of data which can be transmitted over this ALOHA random access channel as more remote stations are added to THE ALOHA SYSTEM.

Capacity of ALOHA channels

In order to describe these limits we assume that the start times of message packets in our channel comprise a Poisson point process with parameter λ packets/second. If each packet lasts τ seconds we can define $S=\lambda\tau$, where

$$S = \text{normalized channel message rate} \qquad (1)$$

S is called the normalized channel message rate since a value of S equal to one would correspond to a channel with packets synchronized perfectly so that the start of one packet always coincided with the end of the previous packet. (Of course this will not occur because of our Poisson assumption.) Note that S takes into account only message packets, not retransmission packets.

In addition we assume that the start times of the message packets plus packet retransmissions comprise another Poisson point process. (This assumption will hold only if the packet retransmission delays are large. See Reference 9.) Then we can define a quantity G, analogous to the normalized channel message rate, which takes into account the message packets plus the retransmission packets.

$$G = \text{normalized channel traffic rate} \tag{2}$$

In general we know that

$$G \geq S \tag{3}$$

In Reference 9 we showed that

$$S = Ge^{-2G} \tag{4}$$

and this relationship is plotted in Figure 1.

Note from Figure 1 that the message rate reaches a maximum value of $\frac{1}{2}e = 0.184$. For this value of S the channel traffic is equal to 0.5. The traffic on the channel becomes unstable at $S = \frac{1}{2}e$ and the average number of retransmissions becomes unbounded. Thus we may speak of this value of the message rate as the *capacity* of this random access data channel. Because of the random access feature the channel capacity is reduced to roughly one sixth of its value if we were able to fill the channel with a continuous stream of uninterrupted data.

The form of channel analyzed above corresponds to THE ALOHA SYSTEM channel now in operation.

It is possible to modify the completely unsynchronized use of the ALOHA channel described in order to increase the capacity of the channel. In the pure ALOHA channel each user simply transmits a packet when ready without any attempt to coordinate his transmission with those of other users. While this strategy has a certain elegance it does lead to somewhat inefficient channel utilization. If we can establish a time base and require each user to start his packets

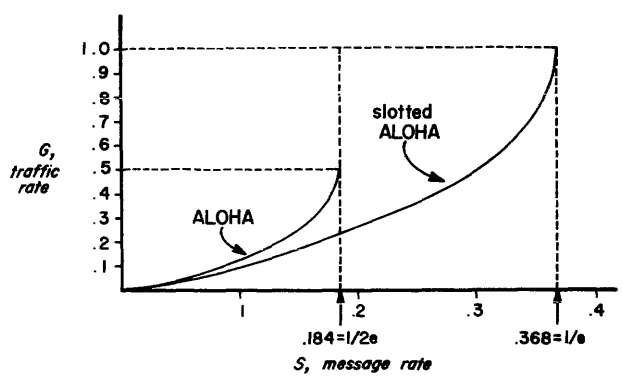


Figure 1—Traffic rate vs. message rate for a pure ALOHA channel and a slotted ALOHA channel

only at certain fixed instants it is possible to increase the channel capacity. In this kind of channel, called a *slotted ALOHA channel*, a central clock establishes a time base for a sequence of "slots" of the same duration as a packet transmission. Then when a user has a packet to transmit he synchronizes the start of his transmission to the start of a slot. In this fashion, if two messages conflict they will overlap completely, rather than partially.

To analyze the slotted ALOHA channel define S_i as the probability that the i'th user will send a packet in some slot. Assume that each user operates independently of all other users and that whether a user sends a message in a given slot does not depend upon the state of any previous slot. If we have n users we can define $S = \sum_{i=1}^{n} S_i$, where

$$S = \text{normalized channel message rate} \tag{5}$$

As before we can also consider the rate at which a user sends message packets plus packet retransmissions. Define the probability that the i'th user will send a message packet or a packet retransmission as G_i. Then, for n identical users we define $G = \sum_{i=1}^{n} G_i$ where

$$G = \text{normalized channel traffic rate} \tag{6}$$

and, as in the pure ALOHA channel

$$G \geq S \tag{7}$$

We note here that although S, the sum of the S_i, is the probability that some user will send a message packet in a given slot, the analogous statement is not true for G. The sum of the G_i is not the probability that some user will send a message or repetition packet in a given slot. In fact even though G is the sum of the probabilities G_i, G is not itself a probability and G may be greater than 1.

For the slotted ALOHA channel with n independent users, the probability that a packet from the i'th user will not experience an interference from one of the other users is

$$\prod_{j=1, j\neq i}^{n} (1-G_j)$$

Therefore we may write the following relationship between the message rate and the traffic rate of the i'th user.

$$S_i = G_i \prod_{j=1, j\neq i}^{n} (1-G_j) \tag{8}$$

If all users are identical we have

$$S_i = \frac{S}{n} \tag{9}$$

and

$$G_i = \frac{G}{n} \tag{10}$$

so that (8) can be written

$$S = G\left(1-\frac{G}{n}\right)^{n-1} \tag{11}$$

and in the limit as $n \to \infty$, we have

$$S = Ge^{-G} \tag{12}$$

Equation (12) is plotted in Figure 1 (curve labeled Slotted ALOHA). Note that the message rate of the Slotted ALOHA channel reaches a maximum value of $1/e = 0.37$, twice the capacity of the pure ALOHA channel.

This result for Slotted ALOHA channels was first derived by Roberts[12] using a different method.

PROPERTIES OF SATELLITE CHANNELS

The cable in the sky

In the worldwide telephone system satellites are used more or less interchangeably with cables for transmission of voice signals. Because of this desirable feature, it is not surprising that the common carriers and even satellite designers have tended not to emphasize the differences between cable and satellite channels.

A communications satellite however is not just a big cable in the sky. There are several significant differences between the communication channel properties of a cable or microwave link and the communication channel properties of a satellite transponder.

In the next three sections we shall explain some of these differences and how they can affect the operation of a packet switched system using a satellite. But first we should mention one property of a satellite channel which the common carriers have emphasized. A satellite transponder in geosynchronous orbit is stationed 36,000 kilometers above the equator. A signal transmitted using the satellite will therefore experience a delay of about a quarter second, corresponding to the round trip propagation time up to the satellite and down again. This delay can decrease the effective data rate of certain error control schemes requiring positive acknowledgments sent from the receiver back to the transmitter. Such schemes should not ordinarily be used over satellite channels.

There are three properties of communication satellites which we want to discuss here, in terms of their significance to packet switched communications. These are:

(a) data rates
(b) bilateral broadcasting
(c) perfect information feedback

Data rates

The first property of satellite channels is not a fundamental property of the satellite itself, but rather a property of how the satellite is used. A single voice channel on INTELSAT IV uses a bandwidth of 45 Khz. and provides the capability of transmitting data at 56 kilobits on a single voice channel. This mode of operation is in fact employed in the SPADE demand assignment system now used in the Atlantic satellite; it is employed in the single-channel-per-carrier digital voice link installed in the Paumalu earth station in Hawaii and the Jamesburg earth station in California. Since December 1972, THE ALOHA SYSTEM has been linked to the ARPANET using a single leased satellite voice channel to transmit data at 50 kilobits to NASA Ames Research Center in California.

Bilateral broadcasting

In the conventional use of communication channels the term "broadcasting" refers to the fact that many receivers may obtain the transmission from a single transmitter. Perhaps the most striking feature of a satellite channel is its broadcast nature as opposed to the point-to-point nature of wire channels. The reception of broadcast signals for satellite communication channels used with conventional circuit switched methods is a natural idea. But when a satellite channel is used in a packet switched mode it is possible to consider broadcasting use of the channel by transmitters as well as receivers. This capability we have called *bilateral broadcasting*.

Since a number of transmitting ground stations operating in a packet switched mode may all access the same channel in an unsynchronized (from ground station to ground station) fashion the analysis of an earlier section applies to bilateral broadcasting without any change. Each of the twenty or more ground stations accessing a given INTELSAT IV channel can transmit packets at will up to the ALOHA random access capacity of that single channel.

There is no technological reason why such a system could not be employed now to extend the capabilities of the existing worldwide satellite communication network in data communications. There is an existing regulatory restriction on such an unconventional use of INTELSAT IV however and discussions are under way with several agencies to remove these regulatory barriers in either the INTELSAT system or one of the several domestic satellite systems to be installed (or already installed in two countries).

Except for the not inconsiderable constraints imposed by regulatory considerations the same 50 kilobit leased satellite channel linking THE ALOHA SYSTEM to the ARPANET could be used to link machines in Alaska, Japan, Australia and any of the other sixteen earth stations which access the Pacific satellite. While these regulatory problems are being worked out however THE ALOHA SYSTEM has established a limited burst random access satellite network using the packet switching techniques described. In a joint experiment with NASA Ames Research Center in California and the University of Alaska we are operating such a link by means of the NASA ATS-1 satellite. The satellite transponder is operated as an unslotted ALOHA channel between earth stations in Hawaii, Alaska and California, and although usage of that channel is now restricted to two hours per day or less and the data rate of the channel is only 20,000 bits/second, the experiment is providing valuable information on this new communications technique.

Perfect information feedback

In the use of satellites for packet switching yet another property of little value in circuit switching assumes importance. In a packet switched system each ground station has the capability of transmitting packets up to the satellite addressed to any other ground station (or to all other ground stations). Each packet is then received by all ground stations, *including the ground station which transmitted the packet*, approximately one quarter second later. Therefore each ground station can initiate transmission of a packet at will as in THE ALOHA SYSTEM. However, whereas in THE ALOHA SYSTEM, it is necessary to provide information on packet interference to the sender in the form of positive acknowledgments, such information is not necessary in the system we are describing. Since each sender can listen to his own packet retransmitted from the satellite each sender can be considered to have the same information on packet interference available to the receiver earth station. (In information theory terms, these channels are modeled as channels with perfect information feedback.)

Unfortunately in the real world, nothing is perfect and there will undoubtedly be circumstances when the transmitter and the receiver do not detect the same bit string from the satellite. The fact remains however that positive acknowledgments to combat packet interference are not required, and the more efficient use of a negative acknowledgment scheme in conjunction with packet numbering is feasible for this system.

EXCESS CAPACITY OF AN ALOHA CHANNEL

The idea

The type of packet switched satellite data channel we have described so far (either pure ALOHA or slotted ALOHA) has a certain elegant simplicity to it. The user of the channel simply transmits a burst of data when he wants at a data rate equal to that of the entire channel. Nevertheless there is a price to be paid for this simplicity in terms of channel capacity and in terms of delay. The question of delay is dealt with by Kleinrock[13] and Roberts[14] in the other two papers of this session. Roberts also discusses an effective method of employing the channel at rates significantly higher than the 37 percent capacity indicated by Figure 1. In the next section we provide some results which show that a slotted ALOHA channel can be used at rates well above 37 percent of capacity, if all users of the channel do not have identical message rates.

The idea of excess capacity in an ALOHA channel was first suggested by Roberts who derived a result for the case of several small users and a single large user of a slotted ALOHA channel. Roberts' proof was published along with a number of other interesting analytic results by Kleinrock and Lam.[15] The approach we shall take was suggested by Rettberg,[16] who also treated the case of a single large user and was able to obtain numerical results for that case. In the next section we provide a complete analytic and numerical solution to the use of slotted ALOHA channels by any number of users, each operating at an arbitrary rate.

The theory

From equation (8) we have a set of n equations relating the message rates and traffic rates of the n users

$$S_i = G_i \prod_{j=1, j\neq i}^{n} (1-G_j) \qquad i=1, 2, \ldots, n \tag{13}$$

Define

$$\alpha = \prod_{j=1}^{n} (1-G_j) \tag{14}$$

then (13) can be written

$$S_i = \frac{G_i}{1-G_i}\alpha \qquad i=1, 2, \ldots, n \tag{15}$$

For any set of n acceptable traffic rates $G_1, G_2, \ldots, G_n$ these n equations define a set of message rates $S_1, S_2, \ldots, S_n$, or a region in an n-dimensional space whose coordinates are the S_i. In order to find the boundary of this region we calculate the Jacobian,

$$J\left(\frac{S_1, S_2, \ldots, S_n}{G_1, G_2, \ldots, G_n}\right).$$

Since

$$\frac{\partial S_j}{\partial G_k} = \begin{cases} \prod_{i\neq j} (1-G_i) & j=k \\ -G_j \prod_{i\neq j,k} (1-G_i) & j\neq k \end{cases} \tag{16}$$

after some algebra we may write the Jacobian as

$$J\left(\frac{S_1, S_2, \ldots, S_n}{G_1, G_2, \ldots, G_n}\right) = \alpha^{n-2} \begin{vmatrix} (1-G_1) & -G_1 & -G_1 & \\ -G_2 & (1-G_2) & -G_2 & \ldots \\ -G_3 & -G_3 & (1-G_3) & \\ & \vdots & & \end{vmatrix}$$

$$= \alpha^{n-2}[1-G_1-G_2-\ldots-G_n] \tag{17}$$

Thus the condition for maximum message rates is

$$\sum_i G_i = 1 \tag{18}$$

This condition can then be used to define a boundary to the n dimensional region of allowable message rates, $S_1, S_2, \ldots, S_n$.

The results

Consider the special case of two classes of users with n_1 users in class 1 and n_2 users in class 2.

$$n_1+n_2=n \tag{19}$$

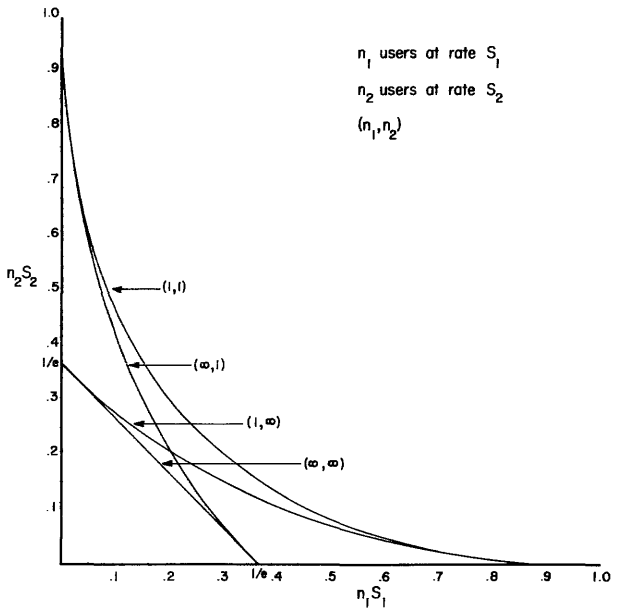


Figure 2—Allowable message rates

Let S_1 and G_1 be the message and traffic rates for users in class one, and S_2 and G_2 be the message and traffic rates for users in class 2. Then the n equations (13) can be written as the two equations

$$S_1 = G_1(1-G_1)^{n_1-1}(1-G_2)^{n_2} \tag{20a}$$

$$S_2 = G_2(1-G_2)^{n_2-1}(1-G_1)^{n_1} \tag{20b}$$

For any pair of acceptable traffic rates G_1 and G_2 these two equations define a pair of message rates, S_1 and S_2, or a region in the S_1, S_2 plane.

From (18) we know that the boundary of this region is defined by the condition

$$n_1G_1 + n_2G_2 = 1 \tag{21}$$

We can use (21) to substitute for G_1 in equation (20) and obtain two equations for S_1 and S_2 in terms of a single parameter G_2. Then as G_2 varies from 0 to 1 the resulting S_1, S_2 pairs define the boundary of the region we seek. A FORTRAN program to calculate the boundary was written and used to calculate several curves of the allowable region for different values of (n_1, n_2) (Figures 2, 3).

The important point to notice from Figures 2 and 3 is that in a lightly loaded Slotted ALOHA channel, a single large user can transmit data at a significant percentage of the total channel data rate, thus allowing use of the channel at rates well above the limit of 37 percent obtained when all users have the same message rate. This capability is important for a computer network consisting of many interactive terminal users and a small number of users who send large but infrequent files over the channel. Operation of the channel in a lightly loaded condition of course may not be desirable in a bandwidth limited channel. For a communications satellite where the average power in the satellite transponder limits the channel however[19] operation is a lightly loaded condition in a packet switched mode is an attractive alternative. Since the satellite will transmit power only when it is relaying a packet, the duty cycle in the transponder will be small and the average power used will be low.

Finally we note it is possible to deal with certain limiting cases in more detail, to obtain equations for the boundary of the allowable S_1, S_2 region.

(a) for $n_1 = n_2 = 1$
Upon using (21) in (20) we obtain

$$S_1 = G_1^2 \tag{22a}$$

$$S_2 = (1-G_1)^2 \tag{22b}$$

(b) for $n_2 \to \infty$

$$S_1 = G_1(1-G_1)^{n_1-1} \cdot \exp[-(1-n_1G_1)] \tag{23a}$$

$$S_2 = (1-n_1G_1)(1-G_1)^{n_1-1} \cdot \exp[-(1-n_1G_1)] \tag{23b}$$

(c) for $n_1 = n_2 \to \infty$

$$S_1 = \frac{G_1}{e}$$

$$S_2 = \frac{1-G_1}{e}$$

Additional details dealing with excess capacity and the delay experienced with this kind of use of a slotted ALOHA channel may be found in References 17 and 18.

PACKET SWITCHING IN DOMSAT

Background

The 50 kilobit INTELSAT channel now being used to link THE ALOHA SYSTEM to the ARPA Network could em-

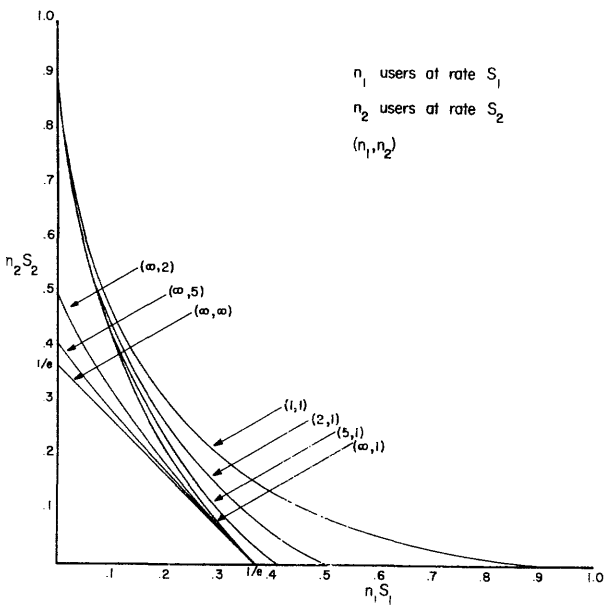


Figure 3—Allowable message rates

ploy the techniques we have described to link additional nodes in the ARPANET at each of the 16 earth stations with access to the Pacific satellite. These same techniques could also be employed by a common carrier to offer packet switched data communications of a quality to which we would all like to become accustomed.

As this is being written, there are in operation two domestic satellite systems (Molniya in the USSR and Anik in Canada) in addition to the worldwide INTELSAT system. Seven US domestic systems (DOMSAT) are under consideration and one has entered the construction phase with a first launch planned for 1974. Japan has announced plans for its domestic communications satellite and several other national systems are expected in the late 1970's. Most of the DOMSAT proposals plan a system of less expensive and therefore more numerous earth stations than the standard 97 foot earth station antennas now used in the INTELSAT system. Thus the advantage of using a lightly loaded packet switched channel in a power limited situation[19] assumes added importance.

A proposal

Consider the use of a single transponder in a US domestic satellite system to provide a public packet switched data communication service. INTELSAT IV employs 12 transponders each with 36 Mhz. bandwidth. Only one of these transponders devoted completely to a public packet switched service in a US domestic satellite system could easily provide data at a rate of 10 million bits/second into a small earth station. The public packet switched service in the US could provide burst data rates between small communication controllers at each earth station of 10 megabits. Assuming 100 earth stations over the US, and assuming the system is operated at a message rate $S=0.15$, the average data rate into and out of each station would be about 15 kilobits although the variance about this average (both from earth station to earth station and at different times at the same earth station) would be large. A packet switched system would function without difficulty in the face of large variations of this type.

The capacity of such a system measured in terms of interactive users of alphanumeric terminals would be about 100,000 such active users at any one time on the system. Of course the system would be used by other devices generating larger amounts of traffic than a single terminal and the number of active users would have to be decreased accordingly. The point is that in a public packet switched service using a US domestic satellite the user of data communications could be charged by the packet, since the user would consume resources in the system proportional to the number of packets sent and received.

The preceding three sections and the accompanying papers by Kleinrock and Roberts explain many of the technological advantages of such a system. We need only add some short observations concerning the operational advantages of a public packet switched service. The system would possess a flexibility of operation simply not attainable with circuit switched systems. Although the *average* data rate into each of 100 earth stations would be 15 kilobits, the *burst* data rate into any given terminal could be close to 10 megabits. This capability for remote job entry and file transfer leads to the same potential for resource sharing shown to be so valuable in the ARPANET.

Another kind of flexibility is the flexibility in being able to start such a system with a small number of communication controllers at a few earth stations. The system would become operational with only two stations and would yield data on packet interference patterns and delay with only three stations. Since the computer-communication network brought into being by such a service is completely connected (topologically) there is no need for routing algorithms at each earth station (such as used in the ARPANET IMPS and TIPS) and to add a new earth station into the network it is only necessary to activate the identification number of that station. Peak load averaging of such a system would operate to increase its total capacity since the system is peak load limited only at the satellite and not at the separate ground stations. (This particular advantage could be especially important for a Pacific packet switched service where the international dateline would serve to average peak loads over different days as well as different hours.)

Finally we note that the economics of such a system are consistent with the economics of existing computer communication systems. The ARPANET in its present configuration provides a factor of ten or more in cost advantage over conventional circuit switched systems.[5] During the month of January 1973, approximately 45,000,000 packets were transmitted by the ARPANET. The capacity of the ARPANET based on an eight hour day was about 300 million packets per month at that time. A public packet switching service using a single transponder of a domestic satellite system, operating at a normalized message rate of 0.15 would have a capacity of about 1,500 million packets per month, again based on an eight hour day. Furthermore, at such a low message rate the system would easily accommodate intermittent users with large files at a megabit data rate and still draw average power from the satellite corresponding to a transponder duty cycle of less than 16 percent. The 50 kilobit lines now used in the ARPANET cost about $1,200,000 per year in January 1973 and this figure was growing rapidly. The ARPANET is but one possible customer of a public packet switched service. The projected average annual revenue of a single transponder in the several proposed US domestic satellite systems ranges from less than $1,000,000 to about $3,000,000 per year.[20]

REFERENCES

1. Lucky, Robert W., "Common Carrier Data Communication," *Computer-Communication Networks*, edited by Norman Abramson and Franklin F. Kuo, Prentice-Hall, 1973.
2. Baran, Paul "On Distributed Communications: V. History, Alter-*Document No. 13044*, Stanford Research Institute, December 6, *PR, The Rand Corporation*, August 1964.

3. Baran, Paul., "On Distributed Communications: XI. Summary Overview," *Memorandum RM-3767-PR, The Rand Corporation*, August 1964.
4. Davies, D. W., Bartlett, K. A., Scantlebury, R. A., Wilkinson, P. T., "A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals," *ACM Symposium on Operating System Principles*, Gatlinburg, Tennessee, October 1-4, 1967.
5. Roberts, Lawrence G., "The ARPA Network," *Computer-Communication Networks*, edited by Norman Abramson and Franklin F. Kuo, Prentice-Hall, 1973.
6. Ornstein, S. M., Heart, F. E., Crowther, W. R., Rising, H. K., Russell, S. B., Michel, A., "The Terminal IMP for the ARPA Computer Network," *AFIPS Conference Proceedings, Spring Joint Computer Conference*, 1972, pp. 243-254.
7. deMercado, J., Guindon, R., DaSilva, J., Kadoch, M., "The Canadian Universities Computer Network Topological Considerations," *Computer Communication: Impacts and Implications, Proceedings of the First International Conference on Computer Communication*, Washington, D.C., October 24-26, 1972, pp. 220-225.
8. Barber, D. L. A., "The European Computer Network Project," *Computer Communication: Impacts and Implications, Proceedings of the First International Conference on Computer Communication*, Washington, D.C., October 24-26, 1972, pp. 192-200.
9. Abramson, Norman, "THE ALOHA SYSTEM," *Computer-Communication Networks*, edited by Norman Abramson and Franklin F. Kuo, Prentice-Hall, 1973.
10. Trifari, J. C., "Rating National Timesharing Services," *Computer Decisions*, November 1971, pp. 28-32.
11. Jackson, P. E., Stubbs, C. D., "A Study of Multi-access Computer Communications," *AFIPS Conference Proceedings, Spring Joint Computer Conference*, 1969, pp. 491-504.
12. Roberts, Lawrence G., "ALOHA Packet System With and Without Slots and Capture," *ARPANET Satellite System Note 8, NIC Document No. 11290*, Stanford Research Institute, June 26, 1972.
13. Kleinrock, L., Lam, S. S., "Packet Switching in a Slotted Satellite Channel," *Proceedings of the National Computer Conference*, June 1973.
14. Roberts, Lawrence G., "Dynamic Allocation of Satellite Capacity Through Packet Reservation," *Proceedings of the National Computer Conference*, June 1973.
15. Kleinrock, L., Lam S. S., "Analytic Results with the Addition of One Large User," *ARPANET Satellite System Note 27, NIC Document No. 12736*, Stanford Research Institute, October 30, 1972.
16. Rettberg R., "Random ALOHA with Slots-Excess Capacity," *ARPANET Satellite System Note 18, NIC Document No. 11865*, Stanford Research Institute, October 11, 1972.
17. Abramson, Norman, "Excess Capacity of a Slotted ALOHA Channel," *ARPANET Satellite System Note 26, NIC Document No. 12735*, Stanford Research Institute, November 15, 1972.
18. Abramson, Norman, "Excess Capacity of a Slotted ALOHA Channel (continued)," *ARPANET Satellite System Note 30, NIC Document No. 13044*, Stanford Research Institute, December 6, 1972.
19. Cacciamani, E. R., Jr., "The Spade System as Applied to Data Communications and Small Earth Station Operation," *COMSAT Technical Review*, Vol. 1, No. 1, Fall 1971.
20. McDonald, J., "Getting Our Communication Satellite Off the Ground," *Fortune*, July 1972.

Packet-switching in a slotted satellite channel*

by LEONARD KLEINROCK and SIMON S. LAM

University of California
Los Angeles, California

INTRODUCTION

Imagine that two users require the use of a communication channel. The classical approach to satisfying this requirement is to provide a channel for their use so long as that need continues (and to charge them for the full cost of this channel). It has long been recognized that such allocation of scarce communication resources is extremely wasteful as witnessed by their low utilization (see for example the measurements of Jackson & Stubbs).[1] Rather than provide channels on a user-pair basis, we much prefer to provide a single high-speed channel to a large number of users which can be shared in some fashion; this then allows us to take advantage of the powerful "large number laws" which state that with very high probability, the demand at any instant will be approximately equal to the sum of the average demands of that population. In this way the required channel capacity to support the user traffic may be considerably less than in the unshared case of dedicated channels. This approach has been used to great effect for many years now in a number of different contexts: for example, the use of graded channels in the telephone industry,[2] the introduction of asynchronous time division multiplexing,[3] and the packet-switching concepts introduced by Baran et al.,[4] Davies,[5] and finally implemented in the ARPA network.[6] The essential observation is that the full-time allocation of a fraction of the channel to each user is highly inefficient compared to the part-time use of the full capacity of the channel (this is precisely the notion of time-sharing). We gain this efficient sharing when the traffic consists of rapid, but short bursts of data. The classical schemes of synchronous time division multiplexing and frequency division multiplexing are examples of the inefficient partitioning of channels.

As soon as we introduce the notion of a shared channel in a packet-switching mode then we must be prepared to resolve conflicts which arise when more than one demand is simultaneously placed upon the channel. There are two obvious solutions to this problem: the first is to "throw out" or "lose" any demands which are made while the channel is in use; and the second is to form a queue of conflicting demands and serve them in some order as the channel becomes free. The latter approach is that taken in the ARPA network since storage may be provided economically at the point of conflict. The former approach is taken in the ALOHA system[7] which uses packet-switching with radio channels; in this system, in fact, *all* simultaneous demands made on the channel are lost.

Of interest to this paper is the consideration of satellite channels for packet-switching. The definition of a packet is merely a package of data which has been prepared by a user for transmission to some other user in the system. The satellite is characterized as a high capacity channel with a fixed propagation delay which is large compared to the packet transmission time (see the next section). The (stationary) satellite acts as a pure transponder repeating whatever it receives and beaming this transmission back down to earth; this broadcasted transmission can be heard by every user of the system and in particular a user can listen to his own transmission on its way back down. Since the satellite is merely transponding, then whenever a portion of one user's transmission reaches the satellite while another user's transmission is being transponded, the two collide and "destroy" each other. The problem we are then faced with is how to control the allocation of time at the satellite in a fashion which produces an acceptable level of performance.

The ideal situation would be for the users to agree collectively when each could transmit. The difficulty is that the means for communication available to these geographically distributed users is the satellite channel itself and we are faced with attempting to control a channel which must carry its own control information. There are essentially three approaches to the solution of this problem. The first has come to be known as a pure "ALOHA" system[7] in which users transmit any time they desire. If, after one propagation delay, they hear their successful transmission then they assume that no conflict occurred at the satellite; otherwise they know a collision occurred and they must retransmit. If users retransmit immediately upon hearing a conflict, then they are likely to conflict again, and so some scheme must be devised for introducing a random retransmission delay to spread these conflicting packets over time.

The second method for using the satellite channel is to "slot" time into segments whose duration is exactly equal to the transmission time of a single packet (we assume constant length packets). If we now require all packets to begin their transmission only at the beginning of a slot, then we

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC-15-69-C-0285.

enjoy a gain in efficiency since collisions are now restricted to a single slot duration; such a scheme is referred to as a "slotted ALOHA" system and is the principal subject of this paper. We consider two models: the first is that of a large population of users, each of which makes a small demand on the channel; the second model consists of this background of users with the addition of one large user acting in a special way to provide an increased utilization of the channel. We concern ourselves with retransmission strategies, delays, and throughput. Abramson[8] also considers slotted systems and is concerned mainly with the ultimate capacity of these channels with various user mixes. Our results and his have a common meeting point at some limits which will be described below.

The third method for using these channels is to attempt to schedule their use in some direct fashion; this introduces the notion of a reservation system in which time slots are reserved for specific users' transmissions and the manner in which these reservations are made is discussed in the paper by Roberts.[9] He gives an analysis for the delay and throughput, comparing the performance of slotted and reservation systems.

Thus we are faced with a finite-capacity communication channel subject to unpredictable and conflicting demands. When these demands collide, we "lose" some of the effective capacity of the channel and in this paper we characterize the effect of that conflict. Note that it is possible to use the channel up to its full rated capacity when only a single user is demanding service; this is true since a user will never conflict with himself (he has the capability to schedule his own use). This effect is important in studying the non-uniform traffic case as we show below.

SLOTTED ALOHA CHANNEL MODELS

Model I. Traffic from many small users

In this model we assume:

(A1) an infinite number of users* who collectively form an independent source

This source generates M packets per slot from the distribution $v_i = \text{Prob}[M=i]$ with a mean of S_0 packets/slot.

We assume that each packet is of constant length requiring T seconds for transmission; in the numerical studies presented below we assume that the capacity of the channel is 50 kilobits per second and that the packets are each 1125 bits in length yielding $T=22.5$ msec. Note that $S_0' = S_0/T$ is the average number of packets arriving per second from the source. Let d be the maximum roundtrip propagation delay which we assume each user experiences and let $R=d/T$ be the number of slots which can fit into one roundtrip propagation time; for our numerical results we assume $d=270$ msec. and so $R=12$ slots. R slots after a transmission, a user will either hear that it was successful or know that it was destroyed. In the latter case if he now retransmits during the next slot interval and if all other users behave likewise, then for sure they will collide again; consequently we shall assume that each user transmits a previously collided packet at random during one of the next K slots, (each such slot being chosen with probability $1/K$). Thus, retransmission will take place either $R+1$, $R+2$, ... or $R+K$ slots after the initial transmission. As a result traffic introduced to the channel from our collection of users will now consist of new packets and previously blocked packets, the total number adding up to N packets transmitted per slot where $pl = \text{Prob}[N=i]$ with a mean traffic of G packets per slot. We assume that each user in the infinite population will have at most one packet requiring transmission at any time (including any previously blocked packets). Of interest to us is a description of the maximum throughput* rate S as a function of the channel traffic G. It is clear that S/G is merely the probability of a successful transmission and G/S is the average number of times a packet must be transmitted until success; assuming

(A2) the traffic entering the channel is an independent process

We then have,

$$S = Gp_0 \tag{1}$$

If in addition we assume,

(A3) the channel traffic is Poisson

then $p_0 = e^{-G}$, and so,

$$S = Ge^{-G} \tag{2}$$

Eq. (2) was first obtained by Roberts[11] who extended a similar result due to Abramson[7] in studying the radio ALOHA system. It represents the ultimate throughput in a Model I slotted ALOHA channel without regard to the delay packets experience; we deal extensively with the delay in the next section.

For Model I we adopt assumption A1. We shall also accept a less restrictive form of assumption A2 (namely assumption A4 below) which, as we show, lends validity to assumption A3 which we also require in this model. Assume,

(A4) the channel traffic is independent over any K consecutive slots

We have conducted simulation experiments which show that this is an excellent assumption so long as $K < R$.

Let,

$$P(z) = \sum_{i=0}^{\infty} p_i z^i \tag{3}$$

$$V(z) = \sum_{i=0}^{\infty} v_i z^i \tag{4}$$

* These will be referred to as the "small" users.

* Note that $S = S_0$ under stable system operation which we assume unless stated otherwise (see below).

Using only assumption A4 and the assumption that M is independent of $N-M$, we find [10] that $P(z)$ may be expressed as

$$\left[\frac{p_1}{K}(1-z)+P\left(1-\frac{1-z}{K}\right)\right]^K V(z)$$

If, further, the source is an independent process (i.e., assumption A1) and is Poisson distributed then $V(z)=e^{-S(1-z)}$, and then we see immediately that,

$$\lim_{K\uparrow\infty} P(z)=e^{-G(1-z)}.$$

This shows that assumption A3 follows from assumptions A1 and A4 in the limit of large K, under the reasonable condition that the source is Poisson distributed.

We have so far defined the following critical system parameters: S_0, S, G, K and R. In the ensuing analysis we shall distinguish packets transmitting in a given slot as being either newly generated or ones which have in the past collided with other packets. This leads to an approximation since we do not distinguish how many times a packet has met with a collision. We have examined the validity of this approximation by simulation, and have found that the correlation of traffic in different slots is negligible, except at shifts of $R+1$, $R+2$, . . . , $R+K$; this exactly supports our approximation since we concern ourselves with the most recent collision. We require the following two additional definitions:

q = Prob[newly generated packet is successfully transmitted]
q_t = Prob[previously blocked packet is successfully transmitted]

We also introduce the expected packet delay D:

D = average time (in slots) until a packet is successfully received

Our principal concern in this paper is to investigate the trade-off between the average delay D and the throughput S.

Model II. Background traffic with one large user

In this second model, we refer to the source described above as the "background" source but we also assume that there is an additional single user who constitutes a second independent source and we refer to this source as the "large" user. The background source is the same as that in Model I and for the second source, we assume that the packet arrivals to the large user transmitter are Poisson and independent of other packets over $R+K$ consecutive slots. In order to distinguish variables for these two sources, we let S_1 and G_1 refer to the S and G parameters for the background source and let S_2 and G_2 refer to the S and G parameters for the single large user. We point out that the identity of this large user may change as time progresses but insist that there be only one such at any given time. We introduce the new variables

$$S=S_1+S_2 \tag{5}$$

$$G=G_1+G_2 \tag{6}$$

S represents the total throughput of the system and G represents the traffic which the channel must support (including retransmissions). We have assumed that the small users may have at most one packet outstanding for transmission in the channel; however the single large user may have many packets awaiting transmission. We assume that this large user has storage for queueing his requests and of course it is his responsibility to see that he does not attempt the simultaneous transmission of two packets. We may interpret G_2 as the probability that the single large user is transmitting a packet in a channel slot and so we require $G_2\leq 1$; no such restriction is placed on G_1 (or on G in Model I).

We now introduce a means by which the large user can control his channel usage enabling him to absorb some of the slack channel capacity; this permits an increase in the total throughput S. The set of packets awaiting transmission by the large user compete among each other for the attention of his local transmitter as follows. Each waiting packet will be scheduled for transmission in some future slot. When a newly generated packet arrives, it immediately attempts transmission in the current slot and will succeed in capturing the transmitter unless some other packet has also been scheduled for this slot; in the case of such a scheduling conflict, the new packet is randomly rescheduled in one of the next L slots, each such slot being chosen equally likely with probability $1/L$. Due to the background traffic, a large user packet may meet with a transmission conflict at the satellite (which is discovered R slots after transmission) in which case, as in Model I, it incurs a random delay (uniformly distributed over K slots) plus the fixed delay of R slots. More than one packet may be scheduled for a future slot and we assume that these scheduling conflicts are resolved by admitting that packet with the longest delay since its previous blocking (due to conflict in transmission or conflict in scheduling) and uniformly rescheduling the others over the next L slots; ties are broken by random selection. We see, therefore, that new packets have the lowest priority in case of a scheduling conflict; however, they seize the channel if it is free upon their arrival. The variable L permits us a certain control of channel use by the large user but does not limit his throughput. We also assume $K, L<R$. Corresponding to q and q_t in Model I, we introduce the success probabilities q_i and q_{it} $(i=1, 2)$ for new and previously blocked packets respectively and where $i=1$ denotes the background source and $i=2$ denotes the single large source. Finally, we choose to distinguish between D_1 and D_2 which are the average number of slots until a packet is successfully transmitted from the background and large user sources respectively.

RESULTS OF ANALYSIS

In this section we present the results of our analysis without proof. The details of proof may be found in Reference 10.

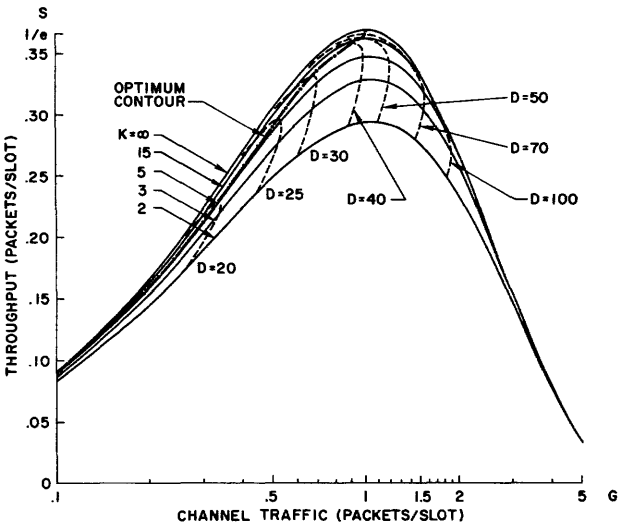


Figure 1—Throughput as a function of channel traffic

Model I. Traffic from many small users

We wish to refine Eq. (2) by accounting for the effect of the random retransmission delay parameter K. Our principal result in this case is

$$S=G\frac{q_t}{q_t+1-q} \tag{7}$$

where

$$q=\left[e^{-G/K}+\frac{G}{K}e^{-G}\right]^K e^{-S} \tag{8}$$

and

$$q_t=\left[\frac{1}{1-e^{-G}}\right]\left[e^{-G/K}-e^{-G}\right]\left[e^{-G/K}+\frac{G}{K}e^{-G}\right]^{K-1} e^{-S} \tag{9}$$

The considerations which led to Eq. (7) were inspired by Roberts[11] in which he developed an approximation for Eq. (9) of the form

$$q_t\cong\frac{K-1}{K}e^{-G} \tag{10}$$

We shall see below that this is a reasonably good approximation. Equations (7–9) form a set of non-linear simultaneous equations for S, q and q_t which must be solved to obtain an explicit expression for S in terms of the system parameters G and K. In general, this cannot be accomplished. However, we note that as K approaches infinity these three equations reduce simply to

$$\lim_{K\to\infty}\frac{S}{G}=\lim_{K\to\infty}q=\lim_{K\to\infty}q_t=e^{-G} \tag{11}$$

Thus, we see that Eq. (2) is the correct expression for the throughput S only when K approaches infinity which corresponds to the case of infinite average delay; Abramson[8] gives this result and numerous others all of which correspond to this limiting case. Note that the large K case avoids the large delay problem if T is small (very high speed channels).

The numerical solution to Eqs. (7–9) is given in Figure 1 where we plot the throughput S as a function of the channel traffic G for various values of K. We note that the maximum throughput at a given K occurs when $G=1$. The throughput improves as K increases, finally yielding a maximum value of $S=1/e=.368$ for $G=1$, $K=$ infinity. Thus we have the unfortunate situation that the ultimate capacity of this channel supporting a large number of small users is less than 37 percent of its theoretical maximum (of 1). We note that the efficiency rapidly approaches this limiting value (of $1/e$) as K increases and that for $K=15$ we are almost there. The figure also shows some delay contours which we discuss below. In Figure 2, we show the variation of q and q_t with K for various values of G. We note how rapidly these functions approach their limiting values as given in Eq. (11). Also on this curve, we have shown Roberts' approximation in Eq. (10) which converges to the exact value very rapidly as K increases and also as G decreases.

Our next significant result is for packet delay as given by

$$D=R+1+\frac{1-q}{q_t}\left[R+1+\frac{K-1}{2}\right] \tag{12}$$

We note from this equation that for large K, the average delay grows linearly with K at a slope

$$\lim_{K\to\infty}\frac{\partial D}{\partial K}=\frac{1-e^{-G}}{2e^{-G}}$$

Using Eq. (11), we see that this slope may be expressed as $G-S/2S$ which is merely the ratio of that portion of transmitted traffic which meets with a conflict to twice the throughput of the channel; since $G-S/2S=\frac{1}{2}(G/S-1)$, we see that the limiting slope is equal to $\frac{1}{2}$ times the average number of times a packet is retransmitted. Little's well-known result[12] expresses the average number ($\bar{n}$) of units (packets in our case) in a queueing system as the product of the average arrival rate ($S_0=S$ in our case) and the average time in system (D). If we use this along with Eqs. (7) and (12), we get

$$\bar{n}=SD=G\left[R+1+\frac{K-1}{2}\right]-S\left[\frac{K-1}{2}\right] \tag{13}$$

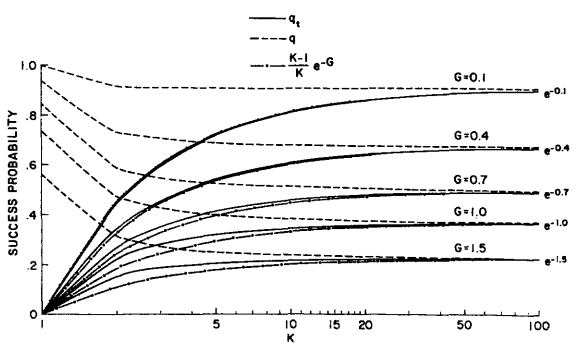


Figure 2—Success probabilities as a function of retransmission delay

In Figure 1 we plot the loci of constant delay in the S, G plane. Note the way these loci bend over sharply as K increases defining a maximum throughput $S_{max}(D)$ for any given value of D; we note the cost in throughput if we wish to limit the average delay. This effect is clearly seen in Figure 3 which is the fundamental display of the tradeoff between delay and throughput for Model I; this figure shows the delay-throughput contours for constant values of K. We also give the minimum envelope of these contours which defines the optimum performance curve for this system (a similar optimum curve is also shown in Figure 1). Note how sharply the delay increases near the maximum throughput $S=0.368$; it is clear that an extreme price in delay must be paid if one wishes to push the channel throughput much above 0.360 and the incremental gain in throughput here is infinitesimal. On the other hand, as S approaches zero, D approaches $R+1$. Also shown here are the constant G contours. Thus this figure and Figure 1 are two alternate ways of displaying the relationship among the four critical system quantities S,G,K, and D.

From Figure 3 we observe the following effect. Consider any given value of S (say at $S=0.20$), and some given value of K (say $K=2$). We note that there are two possible values of D which satisfy these conditions ($D=21.8$, $D=161$). How do we explain this?* It is clear that the lower value is a stable

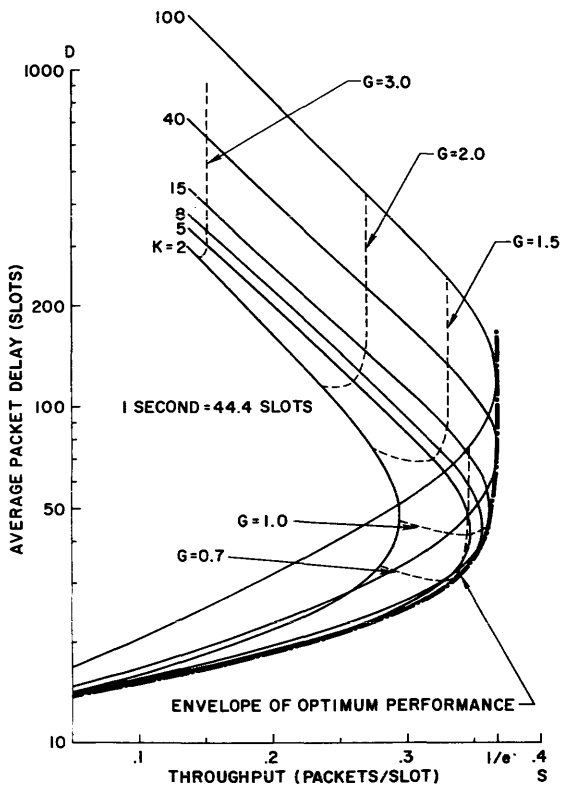


Figure 3—Delay-throughput tradeoff

* This question was raised in a private conversation with Martin Graham (University of California, Berkeley). A simulation of this situation is reported upon in Reference 13.

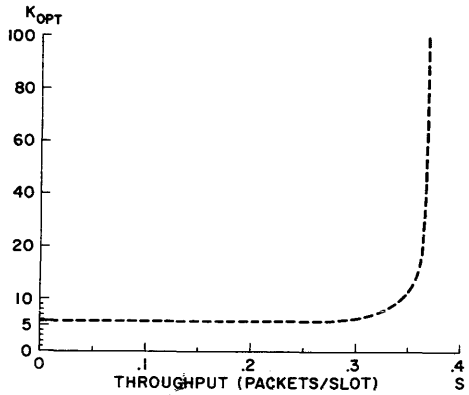


Figure 4—Optimum K

operating point since the system has sufficient capacity to absorb any fluctuation in the rate S_0. Suppose that we now slowly increase S_0 (the source rate); so long as we do not exceed the maximum value of the system throughput rate for this K (say, $S_{max}(K)$), then we see that $S=S_0$ and the system will follow the input. Note that $S_{max}(K)$ always occurs at the intersection of the $G=1$ curve as noted earlier. However, if we attempt to set $S_0>S_{max}(K)$, then the system will go unstable! In fact, the throughput S will drop from $S_{max}(K)$ toward zero as the system accelerates up the constant K contour toward infinite delay! The system will remain in that unfortunate circumstance so long as $S_0>S$ (where now S is approaching zero). All during its demise, the rate at which new packets are being trapped by the system is S_0-S. To recover from this situation, one can set $S_0=0$; then the delay will proceed down the K contour, round the bend at $S_{max}(K)$ and race down to $S=0$. All this while, the backlogged packets are being flushed out of the system. The warning is clear: one must avoid the knee of the K contour. Fortunately, the optimum performance curve does avoid the knee everywhere except when one attempts to squeeze out the last few percent of throughput. In Figure 4, we show the optimum values of K as a function of S. Thus, we have characterized the tradeoff between throughput and delay for Model I.

Model II. Background traffic with one large user

In this model the throughput equation is similar to that given in Eq. (7), namely,

$$S_i=G_i\frac{q_{it}}{q_{it}+1-q_i} \qquad i=1,2 \tag{14}$$

the quantities q_{it} and q_i are given in the appendix. Similarly the average delays for the two classes of user are given by

$$D_1=R+1+\frac{1-q_1}{q_{1t}}\left[R+1+\frac{K-1}{2}\right] \tag{15}$$

$$D_2=R+1+\frac{1-q_2}{q_{2t}}\left[R+1+\frac{K-1}{2}\right]+\frac{L+1}{2}\left[E_n+\frac{1-q_2}{q_{2t}}E_t\right] \tag{16}$$

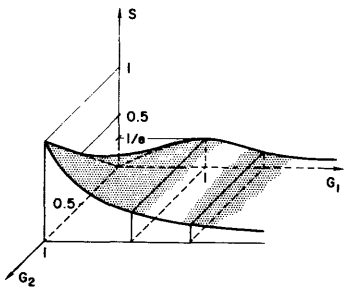


Figure 5—Throughput surface

where E_n and E_t are given in the appendix. It is easy to show that as K, L approach infinity,

$$q_1 = q_{1t} = e^{-G_1}(1-G_2) \tag{17}$$

$$S_1 = G_1 e^{-G_1}(1-G_2) \tag{18}$$

$$q_2 = q_{2t} = e^{-G_1} \tag{19}$$

$$S_2 = G_2 e^{-G_1} \tag{20}$$

$$S = (G - G_1 G_2) e^{-G_1} \tag{21}$$

where we recall $G = G_1 + G_2$ and $S = S_1 + S_2$. From these last equations or as given by direct arguments in an unpublished note by Roberts, one may easily show that at a constant background user throughput S_1, the large user throughput S_2 will be maximized when

$$G = G_1 + G_2 = 1 \tag{22}$$

This last is a special case of results obtained by Abramson in Reference 8 and he discusses these limiting cases at length for various mixes of users. We note that,

$$\frac{\partial S}{\partial G_2} = e^{-G_1}(1-G_1) \tag{23}$$

$$\frac{\partial S}{\partial G_1} = -e^{-G_1}(G - G_1 G_2 - 1 + G_2) \tag{24}$$

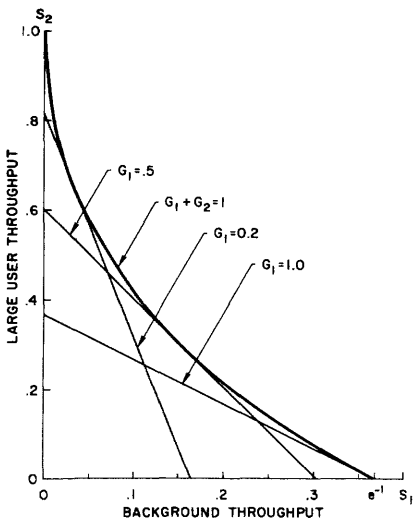


Figure 6—Throughput tradeoff

In Figure 5 we give a qualitative diagram of the 3-dimensional contour for S as a function of G_1 and G_2. We remind the reader that this function is shown for the limiting case K, L approaching infinity only. From our results we see that for constant $G_1<1$, S increases linearly with G_2 ($G_2<1$). For constant $G_1>1$, S decreases linearly as G_2 increases. In addition, for constant $G_2<\frac{1}{2}$, S has a maximum value at $G_1 = 1-2G_2/1-G_2$. Furthermore, for constant $G_2>\frac{1}{2}$, S decreases as G_1 increases and therefore the maximum throughput S must occur at $S=G_2$ in the $G_1=0$ plane.

The optimum curve given in Eq. (22) is shown in the S_1,S_2 plane in Figure 6 along with the performance loci at constant G_1. We note in these last two figures that a channel throughput equal to 1 is achievable whenever the background traffic drops to zero thereby enabling $S=S_2=G_2=1$; this corresponds to the case of a single user utilizing the satellite channel at its maximum throughput of 1. Abramson [8] discusses a variety of curves such as those in Figure 6; he considers the generalization where there may be an arbitrary number of background and large users.

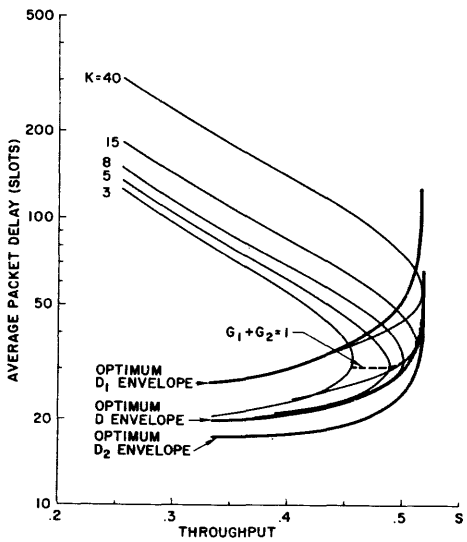


Figure 7—Delay-throughput tradeoff at $S_1=0.1$

In the next three figures, we give numerical results for the finite K case; in all of these computations, we consider only the simplified situation in which $K=L$ thereby eliminating one parameter. In Figure 7 we show the tradeoff between delay and throughput similar to Figure 3. (Note that Figure 5 is similar to Figure 1.) Here we show the optimum performance of the average delay $D = S_1 D_1 + S_2 D_2 / S$ along with the behavior of D at constant values of K and $S_1 = 0.1$ (note the instability once again for overloaded conditions). Also shown are minimum curves for D_1 and D_2, which are obtained by using the optimum K as a function of S. If we are willing to reduce the background throughput from its maximum at $S_1 = 0.368$, then we can drive the total throughput up to approximately $S = 0.52$ by introducing additional traffic from the large user. Note that the minimum D_1 curve is much higher than the minimum D_2 curve. Thus our net gain in

channel throughput is also at the expense of longer packet delays for the small users. Once again, we see the sharp rise near saturation.

In Figure 8, we display a family of optimum D curves for various choices of S_1 as a function of the total throughput S. We also show the behavior of Model I as given in Figure 3. Note as we reduce the background traffic, the system capacity increases slowly; however, when S_1 falls below 0.1, we begin to pick up significant gains for S_2. Also observe that each of the constant curves "peels off" from the Model I curve at a value of $S=S_1$. At $S_1=0$, we have only the large user operating with no collisions and at this point, the optimal value of L is 1. This reduces to the classical queueing system with Poisson input and constant service time (denoted $M/D/1$) and represents the *absolute optimum performance* contour for any method of using the satellite channel when the input is Poisson; for other input distributions we may use the $G/D/1$ queueing results to calculate this absolute optimum performance contour.

In Figure 9, we finally show the throughput tradeoffs between the background and large users. The upper curve shows the absolute maximum S at each value of S_1; this is a clear display of the significant gain in S_2 which we can achieve if we are willing to reduce the background throughput. The middle curve (also shown in Figure 6 and in Reference 8) shows the absolute maximum value for S_2 at each value of S_1. The lowest curve shows the net gain in system capacity as S_1 is reduced from its maximum possible value of $1/e$.

CONCLUSIONS

In this paper we have analyzed the performance of a slotted satellite system for packet-switching. In our first model, we have displayed the trade-off between average delay and average throughput and have shown that in the case of traffic consisting of a large number of small users, the limiting

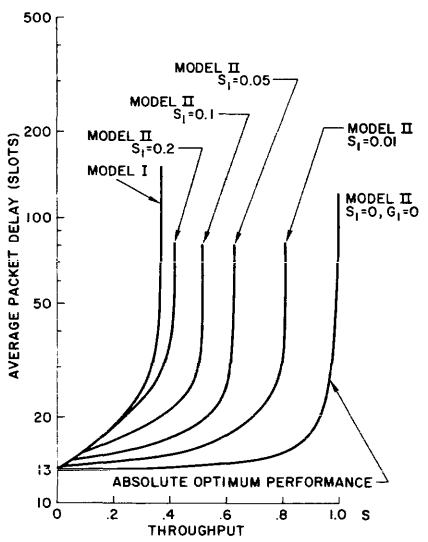


Figure 8—Optimum delay-throughput tradeoffs

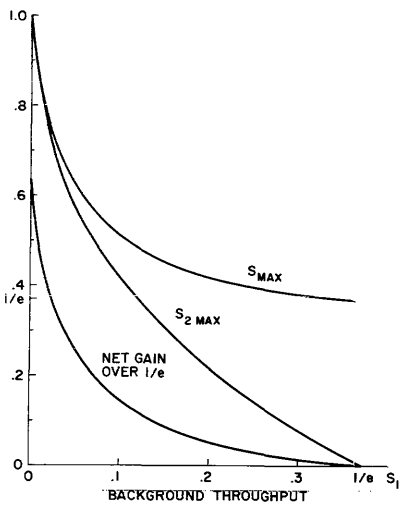


Figure 9—Throughput countours

throughput of the channel ($1/e$) can be approached fairly closely without an excessive delay. This performance can be achieved at relatively small values of K which is the random retransmission delay parameter. However, if one attempts to approach this limiting capacity, not only does one encounter large delays, but one also flirts with the hazards of unstable behavior.

In the case of a single large user mixed with the background traffic, we have shown that it is possible to increase the throughput rather significantly. The qualitative behavior for this multidimensional trade-off was shown and the numerical calculations for a given set of parameters were also displayed. The optimum mix of channel traffic was given in Eq. (22) and is commented on at length in Abramson's paper.[8] We have been able to show in this paper the relationship between delay and throughput which is an essential trade-off in these slotted packet-switching systems.

In Roberts' paper[9] he discusses an effective way to reserve slots in a satellite system so as to predict and prevent conflicts. It is worthwhile noting that another scheme is currently being investigated for packet-switching systems in which the propagation delay is small compared to the slot time, that is, $R=d/T\ll 1$. In such systems it may be advantageous for a user to "listen before transmitting" in order to determine if the channel is in use by some other user; such systems are referred to as "carrier sense" systems and seem to offer some interesting possibilities regarding their control. For satellite communications this case may be found when the capacity of the channel is rather small (for example, with a stationary satellite, the capacity should be in the range of 1200 bps for the packet sizes we have discussed in this paper). On the other hand, a 50 kilobit channel operating in a ground radio environment with packets on the order of 100 or 1000 bits lend themselves nicely to carrier sense techniques.

In all of these schemes one must trade off complexity of implementation with suitable performance. This performance must be effective at all ranges of traffic intensity in that no unnecessary delays or loss of throughput should occur due to

complicated operational procedures. We feel that the slotted satellite packet-switching methods described in this paper and the reservation systems for these channels described in the paper by Roberts do in fact meet these criteria.

REFERENCES

1. Jackson, P. E., Stubbs, C. D., "A Study of Multi-access Computer Communications," Spring Joint Computer Conf., *AFIPS Conf. Proc.*, Vol. 34, 1969, pp. 491-504.
2. Syski, R., *Introduction to Congestion in Telephone Systems*, Oliver & Boyd, Edinburgh and London, 1960.
3. Chu, W. W., "A Study of Asynchronous Time Division Multiplexing for Time-Sharing Computer Systems," Spring Joint Computer Conf., *AFIPS Conf. Proc.*, Vol. 35, 1969, pp. 669-678.
4. Baran, P., Boehm, S., and Smith, P., "On Distributed Communications," series of 11 reports by Rand Corp., Santa Monica, Calif., 1964.
5. Davies, D. W., "The Principles of a Data Communication Network for Computers and Remote Peripherals," *Proc. IFIP Hardware*, Edinburgh, 1968, D11.
6. Roberts, L. G., "Multiple Computer Networks and Inter-Computer Communications," *ACM Symposium on Operating Systems*, Gatlinburg, Tenn., 1967.
7. Abramson, N., "The ALOHA System—Another Alternative for Computer Communications," Fall Joint Computer Conf., *AFIPS Conf. Proc.*, Vol. 37, 1970, pp. 281-285.
8. Abramson, N., "Packet Switching with Satellites," these proceedings.
9. Roberts, L. G., "Dynamic Allocation of Satellite Capacity through Packet Reservation," these proceedings.
10. Kleinrock, L., Lam, S. S., Arpanet Satellite System Notes 12 (NIC Document #11294); 17 (NIC Document #11862); 25 (NIC Document #12734); and 27 (NIC Document #12756), available from the ARPA Network Information Center, Stanford Research Institute, Menlo Park, California.
11. Roberts, L., Arpanet Satellite System Notes 8 (NIC Document #11290) and 9 (NIC Document #11291), available from the ARPA Network Information Center, Stanford Research Institute, Menlo Park, California.
12. Kleinrock, L., *Queueing Systems: Theory and Applications*, to be published by Wiley Interscience, New York, 1973.
13. Rettberg, R., Arpanet Satellite System Note 11 (NIC Document #11293), available from the ARPA Network Information Center, Stanford Research Institute, Menlo Park, California.

APPENDIX

Define $G_S \triangleq$ Poisson arrival rate of packets to the transmitter of the large user

$$= S_2[1+E_n+E_2(1+E_t)] \tag{A.1}$$

The variables q_i, q_{it} $(i=1, 2)$ in Eqs. (14–16) are then given as follows (see Reference 10 for details of the derivations):

$$q_1 = (q_0)^K (q_h)^L e^{-S} \tag{A.2}$$

$$q_{1t} = (q_0)^{K-1} q_{1c} (q_h)^L e^{-S} \tag{A.3}$$

where

$$q_0 = e^{-G_1/K} + \frac{1}{K}\left[(1-e^{-G_s})(e^{-G_1}-e^{-G_1/K}) + G_1 e^{-(G_1+G_s)}\right] \tag{A.4}$$

$$q_h = \begin{cases} (G_s+1)e^{-G_s} & L=1 \\ \dfrac{1}{L-1}(Le^{-G_s/L}-e^{-G_s}) & L\geq 2 \end{cases} \tag{A.5}$$

$$q_{1c} = \frac{1}{1-e^{-(G_1+G_s)}}\left[e^{-G_1/K}\left(1-\frac{1-e^{-G_s}}{K}\right)-e^{-(G_1+G_s)}\right] \tag{A.6}$$

Let us introduce the following notation for events at the large user:

SS = scheduling success (capture of the transmitter)
SC = scheduling conflict (failure to capture transmitter)
TS = transmission success (capture of a satellite slot)
TC = transmission conflict (conflict at the satellite)
NP = newly generated packet

Then,

$$q_2 = \frac{r_n + r_s E_n}{1+E_n} \tag{A.7}$$

$$q_{2t} = \frac{r_t + r_s E_t}{1+E_t} \tag{A.8}$$

where

$$E_n \triangleq \text{average number of } SC \text{ events before an } SS \text{ event conditioning on } NP = \frac{1-a_n}{a_s} \tag{A.9}$$

$$E_t \triangleq \text{average number of } SC \text{ events before an } SS \text{ event conditioning on } TC = \frac{1-a_t}{a_s} \tag{A.10}$$

The variables a_i, r_i $(i=n, t, s)$ are defined and given below:

$$a_n \triangleq \text{Prob}\,[SS/NP] = \left(\frac{q_0}{q}\right)^K (q_h)^L \left(\frac{1-e^{-S_2}}{S_2}\right) \tag{A.11}$$

$$r_n \triangleq \text{Prob}\,[TS/SS, NP] = q^K e^{-S_1} \tag{A.12}$$

$$a_t \triangleq \text{Prob}\,[SS/TC] = \frac{1}{K}\frac{1-(q_0/q)^K}{1-q_0/q} \tag{A.13}$$

$$r_t \triangleq \text{Prob}\,[TS/SS, TC] = q^{K-1} q_{2c} e^{-S_1} \tag{A.14}$$

$$a_s \triangleq \text{Prob}\,[SS/SC] = \left(\frac{q_0}{q}\right)^K \frac{q_{sc}}{L}\frac{1-(q_h)^L}{1-q_h} \tag{A.15}$$

$$r_s \triangleq \text{Prob}\,[TS/SS, SC] = q^K e^{-S_1} \tag{A.16}$$

where

$$q = e^{-G_1/K} + \frac{G_1}{K} e^{-(G_1+G_s)} \tag{A.17}$$

$$q_{2c} = \frac{e^{-G_1/K} - e^{-G_1}}{1-e^{-G_1}} \tag{A.18}$$

$$q_{sc} = \frac{1}{G_s - 1 + e^{-G_s}}\left(\frac{L}{L-1}\right)^2\left[G_s\left(1-\frac{1}{L}\right)e^{-G_s/L} - e^{-G_s/L} + e^{-G_s}\right] \tag{A.19}$$

Dynamic allocation of satellite capacity through packet reservation

by LAWRENCE G. ROBERTS

Department of Defense
Arlington, Virginia

INTRODUCTION

If one projects the growth of computer communication networks like the ARPANET[1,2,3,4] to a worldwide situation, satellite communication is attractive for intercommunicating between the widespread geographic areas. For this variable demand, multi-station, data traffic situation, satellites are uniquely qualified in that they are theoretically capable of statistically averaging the load in total at the satellite rather than requiring each station or station-pair to average the traffic independently. However, very little research has been done on techniques which permit direct multi-station demand access to a satellite for data traffic. For voice traffic statistics, COMSAT Laboratories has developed highly efficient techniques; the SPADE[5] system currently installed in the Atlantic permitting the pooled use of 64KB PCM voice channels on a demand basis, and the MAT-1[6] TDMA (Time Division Multiple-Access) experimental system. Both systems permit flexible demand assignment of the satellite capacity, but on a circuit-switched basis designed to interconnect a full duplex 64KB channel between two stations for minutes rather than deliver small blocks of data here and there. This work forms the technical base for advanced digital satellite communication, and provides a very effective means for moving large quantities of data between two points. However, for short interactive data traffic between many stations, new allocation techniques are desirable.

TRAFFIC MODEL

In order to evaluate the performance of any new technique for dynamic assignment of satellite capacity and compare it with other techniques, a complete model of the data traffic must be postulated. Given the model, each technique can be analyzed and its performance computed for any traffic load or distribution. Although it is difficult to fully represent the complete variation in traffic rates normal in data traffic, the following model describes the basic nature of data traffic which might arrive at each satellite station from a local packet network.

There are Poisson arrivals of both single packets (1270 bits including the header) and multi-packet blocks (8 packets) at each station. The overall Poisson arrival rate for both is L with a fraction F of single packets and the remainder multi-packets. For simplicity, the arrival rates at all stations are stationary and equal. This is not completely representative of normal data traffic but for the assignment techniques of interest, non-stationary and unequal arrival rates will produce nearly identical performance to the stationary case. Techniques which subdivide the satellite capacity in a preassigned manner would be seriously hurt by non-stationary traffic rates but the poor performance of these systems will be demonstrated, at least in part, by their inability to handle Poisson packet arrivals effectively. The average station traffic in packets per second is:

$$T = L(F + 8(1 - F)) \quad (1)$$

The destination of this traffic is equally divided between all of the other stations.

For a truly reliable data communications network, each packet or block should be acknowledged as having been correctly received. Positive error control using acknowledgments and retransmissions is very important for data traffic. Thus, acknowledgment traffic must be added to the station traffic. To achieve rapid recovery from errors there must be one small packet (144 bits) sent for each packet or block sent. This traffic is administrative overhead and will not be counted when computing the channel utilization.

The analytic results presented later in the paper are all for equal arrival rates for single packets and multi-packets ($F = .5$). Other values of F have been examined as well as cases where the input traffic contains small (144 bit) data packets as well. The detailed effect of these variations is not sufficiently pronounced to consider here, however. For comparing techniques the equal arrival distribution is quite representative.

ARPANET experience indicates that the data traffic one can expect is proportional to the total dollar value of computer services being bought or sold through the network. The total traffic generated by one dollar of computer activity is about 315 packets, half going each way.[3] Thus, \$200K/year of computer activity within a region produces 2KB of traffic, of which IKB is leaving the region. Within the next few years it is probable that the computer services exchanged internationally will be between \$50K/country and \$2M/country which suggests that the traffic levels, T,

to consider are from .25KB to 10KB. For domestic satellite usage the dollar flow would be far greater than this if the regions are ones like the east and west coast. However, if small stations become economically attractive, the individual user complexes or computer sites will have traffic levels well within this range. Therefore, several of the analytic results presented are for a station traffic of T = 1KB. This corresponds to one packet or multi-packet arriving every 4.5 seconds, on the average. It is extremely important to note the infrequency of this, considering that the block must be delivered within less than a second. Even at 10KB, with arrivals every .45 seconds, each arrival must be treated independently, not waiting for a queue to build up if rapid response is to be maintained. Only after the individual traffic exceeds 50KB is there significant smoothing and uniformity to the station's traffic flow. Thus, it is quite important to devise techniques which do not depend on this smoothing at each station if stations with under $10M of remote computer activity are to be served economically.

CHANNELIZED SATELLITE TRANSMISSION TECHNIQUES

FDM—FULL interconnection

The most common technique in use today is for each pair of stations which have traffic to lease a small full duplex data channel directly. If this technique were used for a large net of N stations, it would require N (N−1) half duplex channels, each large enough to provide the desired delay response. The total satellite bandwidth required is the sum of the N(N−1) individual requirements plus 2KHz* per channel (minimum) for guardbands. However, since the channels are dedicated, variable packet sizes can be handled and the small acknowledgments fit in efficiently.

FDM—Store and forward star

Since it is clearly very costly for full interconnection, store and forward is an obvious alternative. With short, leased ground lines, the ARPANET very effectively uses this technique, but since each hop adds at least .27 sec due to the propagation delay, it is important to minimize the number of hops. Thus a star design is probably as good an example of this technique as any. The total number of channels for a star is N−1. The delay is the two hop total plus any switch delay (herein presumed zero and of infinite capacity).

TDMA

Since all stations could theoretically hear all the transmissions, a store and forward process is really unnecessary if each packet has an address and its destination can receive it. Further, the guardbands required for FDM can be eliminated if Time Division Multiple Access techniques are used. Instead, an 80 bit start up synchronization leader is required. This increases the small acknowledgment packets to 225 bits and the normal packets to 1350 bits, a 7.6 percent overhead. For this type of data traffic a strict alternation of time slot ownership between the stations was evaluated. All slots are the same size, 1350 bits, except for small acknowledgment packets which are packed in at the necessary intervals. Thus, each station has one Nth of the channel capacity and can use it freely to send to any station. Each station must examine all packets for those addressed to itself. To adapt to unequal or non-stationary traffic levels, there are many techniques[6] for slowly varying the channel split.

ALOHA

Instead of preassigning time slots to stations and often having them be unused, in the ALOHA system they are all freely utilized by any station with traffic. When there are many stations this reduces the delay caused in waiting for your own slot, but introduces a channel utilization limit of 36 percent to insure that conflicts are not too frequent. When conflicts do occur the sum check clearly indicates it and both stations retransmit. A very complete treatment of this technique is presented in the papers by Abramson[7] and Kleinrock and Lam.[8] For the comparison curves presented here, an approximation to the precise delay calculation was used and the possibilities of improved performance due to excess capacity were ignored. Thus, the ALOHA results are slightly conservative.

RESERVATION SYSTEM

In order to further improve the efficiency of data traffic distribution via satellite, the following reservation system is proposed. As with TDMA and ALOHA the satellite channel is divided into time slots of 1350 bits each. However, after every M slots one slot is subdivided into V small slots. The small slots are for reservations and acknowledgments, to be used on a contention basis with the ALOHA technique. The remaining M large slots are for RESERVED data packets. When a data packet or multi-packet block arrives at a station it transmits a reservation in a randomly selected one of the V small slots in the next ALOHA group. The reservation is a request for from one to eight RESERVED slots. Upon seeing such a reservation each station adds the number of slots requested to a count, J, the number of slots currently reserved. The originating station has now blocked out a sequence of RESERVED slots to transmit his packets in. Thus, there is one common queue for all stations and by broadcasting reservations they can claim space on the queue. It is not necessary for any station but the originating station to remember which space belongs to whom, since the only requirement is that no one else uses the slots.

* Two KHz is the minimal possible channel separation determined by oscillator stability for current INTELSAT IV equipment based on a private communication with *E.* Cacciamani, COMSAT Laboratories. Actual guardbands in use are wider.

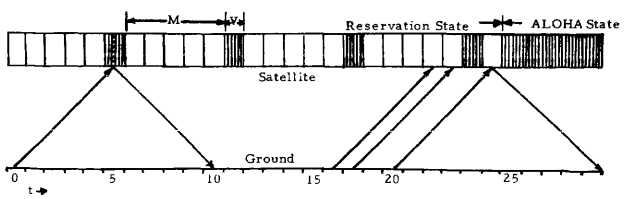
Referring to Figure 1, a reservation for three slots is transmitted at t=0 so as to fall in an ALOHA slot at t=5. If a conflict occurs, the originating station will determine the sum check is bad at t=10 and retransmit the reservation. However, if it is received correctly at t=10 and assuming the current queue length is thirteen, the station computes that it can use the slots at t=21, 22 and 24. It does this by transmitting at t=16, 17 and 19. By t=30 the entire block of three packets has been delivered to their destination. If no other reservations have been received by t=19 the queue goes to zero at this point and the channel reverts to a pure ALOHA state until the next valid reservation is received.

Reservations

To maintain coordination between all the stations, it is necessary and sufficient that each reservation which is received correctly by any station is received correctly by all the stations. This can be assured even if the channel error rate is high by properly encoding the reservation. The simplest strategy is to use the standard packet sum check hardware, and send three independently sumchecked copies of the reservation data. A reservation requires 24 bits of information and with the sum check is 48 bits. Three of these together with the 80 bit sync sequence made a 224 bit packet. Given this size for the small slot and 1350 bits for the large slot, we can pack six reservations in the large slot space; therefore, V=6. If the channel error rate is 10^{-5} and there are 1000 stations, the probability that one or more of the stations will have errors in all three sections is approximately 1000 $(48\times10^{-5})^3$ or 10^{-7}. With a 1.5MB channel this is one error every three days, a very tolerable rate considering the only impact is to delay some data momentarily. If the reservation were not triplicated, however, the probability of an error is .48, sufficient to totally confuse all the stations.

Channel states

There are two states, ALOHA and RESERVED. On start up and every time thereafter when the reservation queue goes to zero, the channel is in the ALOHA state. In this state, all slots small and the ALOHA mode of transmission is used. Reservations, acknowledgments and even small data packets can be sent using the 224 bit slots. However, the first successful reservation causes the RESERVED state to begin. Let us define Z to be the channel rate in large slots per second and R to be the number of large slots per round trip (R=.27Z). Then, considering time as viewed from the satellite, the data packets associated with the first reservation should be transmitted so as to start R+1 large slots after the reservation. To avoid confusion, M is kept constant for the entirety of each RESERVED state but it is allowed to change each time the state is entered. The initial reservation which starts the state contains a suggested new value for M. This value is used until the state terminates. The determination of M will be considered later.



RESERVATION SYSTEM CHANNEL DIVISION
50KB channel (R = 10 slots per round trip), M = 5, V = 6

Figure 1

Channel utilization

The traffic of small packets (reservations, acknowledgments) is twice the overall arrival rate (NL) since every data block requires a reservation and an acknowledgment. If we assume that the arrival rate of these small packets is independent of the state (a good approximation since they are fully independent at both low and high traffic levels where the average duration of one of the states is short compared to R), then:

Small Slot Channel Utilization in ALOHA State:

$$S_1 = 2NL/ZV \quad (2)$$

Small Slot Channel Utilization in RESERVED State:

$$S_2 = 2NL(M+1)/ZV \quad (3)$$

The channel utilization for large slots must be computed as if the channel were always in the RESERVED state since the ALOHA state is a *result* of the non-utilization of the reserved slots, not the cause. Thus:

Large Slot Channel Utilization: $S_3 = BNL(M+1)/MZ$

Where, average block size: $B = F+8(1-F)$

$$B = 4.5 \quad (4)$$

For the ALOHA transmissions, the channel utilization is related to the actual transmission rate (G) by the relation (see references 7 and 8):

$$\text{ALOHA State:} \quad S_1 = AG_1e^{-G1}$$
$$\text{RESERVED State:} \quad S_2 = AG_2e^{-G2}$$

These relations must be solved for G by iteration since S is the known quantity. The correction constant, A, depends on the retransmission randomization technique and R, but is always between .8 and 1.0. As a result of these relations the maximum useful ALOHA throughput is S=A/e. An empirically derived approximation* to A used for this analysis was (K=retransmission randomization period in slots):

$$A = \frac{K-1}{K} \quad \text{where } K = 2.3\sqrt{R}$$

* For an accurate and more detailed solution to the effect of a fixed retransmission delay, refer to Reference 8.

Small packet delay

The average fraction of time the system is in the RESERVED state is equal to the large slot channel utilization, S_3, since that is the fraction of time the reserved packets are being sent. Thus, if we compute the delay for the small packets in both states a weighted average can be taken, using S_3, to obtain the average delay.

ALOHA State:

Initial queueing delay: $$W_1=\frac{G_1/N}{2V(1-G_1/N)}$$

Retransmissions: $$H_1=\frac{1-Ae^{-G_1}}{Ae^{-G_1}}$$

Small Packet Delay: $$D_1=\frac{R+1.5/V+W_1+H_1}{Z}\times\frac{(R+W_1+1/V+K/2V)}{Z}$$

RESERVED State:

Initial queueing delay: $$W_2=\frac{(M+1)G_2/N}{2V(1-G_2/N)}$$

Retransmissions: $$H_2=\frac{1-Ae^{-G_2}}{Ae^{-G_2}}$$

Small Packet Delay: $$D_2=\frac{R+1.5V+M/2+W_2}{Z}+\frac{H_2(R+W_2+1/V+\frac{K(M+1)}{2V}}{Z}$$

Now, the overall average small packet delay can be determined:

Overall Small Packet Delay: $$D^2=D_1(1-S_3)+D_2S_3 \qquad (5)$$

Large packet and block delay

For the reserved packets, the delay has three components; the reservation delay (D_S), the central queueing delay and the transmission-propagation delay of the packet or block. For a block of B packets where the general load is the defined traffic distribution the delay is:

Average Delay for reserved: $$D_r=\frac{ZD_S+R+B\frac{(M+1)}{M}+\frac{YS_3(M+1)}{2M(1-S_3)}}{Z} \qquad (6)$$

Where: Y = 7.2 packets (second moment of block size/avg. block size)

and: B = 4.5 packets (average block size)

Determination of M

An optimal value for M can now be determined numerically for any given channel and traffic load. However, this value is not very critical at low channel loading factors. It is only when the channel is operating near peak capacity that M affects the delay more than a few percent. Since M cannot be changed rapidly it is desirable to set M to the value which optimizes the channel capacity and thereby minimizes the delay at peak load. For peak capacity, both the small and large slot portions of the channel in the RESERVED state should be fully loaded. This occurs when $S_2=A/e$ and $S_3=1$. Doing this and solving equations (3) and (4) for the arrival rate, L, gives us:

$$L=\frac{ZVA}{2eN(M+1)}=\frac{ZM}{BN(M+1)}$$

Solving for M:

$$M=\frac{AV}{2e}B \text{ rounded up to nearest integer}$$

for B = 4.5, V = 6: M = 5

If this peak capacity value for M is always used the delay is within 10 percent of optimal and the system is quite stable. As can be seen, the only traffic parameter M depends on is B, the average block size. M can be adjusted by the stations if the channel is monitored and the fractions of each type of packet sent are measured. From these fractions it is easy to determine M.

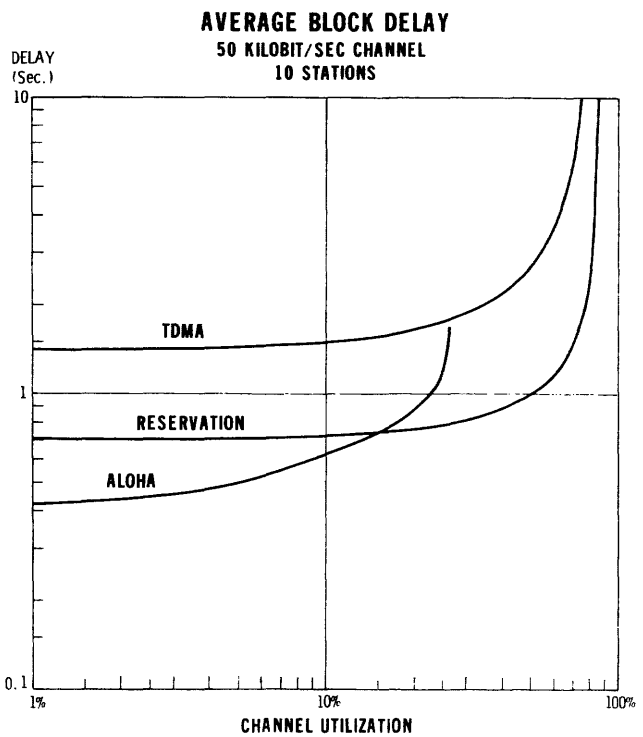


Figure 2

Performance

Now it is possible to determine the delay given the traffic distribution (F,B), number of stations (N), and input arrival rate (L). One common way to examine performance is by plotting delay versus the channel utilization for a fixed channel. The channel utilization, C, is the ratio of the good data delivered to the new channel speed:

Channel Utilization: $C = NLB/Z$

Figure 2 shows the delay vs. C for the TDMA, ALOHA, and Reservation techniques. The traffic distribution is as

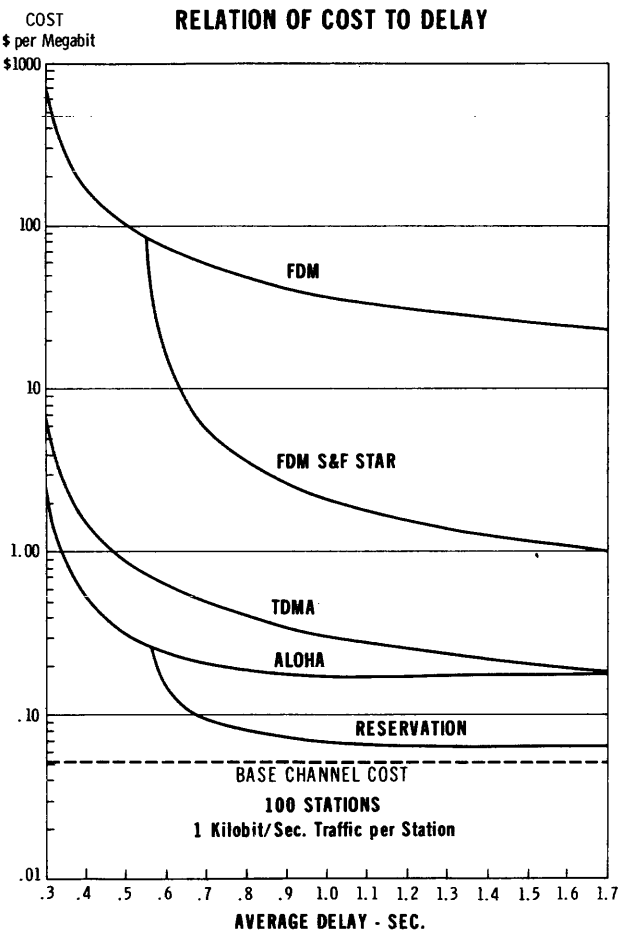


Figure 3

previously defined; half single packets and half blocks of eight.

This type of presentation is not the best for deciding what technique to use for a specific job, but it does show the general behavior of the systems for a fixed channel size, as the traffic load varies.

In order to really compare the cost of the various techniques to do a certain job, it is necessary to set the traffic level, number of stations, and the delay permissible. Then, for each technique, the channel size required to achieve the delay constraint can be searched for. To make the presentation more meaningful the cost of this channel per

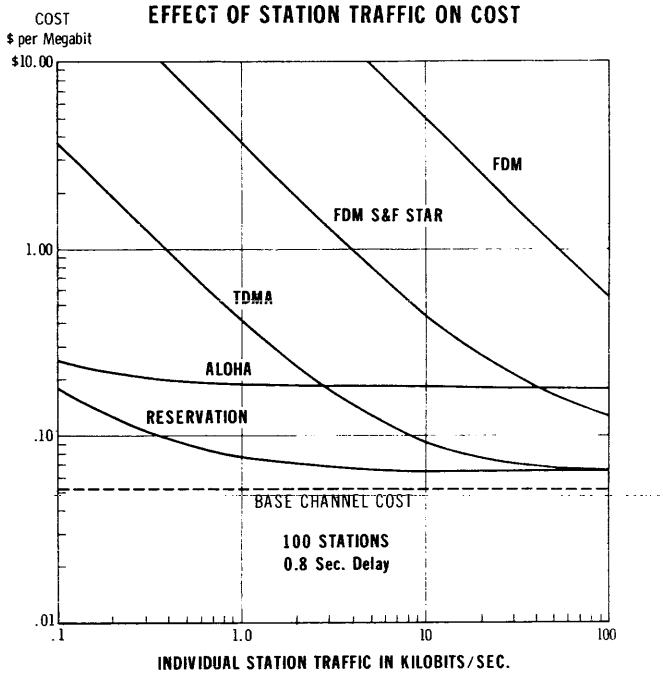


Figure 4

megabit of traffic can then be determined using as a price basis the current tariffed price of the 50KB INTELSAT IV channel (45 KHz) used in the ARPANET between California and Hawaii. It is presumed that any bandwidth could be purchased for the same price per KHz. Converting the cost to dollars per megabit permits easy comparison with the cost in the current ARPANET where distributed leased line capacity can be achieved for $.10 per megabit.

Figures 3, 4 and 5 show communications cost as a function of the three variables; delay, traffic and number of

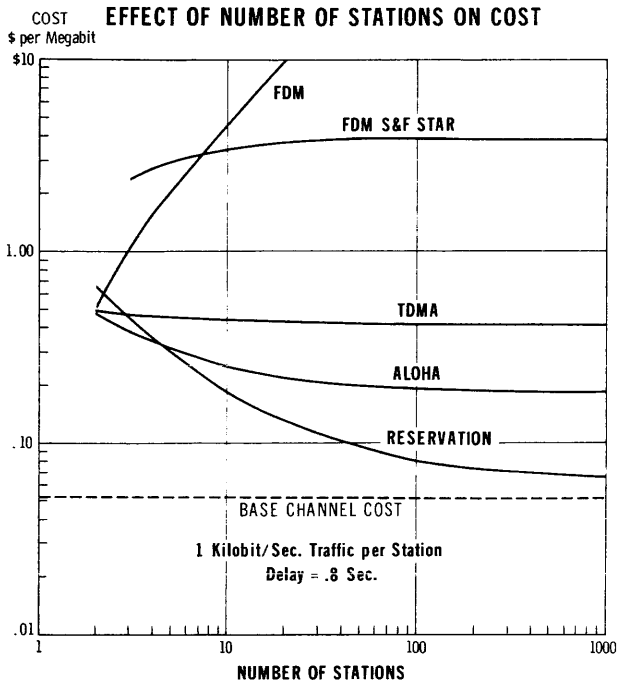


Figure 5

stations. Examining Figure 3 it is clear that if a delay of less than two round trips (.54 sec) is required, the ALOHA system is superior. However, the cost for .4 sec service is over 6 times that of .8 sec service (using the reservation technique). It is also clear that delays of more than .8 sec are not necessary and save very little money. Figure 4 shows that as the individual station traffic is increased to 50KB or higher, TDMA becomes almost as good as the reservation system since there is sufficient local averaging of traffic. Similarly, at this same traffic level FDM—Store and Forward achieves its maximum efficiency but due to sending each packet twice its asymptotic cost is twice that of TDMA or Reservation. These traffic levels for each station are unrealistically high, however, and the flat performance of ALOHA and the Reservation System is vastly preferable since the cost of data communications to small stations is the same as for large stations. Finally, Figure 5 shows the effect of adding stations to the net. With FDM the cost grows out of bounds quickly whereas the reservation technique improves its efficiency until the total traffic from all stations exceeds 100KB. Below 5KB total traffic ALOHA is superior, but this is not a very important case. For large numbers of stations at 1KB traffic per station and .8 seconds delay the reservation system is 3 times cheaper than ALOHA, 6 times cheaper than TDMA, and 56 times cheaper than FDM Store and Forward.

CONCLUSIONS

The reservation technique presented here is one of several techniques which have been developed recently to take full advantage of the multi-access capabilities of satellites for data traffic.[9,10] Both the ALOHA technique and the reservation system depend for their efficiency on the total multi-station traffic rather than the individual station traffic as does TDMA and FDM Store and Forward. The performance improvement reflects this with the reservation system being up to 10 times as efficient as TDMA for small station traffic levels. The worst possible technique for data traffic is pure FDM links between each station pair since this is only efficient if all *pairs* of stations have 50KB of traffic, driving the cost out of bounds for normal usage. The reservation system is also a factor of 3 more efficient than the ALOHA system and for large (100KB) traffic levels achieves almost perfect utilization of the channels.

REFERENCE

1. Roberts, L. G., Wessler, B., "Computer Network Development to Achieve Resource Sharing," *AFIPS Spring Joint Computer Conference Proceedings*, pp. 543-549, 1970.
2. Heart, F., Kahn, R. E., Ornstein, S., Crowther, W., Walden, D., "The Interface Message Processor for the ARPA Computer Network," *AFIPS Spring Joint Computer Conference Proceedings*, pp. 551-567, 1970.
3. Roberts, L. G., "Network Rationale: A 5-Year Reevaluation," *Proceedings of COMPCON 73*, February 1973.
4. Kahn, R. E., "Resource-Sharing Computer Communications Networks," *Proceedings of IEEE*, pp. 1397-1407, November 1972.
5. Cacciamani, E. R., "The Spade System as Applied to Data Communications and Small Earth Station Operation," *COMSAT Technical Review*, Vol. 1, No. 1, Fall 1971.
6. Schmidt, W. G., Gabbard, O. G., Cacciamani, E. R., Maillet, W. G., Wu, W. W., "MAT-1: INTELSAT's Experimental 700-Channel TDMA/DA System," *INTELSAT/IEE International Conference on Digital Satellite Communications Proceedings*, London, November 25-27, 1969.
7. Abramson, Norman, "Packet Switching With Satellites," *These proceedings*.
8. Kleinrock, Leonard, Lam, Simon S., "Packet-Switching in a Slotted Satellite Channel," *These proceedings*.
9. Crowther, W., Rettberg, R., Walden, B., Ornstein, S., Heart, F., "A System for Broadcast Communication: Reservation—ALOHA," *Proceedings of the Sixth Hawaii International Conference on System Sciences*, 1973.
10. Binder, Richard, *Another ALOHA Satellite Protocol*, ARPA Satellite System Note 32, NIC 13147.

Session on views of the future—Chairman's introduction—Opposing views

by MURRAY TUROFF

Office of Emergency Preparedness—Executive Office of the President*
Washington, D.C.

INTRODUCTION

This session represents a "first of a kind" for a major computer conference. The session is devoted entirely to formal technological forecasting and assessment efforts dealing with the computer industry. Technological forecasting[1] as an autonomous discipline, with its own set of methodologies and techniques, is only about five years old. Of course, similar efforts have taken place over the years within the long range planning staffs of most technology-oriented companies and organizations. Furthermore, the intuitive judgment of recognized experts is a technological forecasting technique that has always been with us and has been well represented at these meetings by various panel presentations. What appears to be really new is a growing recognition of the need to examine potential futures systematically in order to assess a wide variety of concerns and potential consequences of technological development. The days of looking only for profit related effects seem to be passing into history. Because the scope of concern has significantly widened, with an accompanying increase in the complexity of the required analyses, new approaches to forecasting have been sought.

Several of these techniques are represented in the papers of this session (Delphi,[2] Scenario construction, Model Building, correlation analyses, and trend extrapolation). To a large extent the contributions are of interest not only for what they have to say on the future of computers and associated technology, but also for the manner in which they arrive at their observations.

The number of technological forecasting studies dealing with computers has increased considerably in recent years. Some of these are informative primarily in telling us how not to do forecasting. We have, in fact, reached the point where some of these are old enough so that a portion of their forecasts can be evaluated for their accuracy of prediction.[3] It appears from observing these efforts that the considerations which give the forecaster the most difficulty are not the projections of the technology itself, but rather its potential applications and interactions with the rest of society. There is a primary flaw underlying much of this earlier work, namely, an implicit tendency to view the computer field primarily as a driving force in society, while ignoring the impact of social, political, and economic forces on the computer field itself. Unfortunately, the rapid evolution of this technology, and the rate at which it has penetrated numerous segments of society, tends to reinforce this view subconsciously among those of us intimately involved in this area of endeavor. Therefore, as chairman of this session, I shall exercise my prerogative to offer the alternative view that the ultimate use of computer and communication technology will depend largely on factors outside the immediate scope of the field. To do this I offer two scenarios suggesting the use of computers in the 21st century.[4]

These scenarios are based upon the same projected basic information technology, but use it in opposing or contrasting manners. Each one rests on differing but plausible assumptions about resources and values of society in the 21st century. Together they provide a fundamental caveat for all the papers in this session by dramatizing the difficulties facing anyone attempting to forecast the future of information technology and its application.

These two scenarios represent what might be characterized as a plausible "open" society and "closed" society. They are not intended to portray the extremes of open society, which would be anarchy, or of closed society, which would be a slave state. Both scenarios are presented as selected day-to-day communications that an average citizen might receive via his computer terminal. Thus they do not provide an exhaustive description of the alternative societies, but rather convey to the reader a Gestalt—a feeling for what these alternatives might be like to live in.

Common assumptions for both scenarios

We assume that society in the year 2000 and thereafter can be characterized as "information rich." Essentially, all the information generated by society and needed for its operation exists in electronic form. The collection, processing, transmission, distribution, storage, and retrieval of information on a day-to-day basis takes place

* The views presented are those of the author and do not necessarily reflect official policy of the Office of Emergency Preparedness.

on a largely "self-generating and sustaining" basis. The information centers, the networks tying them together, and the procedures governing their use are sufficiently compatible that they may be viewed jointly as a single nationwide information complex largely transparent to the users and their applications—much as the telephone system is today. Terminals exist in every home, voice recognition is a common form of computer input, mass on-line storage is cheap and plentiful, and other similar marvels of technology (by today's standards) abound. Both scenarios, therefore, assume the continued advance of information technology along the directions now perceived. Barring a major holocaust, such evolution of the technology does not seem unreasonable. Of course, the implicit assumption that the industry has managed to agree by the year 2000 on standards for internal compatability of a nationwide information complex is somewhat more questionable.

The "closed" information-rich society

It is assumed that by the year 2000 some evolutionary process has resulted in a scarcity of important material and human resources—energy, mineral ores, medical talent, etc. Society is characterized and regulated by various algorithmic and procedural models, operating on a real time basis and striving to maintain the precarious balance between supplies and demands. Government consists of a dictatorship by the system over which no one individual or group has an effective control. The various components of this system are largely the product of a reductionist philosophy and represent uncorrelated short term fixes to problems as they occur. The complexity of the overall interaction of these individual fixes is not really well understood.

As resource limitations have become increasingly critical individual options of choice have been eliminated whenever there has been any suspicion that this would benefit the objective of survivability of society as a whole. The education process is correspondingly driven largely by the needs of society and provides rather narrow training. Information flow is regulated on a "need for" basis, strong pressures exist for conformance to a common "official" ideology or singular value set, and the system as a whole fosters a high degree of centralized control. However, an illusion of free choice is not only allowed to exist but even encouraged, and other types of escapism are provided via recreational pursuits.

The "open" information-rich society

It is assumed that through effective planning society has reached by the year 2000 a posture in which resources are relatively abundant with respect to societal and individual needs. Emphasis is placed on maximizing individual options of choice. Extensive understanding of adaptive and cybernetic approaches to solution of practical problems allows a high degree of decentralized controls, although centralized predictive capability is retained to detect and announce potential conflicts. As a result of this abundance, society is tolerant of a heterogeneous set of ideologies and a multiple value system flourishes. Distinctions among the functions of education, employment, and recreation have become blurred. Educational philosophy is holistic in nature, striving to prepare the citizen for a society which enjoys highly individualistic life styles but requires strongly participatory government.

EMPLOYMENT AND EDUCATION

Open

The package of information which you requested relative to career and educational choices for your son has been prepared and is now available for access at your leisure. The package includes educational requirements and potential income ranges for the careers you selected. In addition, several careers that are similar in income and educational requirements to those you selected have been included for your consideration. Also included in the package is a report showing how people with similar performance profiles to that of your son, now feel about the career they entered.

Your recent series of unsuccessful and unprofitable purchases in Phase XL of the Department Store Game indicates that you are not yet ready to commence participating in the Economic Market Policy Game, Phase A2. Upon successful implementation of the A2 buyer's strategy, you will be qualified as Assistant Buyer and eligible for employment at the Group Store, if you so choose.

I would like to exchange for one year my current job function as manager of distribution for a midwest appliance manufacturer for a job function in the sales area (at level B or above).

Closed

As a result of your son's performance this past educational cycle, he has been transferred to track three for preparation in employment class five.

Under your job classification you are entitled to attend one professional meeting this year. According to your schedule of assignments and available travel funds the MIS system has determined that for the meeting in March of the SOCIETY FOR FORTRAN 84 PROGRAMMERS is best suited to fulfill this privilege. It is further suggested that you attend the following sessions....

Your recent series of unsuccessful and unprofitable purchases in Phase XL of the Department Store Game indicates non-orientation toward the merchandise manager's field. You are to report to the Group Store, as Stock Clerk, Class C, Malthusium Kit department.

A recent computer evaluation of your job performance exhibits a discrepancy with respect to the job's requirements. In order to avoid declassification, you must report for updating on....

Open

Your request for educational opportunities has prompted us to bring to your attention the fact that Ecological Watchmen and Recycling Engineers are urgently needed. Many openings exist for this outdoor occupation, offering excellent wages and, currently, a one-to-one exchange program: for one year on the job, spend one year at any knowledge or recreation center of your choice, all expenses paid.

Openings are now available for robotic controllers in building construction. In view of your experience in this area, we would like to enter negotiations with you for a work period of six months.

Closed

As of... you have been awarded a new job with the Government Employment Corporation. This opportunity to serve your government, of course, carries with it certain sacrifices in dwelling area and salary cash flow; however, be assured that the data banks have your file in the active list and will do all that is possible to reinstate you in private enterprise at some time in the future. Your current employer has requested that you be notified that a suitable transfer bonus will be applied to your account upon proper training by you of the replacement for your current job function.

SERVICES

Open

Your proposal submitted last week to our venture bid service for a new product in industry sector 83 has received six complete bids and eighteen partial bids for financial support. None of these, however, meet all your initial constraints on ownership and profit sharing parameters—a complete analysis of differences is attached including an estimated ten year profit flow analysis to you on the three most favorable bid combinations. Please advise should you wish to restate your constraints.

Your current offer for a house painter has had no responses the past week. Based on current market conditions we estimate only a .3 probability of response this month. A raise in your offer of 15 percent would increase the response probability to .95.

The paper you submitted last month has been reviewed. It appears that your paper not only has original content but it will allow us to retire from our immediate access files ten other papers to the offline archives. We are, therefore, adding your paper to our system immediately and we are informing users with appropriate interest profiles of its availability. Thank you very much for your contribution.

In addition to the material resulting from your particular search of our literature banks, the following four individuals have indicated they are seeking contact with individuals exhibiting your search pattern. This auxiliary service of your local knowledge center is intended to promote contact among individuals with common interests. It is your option to establish contact. However, if you wish your name added to the list, please notify us.

Your request of our news file has revealed that the information you desired has not yet been released by the appropriate agency. We have therefore entered a formal request for disclosure and established a reporting team to handle the matter. You will hear further within ten days.

Closed

A recent survey by this office has resulted in your being chosen for a formal exchange of views with the "PRIME MONITOR." Any licensed barber shop may perform for you the hair shaving necessary for electrode placement.

Your recent behavior at the community meeting of June 15 seems to deviate from your filed psychological profile. Please report for a reexamination on...

As a regular service we offer at bargain rates a monthly list of "suspicious" word combinations used by the government computerized monitoring systems to select written or verbal communications for review by the Office of Internal Stability. If you wish to avoid observation for potential deviant behavior, our service is a must.

Our agency stands ready to provide you with the data requested and to which you are entitled. However, since this data is computerized, we cannot predict the actual cost to you of providing this information. It is, therefore, necessary that you post a payment bond in the amount of... before we undertake to process your request. In addition, your request is not sufficiently detailed and a new request certified by an information engineer must be submitted.

A computer analysis of your professional writings indicates that you have been writing on subjects outside your rated discipline. Your current rating within your current field will be lowered unless this situation is corrected. You are of course free to apply for change of discipline area and the appropriate forms and schedules of governing review hearings may be obtained from....

An analysis of your request of our news service shows your background profile provides no justification for supporting your need for this information. A check on this analysis by the government agency responsible for this information further confirms this result. Therefore, in order to conserve resources, your request is denied. You are, of course, free to seek modifications of your profile.

LEISURE

Open

Your requirements for a vacation house for three months have been matched against housing available location and other features of homes that appear to meet your specifications. If you wish to negotiate detailed arrangements, please let us know. If you wish to make your own home available during the time you are on vacation, please indicate this on our next communication.

We offer a complete line of "recreational," "educational," and "experience" vacations. Our information service and planning system offers comprehensive data on the environment and facilities of all vacation centers in the southwest, including the scope of available knowledge banks and communication and processing capabilities. Depending on the type of facility you choose, our knowledge banks can offer a wide variety of games, courses on many topics and dreams in many emotional variations. Our analysis routines will provide complete simulated alternatives to meet your specifications.

Mr. Norjk of Norway and his family will be in your city during the month of July. Our examination of your active hosting record in our files indicates a strong compatibility of interests for your two families (analysis enclosed). With your permission, we will pass this information on to them.

We regret to inform you that our home game service does not currently offer group simulations of primitive societies for youngsters. We will, however, poll the families using our service and establish if there is sufficient interest to modify one of the adult games in this area.

The Boston Fine Arts Museum offers the recorded experiences of creating over four hundred works of the finest art of the day. Learn the techniques of many fine artists by reliving their emotions and actions in the creative process. Our staff metric-psychologist is available for consulting in avoiding the psycho shock possible from merging disjoint personality traits of you and the artists of your choice. Be sure also not to neglect our recent acquisition in the performing arts—conductors, dancers, actors and singers.

Closed

You can rest assured that our travel service provides a full range of vacation plans matched to your travel, food, and energy allowances. Do not hesitate to call....

We are very pleased to inform you that your eighth preference choice for a vacation has been approved this year.

We are happy to announce that we were able to obtain accommodations for you at the Mountain hideaway resort. Accommodations for your wife and child were found at the Cliffside resort, a mere twenty miles away.

Our robotic sports areas offer participation in a wide variety of robotic combats. Duel to the point of robotic destruction with your own wide choice of weapons—swords, tridents, mace, clubs of all shapes and sizes just to mention a few. Duels arranged to match your skills to those of your opponent's included in the standard fee. At slightly higher rates you may challenge the current champion in various weapon classes. Also, two mass battles offered each day and special training sessions for beginners.

The apex dream parlor offers over one hundred in the latest drug-electronic stimulated dreams. Our three most popular dreams this week are:

(1) Own your own small business for a day—take full responsibility for all decisions—be your own boss.
(2) A day on the beach—enjoy unpolluted waters and clear white sand, feel the warmth of the sunlight through a crystal clear atmosphere.
(3) Have a creative idea—create and document a new idea, present it orally to a peer group and receive renown and acclaim.

GOVERNMENT

Open

To: K. Midas

The XYZ Institute for Tax Assistance is happy to inform you that a tax rebate of 313,000 credits is due to you on your 1989 income. This amount includes a .06 percent remonstrative penalty levied on the Federal Tax Bureau for its error, plus an added 7.40 percent compensating interest to compensate you for non-use of the credits in the intervening years. As your neighborhood Tax Assistance Center, we are ready to help you in any other tax matter. Please call on us.

Your application for free control of your automobile on public metroconnectors has been denied. Your tested reaction timing is not sufficient to ensure adequate safety

Closed

To: K. Midas, Sr.

The Federal Bureau of Tax Analysis hereby informs you that a tax rebate of 288,000 credits is due to you on your 1989 income. The Bureau assures you that its evaluation of your 1989 tax return is now complete and correct.

To: K. Midas, Sr.

The Federal Bureau of Census has noted that you received an added income of 288,000 credits during FY 1989 which was not reported on the census form. Accounting Department has therefore computed the required federal, region, state, county, city, block, head, automobile, and penalty taxes on this unexpected amount. A copy of your tax bill is herewith enclosed for

Open

margins for the integrity of all travelers. However, it is noted that your test results show a high probability that a standard course of training in Judo or tennis or a similar sport would probably increase your reaction rate to the point of satisfying our standards.

This is an official notice under local ordinance 817 that a set of computerized caucus conferences reflecting pro, con, neutral, and alternative positions has been established to examine a rezoning bid that is pending in your area. As a local property owner, you may join any of these conferences at no cost.

We are required by law to notify you once a month of any outstanding communications you have not accepted for delivery. These now include 118 advertisements, 16 governmental notices, and 13 personal correspondences. Of the latter, one is classified as pertaining to an in-progress contract arrangement which you must act on by July 13 or suffer unnecessary financial loss to your credit account due to contract penalties to be awarded the other party.

Following is your yearly tax analysis provided by IRS to each citizen based upon all automatically reported data pertaining to your tax account. This computerized analysis attempts to indicate your lowest possible tax liability; however, it is possible in unusual circumstances for this to not occur. You may therefore dial our local analysis service to attempt further optimization. Please notify us when filing of any apparent errors in the data pertaining to your account and supplied by other services.

Closed

the amount of 373,000 credits, which will be deducted from your purchasing account today. A copy of this message has been sent to your employer and to your regional fiscal therapy center.

Your request to move to another dwelling area must be submitted to the office of housing permits with copies for approval to the departments of energy management, transportation, tax assessments, environmental monitoring, societal impacts, and financial control. Upon reply by these federal offices you may proceed to seek concurrence from appropriate government offices.

You are hereby notified that per Public Law 813, you must vacate your dwelling unit within one month of retirement in order to maintain equitable transportation patterns. Your new classification will allow you to seek a living unit in the following areas . . .

Your violation of allowed energy consumption this month has resulted in an automatic fine of . . . due to the inability of your cash flow account to meet this deduction a proportion of your salary has been attached for . . .

It has come to the attention of the Economic Growth Office that your cash flow account is in excess of allowed positive limits. If you do not establish a higher purchase rate by June 5, we will be forced to impose a personalized tax on this account.

MEDICAL

Open

To: Dr. O. Mark Hyman

After reviewing the diagnosis of the patient you submitted to the neurological consulting network, a number of us at the Berkeley bio-engineering laboratory feel that a motor nerve hypors system we have recently developed may allow your patient normal use of his right arm and hand. Please notify us if you wish full specifications for implantation of this system.

Your requested analysis of your health records and correlation to current test data indicate a need for at least a twenty pound weight reduction under a supervised program, if you are to avoid a future heart problem. Please contact your physician at your earliest convenience. Thank you for using our automated diagnostic booth.

Thank you for calling on our community information service. In answer to your request, following is a list of medical personnel and clinics in your area with an assessment of specialties, performance histories, and fees for each.

For a slight additional fee our computerized dating service offers an auxiliary matching procedure based on a complete genetic analysis of both parties.

Closed

To: Dr. O. Mark Hyman

An examination of the neurological condition of your patient's arm and an evaluation of his job function shows too low a benefit/cost ratio to warrant further corrective treatment. You are hereby instructed to terminate further effort on this case and revise your allocation of resources accordingly.

Your computerized diagnosis does not provide you with a high enough priority to see a doctor at this time. Your appointment has been scheduled for next Thursday at 10:00 A.M. Your probability of survival to that time is estimated at 68 percent; the current immediate appointment threshold is 57 percent. Do not hesitate to dial us again should you feel a deterioration in your condition.

Due to genetic mismatching your application for breeding with Ms. . . . has been denied; however, we offer the following list of egg or sperm alternatives with which the two of you may seek to form a family unit.

Enclosed is a list of local discussion groups which our analysis shows have a high potential of aiding you in resolving your current concerns. You may joint these physically or via remote terminal hookup on either an anonymous or non-anonymous basis. Do not hesitate to call on us for any further mental health service you may wish.

This is to notify you that Mr. Jung located at unit 437 in your living complex is under treatment for MDN (Mental Deflections from the Norm). It is your civil duty to report all interactions with this individual and observations of interactions between Mr. Jung and others. Your aid in this matter will insure that the treatment will be brought to a speedy and successful conclusion.

REFERENCES

1. Mitroff, I., Turoff, M., "Technological Forecasting and Assessment," *IEEE Spectrum Magazine*, March 1973.
2. Linstone, H., Turoff, M., (eds) *Delphi and Its Application*, American Elsevier, late 1973. Turoff, M., "Delphi and Its Potential Impact on Information Systems," *AFIPS Conference Proceedings*, Vol. 39, 1971.
3. Grabble, E. M., Pyke, I. L., "An Evaluation of the Forecasting of Information Processing Technology and Applications," *Journal of Technological Forecasting and Social Change*, Vol. 4, No. 7, 1972.
4. A large part of the material in these scenarios was developed by the working group on the Assessment of Information Technology at the NATO Advanced Study Institute on Information Science in August of 1972. The members of this group were: Shuhei Aida, Issac Auerbach, Dennis Conrady, Lee Friedman, Carl Hammer, John Martell, Gil Puente, Alex Strasser, and Murray Turoff. The full report of this group will be available in Vol. 1 of *Challenges to the Development of a Science of Information* to be published in 1973 by Marcel Dekker, Inc.
5. More elaboration on the "closed" society may be found in "Meeting of the Council on Cybernetic Stability—A Scenario by M. Turoff, *Journal of Technological Forecasting and Social Change*, Vol. 4, No. 2, 1972.

The future of computer and communications services

by LAWRENCE H. DAY

Bell Canada
Montreal, Canada

INTRODUCTION

Rigorous studies of the future have become relatively commonplace in the past decade. This has been especially true in fields of rapid technological and social change. Two of these areas, the computer and communications industries, have been the subject of increasing interest and examination in the past few years.[1] The development of complex technological, regulatory, social and institutional interrelationships between these two industries and increasing recognition of their impact on the future of the economic, social, and political framework of modern nations has led to this intense interest. The future of these industries has been the subject of inquiry in both Canada and the United States by many governmental and institutional bodies as well as by groups within the affected industries. The purpose of this paper is to examine a number of these studies and outline their projections of events as they impact upon the members of the developing post-industrial societies.

The utilization of computer based systems with widespread communications links has been commonplace for some time in the military, space, corporate, R and D, and educational sectors of the economy. However, with few exceptions to date, these systems have not had any significant impact upon the everyday life of most North Americans.[2] This situation is starting to change and it is this area that will be examined in the following analysis. The impacts to be explored include the potential widespread adoption of Computer-Assisted-Instruction services in both the classroom and the home. The use of computer and communications capabilities to substitute for inter- and intra-urban business travel will also be discussed. The future of various computer based services that could be provided in homes will also be forecasted. In all cases, factors that favor or mitigate against the adoption of the various services will be examined.

FORECASTING THE FUTURE

Forecasting in general

Forecasting the future is an integral part of everyday life in both the private and institutional sense. In many cases individual decisions are made with some implicit model of the short and intermediate future in mind (e.g. plane schedules, weather conditions, stock market trends, consumer price trends, expected career paths, real estate conditions, etc., etc.). Longer term forecasting of economic indicators in the business and governmental sectors has been common for decades. In economics the tools used are mainly the various forms of trend analysis and extrapolation. Science and technology have also been forecasted by knowledgeable observers using various approaches. The emergence of Policy Research Institutes ("think tanks") and the need to forecast broader futures for many organization planning activities has led to the development of many new technological and social forecasting techniques since World War II.[3]

Delphi analysis

One of the important developments in the forecasting field has been the creation of a number of techniques that rigorously collect, analyze, and disseminate qualitative forecasts on expected developments in a particular area of interest. Generally these forecasts utilize the consensus opinion of groups of "experts" in the field under review. In relatively rare cases the output may be from a single individual who is acknowledged to have a talent and track record for developing plausible futures i.e., "genius" forecasting and scenario building—some science fiction authors have been acknowledged (usually posthumously) to have this gift of clear vision. One technique that collects and analyzes opinions from a group of experts on a particular subject under defined conditions has been called the "Delphi" technique by its original inventors at the RAND Corporation.

The Delphi technique has evolved considerably since its development at RAND in the early 1950's.[4] The basic RAND definition is described as follows:

> "The Delphi technique is a method of eliciting and refining group judgments. The rationale for the procedures is primarily the age-old adage 'two heads are better than one', when the issue is one where exact knowledge is not available. The procedures have three features: (1) *Anonymous response*—opinions of members of the group are

obtained by formal questionnaire. (2) *Iteration and controlled feedback*—interaction is effected by a systematic exercise conducted in several iterations, with carefully controlled feedback between rounds. (3) *Statistical group response*—the group opinion is defined as an appropriate aggregate of individual opinions on the final round. These features are designed to minimize the biasing effects of dominant individuals or irrelevant communications, and of group pressure toward conformity."[5]

Material from Delphi studies has been selected for this paper since this literature has the richest store of forecasts on the future applications of computer based communications systems. It should be stressed that there cannot be any one "right" forecast of the future. The material outlined below illustrates expert opinions developed at specific points in time. Each will have unique built-in biases. Proponents favoring the use of various new systems or technologies usually forecast their widespread adoption at an earlier date than other observers. Others may forecast what they would *like* to see happen rather than what may be socially, politically, or financially likely. On the other hand, many of those invited to serve on Delphi panels are in positions to help influence the future through their decisions and actions, that is, they help *invent* the future. The combined viewpoints presented here should balance out these potential biases and outline plausible directions for the future.

Computer and communications forecasts

Examination of the available forecast data base for computer and communications capabilities indicates that there are many projections available on a pure technology base. That is, many studies on hand have outlined expected developments in science and technology. However, studies that examine in any detail the adoption of technological capabilities by end-user groups in society are far rarer. The first area explored, educational technology, has been reviewed by the greatest number of user oriented research efforts. Selected examples are illustrated in this paper. The Bell Canada Delphi research findings are integrated with the results of other examinations to present a more extensive data base. The substitution of business travel through the use of new computer and communications capabilities is more virgin territory. Some broad scenarios and general research have been conducted but the data base is much thinner. The final area of interest, future home computer and communications applications, has been the focus of generalized analysis for the past few years and is currently being examined by many organizations in North America.

Organization of the forecasts

Presentation of a number of differing forecasts on the same subject in a simple manner is difficult and sometimes misleading. Each study uses different definitions, assumptions, and styles in result presentations. Delphi study results are often presented graphically but well conducted studies also outline the panelist's background assumptions, comments, reservations, etc. as a modification and/or amplification of the results summarized in the statistical charts. This background material cannot be included here although it is very valuable information for analysis.

The results of several studies on a particular subject will be summarized here graphically in the following manner (see example - Figure 1).

1. A map of the future is presented as a series of concentric rings moving out from the center which is today (1972). Each ring represents a five year time period.
2. Forecasts of an event are presented in the form of a small circle placed in a particular five year time period. This forecast only represents the *median* estimate of the Delphi panel. The inter-quartile range (middle 50 percent) of the forecasts which is normally illustrated in Delphi results has been ignored for the sake of simplicity.
3. The forecast circle is divided into two halves: e.g.,

   The top half of the circle includes the Delphi panel's estimate of the expected *percentage penetration* of the service into the *applicable* universe (e.g., 20 percent above). The universe is defined in the charts (no. of schools, homes, business, etc.). The lower half of the forecast circle illustrates the panel's estimate of the probability of the forecast actually occurring (i.e., 3 or 30 percent above). In cases where either type of the above information is not available in the original study, the half circle is solid. The number to the left of the forecast circle references the source of the material. The sources are included on the figure at the bottom.

While this process may appear somewhat complicated initially, it should enable comparisons between several forecasts on a subject on one chart rather than cross-referencing between a number of figures and charts.

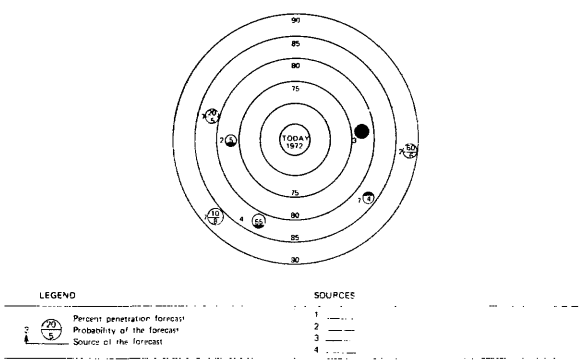


Figure 1 Forecast comparisons Example

FUTURE APPLICATIONS OF COMPUTER AND COMMUNICATIONS SERVICES

Educational technology

Introduction

The educational technology field has been the subject of a large number of studies in recent years. This is the result of several factors. Educational advancement has been a dominant national concern in North America for the past two decades. It has been assumed that a high degree of educational training in the population is a key to continued rapid economic growth and international standing. The education market has been a large one, although most of the money has gone into salaries and physical plants in the past. Many large corporations have regarded educational technology as an emerging growth market and have entered with computer and communications based instructional aids. This has resulted in the funding of pilot systems in a few centers and research into the larger potential markets for these systems. The R & D complex in government and industry has also sponsored considerable activity in this field. These and many other factors have combined to produced a large number of Delphi studies in the educational area.[6] The selective sampling below illustrates some of the findings of this research.

Computer-assisted instruction (C.A.I.)—Defintion

C.A.I. is one of the earliest off-springs of the merger of computer and communications capabilities that could have a significant impact on the everyday life of students. C.A.I. embraces the remote use of computer capabilities by students who engage in a number of instructional activities. C.A.I. systems can provide very basic as well as sophisticated capabilities. These include:

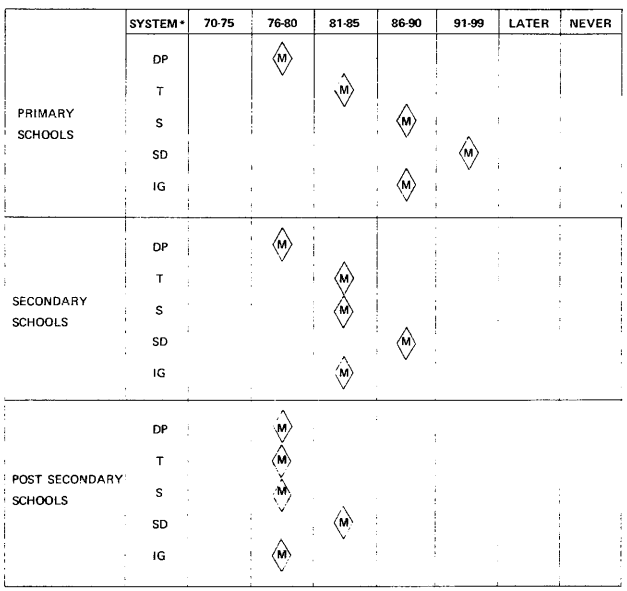
1. Drill and Practice systems (DP): a supplement to the regular curriculum taught by a teacher. The computer can relieve the teacher of the burden of routine work by reinforcing learning and at the same time provide practice work for a student at his own pace and level of complexity.
2. Tutorial systems (T): those which take over the main responsibility of developing skill in a specific area. The instructions permit freely constructed responses on the part of the student and will analyze each student's comprehension in greater depth and detail than is possible for a teacher with a classroom of twenty or thirty students.
3. Simulation systems (S): the student can change the inputs and vary the parameters.
4. Socratic Dialogue systems (SD): participative programs where the student helps develop the course. These systems would need extremely large branching facilities.
5. Instructional Games systems (IG): creative thinking games perhaps used for group as well as individual learning experiences.[7]

The Bell Canada Educational Delphi study examined the adoption of these five types of C.A.I. in three different school levels. The penetration rate examined was 20 percent of the applicable universe. The 20 percent penetration rate was considered to be well beyond the experimental stage or adoption in a few well funded but isolated centers of excellence. The 55 percent penetration rate (shown later in Figure 3) was chosen to illustrate widespread use of CAI capabilities across the educational spectrum. The Bell Panel forecasts are presented in Figure 2.

The study results suggest that Drill and Practice and Tutorial systems will be the first two types to be adopted at all levels, followed by Simulation and Instructional Games Systems. Socratic Dialogue programs will be the most complex to write and will therefore be late in gaining usage. In fact, it has been suggested that SD programs may never be adopted at the Primary level as children "have such a limited attention span and shallow interest areas that the depth knowledge that is supposed to develop from such dialogue would be useless." (panelist quote). Most panelists agree on the programming difficulties but are not willing to concede "never."

The pattern of adoption of these systems follows that of CAI systems in general, gaining initial usage at the higher level institutions and then subsequently being utilized by the lower level schools. Of course, experimentation with these systems at all levels will be an ongoing process.[8]

Examination of several forecasts of CAI usage must be presented on a more general plane. Most studies only



* Defined in text, (M) = median panel response.
Source: Frank J. Doyle and Daniel Z. Goodwill, An Exploration of the Future in Educational Technology, Business Planning, Bell Canada, Jan. 1971 (Proprietary) P. 18

Figure 2—Usage of C.A.I. systems in 20 percent of schools

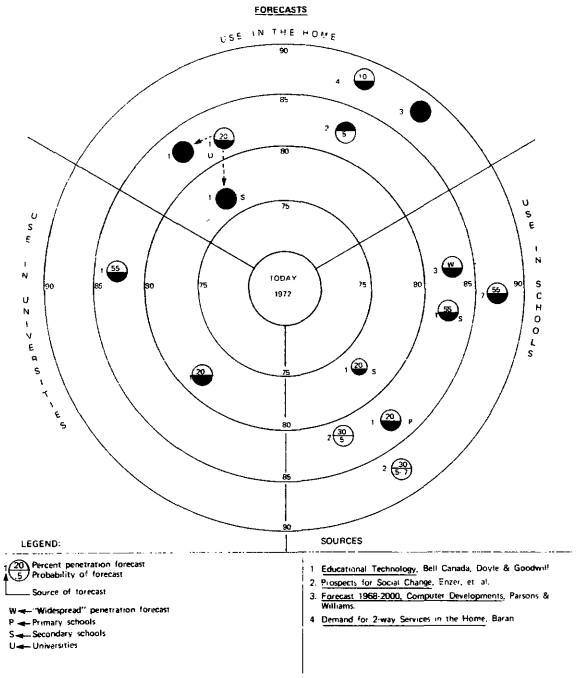


Figure 3—Computer assisted instruction—Forecasts

forecast the use of CAI as one overall service package, not the five types of services shown in Figure 2. The forecasts of adoption of general CAI services by several studies are illustrated in Figure 3. Figure 3 is subdivided into three sections which show the future use of CAI in: (a) universities (b) schools and (c) homes.

CAI use in universities

Much of the pioneering work in CAI has been undertaken at universities. However, it is interesting to note that most available forecasts of CAI adoption are in the school or home areas. The Bell Canada Delphi panelists forecast 20 percent penetration into the university environment by the end of the decade. Widespread adoption was expected in the first half of the 1970's. These forecasts implicitly assume continued funding of prototypes. Production of content material for CAI systems is another important pre-condition. An analysis of the various supporting and inhibiting factors for the adoption of C.A.I. of all types will conclude this part of the analysis.

CAI use in schools

The several sets of forecasts illustrated in Figure 3 relate to CAI adoption in schools. Initial adoption thresholds in this field are not expected by the various Delphi panels until the early to middle 1980's. An exception to this is the forecast of the Bell Canada panel which projects 20 percent penetration in secondary schools by the late part of this decade. The Institute for the Future (IFF) study: *Some Prospects for Social Change*...projects 30 percent adoption in the early eighties at a .5 probability or in the late eighties at a .5 - .7 probability.[9] The Bell study for primary schools predicts a similar time frame for addition of CAI services.

Widespread adoption (55 percent) of CAI is forecast by the Bell Canada study within approximately five years after early (20 percent) threshold adoption for the various school levels (including universities). The Bell group of panelists felt that once the threshold penetration was reached, dissemination of CAI capabilities throughout the various levels of the school system would occur rapidly. This "bandwagon" effect has been demonstrated many times before in other fields. The philosophy of "S" curve (logistic or Gompertz curve) trend extrapolation supports this contention.

The Parsons and Williams forecasts for "widespread" adoption are earlier than the other predictions.[10] A number of factors should be considered when comparing these estimates to the others.

(a) The study was undertaken in an earlier period (1968) than some of the more current research. The feelings of optimism for the future of educational technology were much more euphoric in the late sixties than they are in the early seventies.
(b) This study had many European panelists. The control of most educational funding is much more centralized for many European countries than in Canada and the U.S. Hence, widespread CAI adoption could occur faster in Europe than in North America as a result of more central decision making and funding processes.
(c) There are probably implicit differences of definition for the term "widespread" between the studies.

These factors and the interpanel differences help illustrate the "soft" nature of qualitative forecasting. Different groups invariably will have somewhat varying views of the future even if the factors noted above could be held constant when conducting and comparing studies.

In summary, most of the forecasts for the use of CAI in the educational system do not expect any significant rates of adoption until the 1980's. CAI system growth will continue to be an evolutionary process, assuming that the various roadblocks that develop can be overcome. The developers and promoters of various CAI systems often forecast widespread development and societal benefits in the nearer future. A more balanced viewpoint from several groups of knowledgeable individuals indicates a cautious optimism.

CAI use in the home

Many futurists and speculators on the prospects for what is sometimes termed the "wired city" include CAI as one of the important services to be offered in the home.

The logical provision of CAI services to the home implicitly assumes that the computer hardware, software, and content material is already available from school usage at zero cost and can be accessed cheaply from the home. Additional implicit assumptions also include low cost communications capabilities, terminals, and a consumer demand for the service (beyond that required for ill or handicapped students). Institutional roadblocks and red tape are also assumed to be overcome.

This network of implicit assumptions (often not outlined in popular scenarios) is reflected in the conservatism of the various forecasts illustrated in Figure 3. The Bell Canada panelists forecast "some use" (no percentages given) of CAI capabilities in the home by secondary students in the late 1970's and in the early 1980's for university students. The panelists also considered such factors as the availability of portable acoustic coupled terminals which could help make supplementary CAI service in the home available before permanent terminals would make it a routine part of home educational activities. The 20 percent penetration rate shown in the forecast circle with the dotted line relationship to the solid circle forecasts illustrates the panel's forecasts for 20 percent of the homes to be equipped with the audio-visual communications capabilities that would make effective CAI-type home service possible. Of course these technical capabilities could be used to provide many other services into the home as well.

All other studies referenced support the viewpoint that there will be little access to CAI in the home until the middle or late 1980's. The IFF study *Some Prospects for Social Change*... projects that there is a 50 percent chance of some use of CAI in the home by the early 1980's. The Parson's and Williams panelists feel that this will not occur until the late 1980's. This view is also held in another IFF study, *Potential Demand for 2-way Services In the Home.*[11] The findings here are that 10 percent of the homes in the U.S. might be using CAI by the late 1980's.

All of the studies noted above expect a considerable lag between adoption of CAI in the school system and use in homes. This is a logical conclusion since many more economic, social and psychological factors have to alter before adoption occurs on a widespread basis in the home. It would appear that forecasts of the future use of educational technology in the home should be examined on a broader basis than just CAI services, as sometimes occurs in the current "wired city" literature.

Factors inhibiting CAI adoption

The forecasts above were all based upon various assumptions and qualifications outlined in the original reports. Rather than repeating them here in any detail, another recent Delphi study conducted by EDUCOM for the National Science Foundation will be referenced: *Factors Inhibiting the Use of Computers in Instruction.*[12] This study is different from the ones examined earlier as it does not try to forecast dates or rates of adoption for various types of CAI in the educational system in the intermediate and long term future. This Delphi study was designed "to identify those obstacles which have hindered the development and acceptance of computer use in instruction, and to suggest means for overcoming them." The basic factors considered were in three dimensions—educational, economic and technical.

It found that the most critical dimension was the educational one. That is, most of the issues and problems relate to the availability and quality of educational content material in the systems. There is also a lack of detailed evidence to support the claims of CAI's effectiveness in the educational process. In total, 28 of the 37 factors examined and rated in importance by the panel lay in these areas. These factors are illustrated in Figure 4. The question of cost effectiveness was judged to be almost as important as the first issue by the panel. Three overall factors were examined: 1. CAI is usually an "add-on" cost in the educational process, 2. capital investment is high even where cost effectiveness can be demonstrated in the long run, and .3 existing systems have had poor cost effectiveness to date.[13] The six technical issues reviewed were only regarded as moderate or slight overall inhibiters to widespread adoption of CAI.[14]

The Educom study recommends 15 action activities that will enhance the more widespread use of CAI. These are summarized in Figure 5 which is taken from the report. The Bell Canada Delphi research had similar recommendations. The important point to emphasize is that it is the human, social and financial implications that are paramount, not the technical issues. This situation is normally overlooked by computer and communications professionals or system promoters. The end user requirements cannot be overlooked if there is to be widespread acceptance of CAI.[15]

CAI future—Conclusions

The forecasts outlined above, and the important inhibiting factors that have to be considered and overcome before CAI gets widespread adoption, leave the observer with a feeling of cautious optimism. The technical and cost factors will all be resolved with time. The social and behavioral issues will require patient planning and experimentation before they are resolved. This realistic viewpoint is not designed to downplay the *importance* of widespread CAI usage. Routine interaction with computers during the formative education years will create a fertile ground for the widespread acceptance of the use of computer and information processing power in all sectors of society. The students who grow up with these capabilities close at hand will expect their common availability at work, home, and even leisure activities. This will lead to a true information revolution. Until that time we can only expect the use of computer power in everyday life in cer-

tain isolated and special situations. The rest of this paper will examine two of these situations:

1. The use of computer communications capabilities to help substitute for some forms of business travel and
2. The adoption of certain types of computer based services in the home.

Travel—Communications substitution in business

Introduction

The use of communications and computer systems in the business environment has been extensive for some time. Most business computer applications involve large volume "number crunching" activities or mechanization of existing manual procedures. The development of time-sharing led to a second generation of applications and the

	unimportant 1	slightly 2	moderately 3	very 4	extremely 5
A. PRODUCTION DISTRIBUTION OF INSTRUCTIONAL MATERIALS.					
1. Lack of readily available and good computer-based educational materials.					M
2. Lack of professional and economic incentives for development of computer-based materials.				M	
3. Lack of incentive for faculty members to expend any considerable time and effort in modifying or creating alternative, instructional methods.				M	
4. Lack of incentives for dissemination of software.				M	
5. The lack of personnel with appropriate training and talent in the diverse disciplines required; i.e., instructional psychology, computer science, engineering, educational administration, radio-TV-film.			M		
6. Application of the "textbook" or single-author model to curriculum production instead of the "movie production" model involving a highly skilled differentiated team.			M		
7. Lack of initiative with regard to distributing software and providing training and services for its users.			M		
8. Lack of appropriate mechanisms for protecting patents, copyrights, etc., for CAI materials.			M		
9. Lack of standardization of computer systems, limiting free exchange of software.		M			
10. Lack of an organization to facilitate interchange of CAI program materials.		M			
B. DEMONSTRATION					
11. Too few examples of high quality use.				M	
12. Lack of compelling evidence that CAI is more effective than other methods of comparable cost.				M	
13. Lack of carefully planned broad programs of CAI experimentation in actual school settings.				M	
14. Failure to design curricula and systems for high-impact, low-resistance "markets" where real institutional problems can be solved.				M	
15. Lack of "critical mass" in setting up programs.			M		
C. THEORY OF INSTRUCTION					
16. Failure to recognize that material must be completely recognized and restructured if it is to be taught effectively with computer systems.				M	
17. Inadequate development of a range of computer-based pedagogical techniques. The range might include question-answers, tutorial, drill and practice, simulations, games, problem solving modes, etc.				M	
18. Tendency to put too much "on the computer" rather than share the presentation and testing of curriculum objectives with other instructional media.				M	
19. Lack of experimental data and theories in learning psychology which would facilitate the design of effective CAI programs appropriate to each age level.			M		

Figure 4—Educational problems related to widespread use of C.A.I.

D. EDUCATIONAL SYSTEM AND THE TEACHER

20. Reluctance of school personnel to go through reorganization and training that a broad use of CAI would entail.				M	
21. Cautiousness and uncertainty on part of educators as to effectiveness of CAI in comparison with traditional teaching methods.				M	
22. Scarcity of resources available to train teachers and others in the skills required to use CAI successfully.			M		
23. Fear of educators of being reduced to a "button-pushing" or clerical role by computer.			M		
24. Reservations as to possible negative effects of removing instructional process from social situation and replacing interpersonal feedback with mechanical.			M		
25. Extreme diversity of, and lack of coordination among, school systems throughout the county.			M		
26. A prevailing attitude that the computer will be used to replace poor teachers instead of to make good teachers more effective.		M			
27. Insufficiency of evaluative techniques, criteria, and agencies with which to satisfy educational standards.		M			
28. Not enough opportunity for local school people to participate in development of CAI programs.		M			

Source: E.J. Anastasio and J.S. Morgan, Factors Inhibiting the Use of Computers in Instruction, Educom, 1972 PP. 19, 27, 31, 3

Figure 4—(Continued)

spread of computer usage into more routine business activities. Generally these activities were in the technical research, scientific and statistical analyses areas. The impact of widespread computer system usage has not had a significant impact on regular middle management and executive activities to date (except perhaps to inundate these people with many computer generated reports that they don't have time to read).

Intra-urban substitution

One area where information management and distribution activities may have a significant impact in the future is in the area of business travel. This can occur in two areas: Intra-urban and inter-urban travel substitutes. Some intra-urban commuting may become subject to substitution. Continuing problems with urban concentration (pollution, poor public transportation, high cost office space, employee dissatisfaction, etc., etc.,) have led some to forecast the shifting of many types of occupational activities to the suburbs or even the home itself. Travel to the downtown office would only occur for certain types of meetings and activities while the more routine activities would be handled by various types of communication systems and remote access to computer stored files.

One of the Bell Canada Delphi studies examined some potential tradeoffs between intra-urban travel and computer communications services. The various forms of possible developments considered were:

(1) *Office Center*—a central location is maintained but there is increased reliance on audio-visual capabilities to supplement travel.

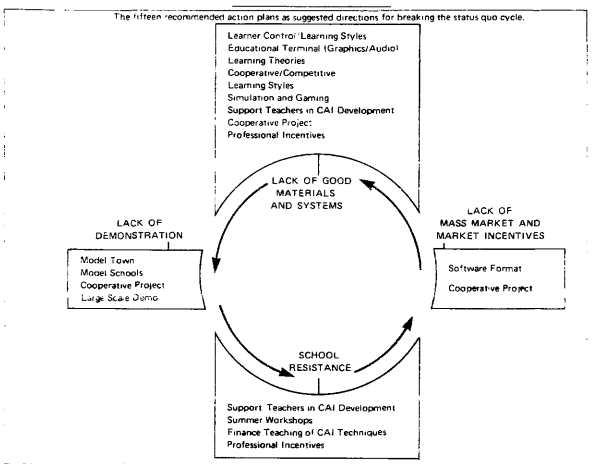
(2) *Home Remote Work Center*—located in an employee's home with access to computer and communications capabilities as required.

(3) *Neighbourhood Remote Work Center*—within walking distance of employee's home. Has work spaces equipped with required computer—communications capabilities.

(4) *Mobile Worker*—employee uses portable terminals with required input/output capabilities.[16]

The panel forecasts for potential adoption of these types of activities are shown in Figure 6.

The Bell panelists commented that the home remote work center could create many social problems, even



Source: E.J. Anastasio and J.S. Morgan, Factors Inhibiting the Use of Computers in Instruction, Educom, 1972, P. 49

Figure 5—Educom C.A.I. action plans

	70-75	76-80	81-85	86-90	91-99	LATER	NEVER
1. OFFICE CENTER		(M)					
2. HOME REMOTE WORK CENTER					(M)		
3. NEIGHBOURHOOD REMOTE WORK CENTER				(M)			
4. WORK LOCATION(S) APPROPRIATE TO THE JOB IN ORDER TO MAKE BEST USE OF TIME AND SPACE				(M)			

(M) = median panel response
Source: Daniel Z. Goodwill, An Exploration of the Future in Business Information Processing Technology, Business Planning Bell Canada, Oct. 1971, (Proprietary) P. 33

Figure 6—Intra-urban travel substitution

though the system may become technically and economically feasible. The problem of isolation from the intellectual stimulation that can occur in a professional work environment was mentioned. The problems of working effectively in the home environment surrounded by family and other distractions were also stressed. On the other hand, these and other panelists recognized the benefits that could accrue from reduced commuting time, urban congestion, etc. The ability to pick the work period most suitable to individual life styles was pointed out. Several panels suggested a compromise view whereby white collar workers in the future might split their work periods between working at home and in the office, depending upon the task at hand.

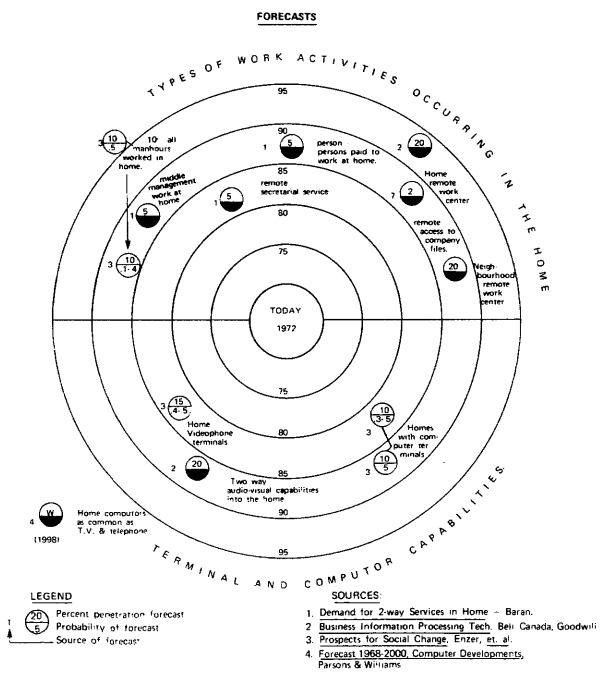


Figure 7—Computer/communications services substituting for intra-urban business travel—Forecasts

The neighbourhood remote work center was regarded by the panelists as another form of compromise between the extremes of continuing today's work patterns versus shifting work to the homes. This development and that of the mobile worker were forecasted to occur in the mid 1980's versus the mid 1990's for 20 percent penetration into the home work center concept.

The Bell Canada panel findings have been compared to those of several other studies to determine whether or not these forecasts were unduly pessimistic. Figure 7 follows the basic format of the earlier comparison chart. However, in this case, each of the forecast circles has a brief description explaining the forecasted item in more detail. These views are also conservative when compared to those of some of our more utopian social thinkers and planners. This reflects the broader variety of material available when compared to the CAI studies. The chart is divided into halves, the upper half indicating the forecast of activities that might occur in the home and the lower half indicating the availability of supporting technology in the home. An overall evaluation of these forecasts indicates that the various groups who have examined the likelihood of these work functions occurring in the home share a similar conservative viewpoint to the respondents of the Bell Canada study (top half of Figure 7). Low threshold market penetration (2 percent-5 percent) for services such as remote secretarial service, remote access to company files and person-to-person (clerical etc.) services provided electronically in the home is not expected until the middle or late 1980's. Significant acceptance of middle management activities at home (5 percent) or of total work hours at home (10 percent) is forecast for the 1985 to 2000 time frame. In summary, the Bell Canada panelists' forecasts actually look optimistic when compared to the views of other study findings.

The second half of Figure 7 displays some of the terminal and computer capabilities that may be available in the homes of the future. These types of capabilities would be necessary before many of the work activities forecast above could take place effectively. It is interesting to note here that most studies forecast 10 percent-20 percent penetration of the technology in the homes before the work function shifts significantly to the home. This lag may be as much as 10 years or more.

The overall conclusion to be drawn from the available forecasts for intra-urban substitution of travel by computer/communications capabilities is that this will not occur in a widespread fashion until very late in this century. The build-up period prior to the work location shift will see an increasing proliferation of remote technological capabilities in the home. Once again the main deterrents to change are expected to be social, not technological in nature. Individuals will need to develop new attitudes toward working in physical isolation, and in electronic partnership with their fellow employees or professionals. Home life likely will be restructured. The secondary benefits and changes involved with this form of social engineering must be examined as well. Employers will have to shift their attitudes toward "managerial spans of

control" when the "knowledge" workers are no longer physically present. Literally hundreds of questions of this nature will have to be examined and answered before computer communications and work systems are restructured to favor the widespread substitution of intra-urban travel by computer based communications systems.

Inter-urban substitution

The substitution of inter-urban business travel by various types of communications services is quite different from that of intra-urban substitution. In this case, travel tends to be less frequent but more expensive in terms of time and money. The purposes for business travel can be slotted into a number of different categories. Certain forms of travel are quite repetitive and might be substituted for by teleconferencing systems using various combinations of audio/visual/computer capabilities. This area has begun to receive some attention from various researchers in the social field as well as those in the communications field. Figure 8 illustrates various forms of transportation of men, material and information, and speculates on their substitution possibilities. Figure 9 indicates some of the results of a survey of the opinions of business executives toward the use of various communications media in specific situations. The overall conclusion is that as the complexity of the communications task increases the perceived felt need for face-to-face contact is greater.

The behavioral research cited above illustrates the types of thinking being directed toward the question of inter-urban travel substitution. Bell Canada has several proprietary, current research activities in this area. The key point to consider when forecasting the future in this area is that the basic technology is *available now*. Prototype audio, 2 way television, and computer controlled teleconferencing systems exist today and are being used. Future directions in technology will only provide further subsidiary improvements (e.g., wall size flat color T.V. screen for visual teleconferencing) or cost savings. The main thrust in the current research efforts is to identify the types of business travel interactions that can be substituted and the best combinations of teleconferencing

MOTIVATION FOR TRANSPORTATION	SUBSTITUTION POSSIBILITIES
I. Transportation of Men In Order To:	
1 Observe an activity	Visual substitution
2. Participate in an activity	Non-substitutable, if physical, otherwise substitutable
3 Hear	Audio substitution
4. Talk, discuss	Audio substitution
5. Spend leisure time, change scenery	Non-substitutable, if attraction is in physical travel or activity
6. Have services performed	Non-substitutable
7. Perform services	Non-substitutable
8. Meet business associates	Visual and audio substitution
9 Because some characteristics of personality or intelligence makes face-to-face contact desirable	Non-substitutable
II. Transportation of Materials and Energy in Order To.	
1. Increase their value	Non-substitutable
2. Stimulate trade, commerce, industry	Non-substitutable
3. Provide the essential constituents of various processes (body, industry, or commercial)	Non-substitutable
III. Transportation of Information In Order To	
1 Facilitate the process of decision-making	Substitutable

Source: Frederick W. Mennot III, "Substitutability of Communications for Transportation," Traffic Engineering, Feb. 1963, p. 22

Figure 8—Travel/communications substitution possibilities

SITUATION	LETTER OR MEMO	TELEPHONE CALL	FACE-TO-FACE CONTACT	COMBINATION	NA
Resolving a disagreement	5	5	85	4	1
Praising a subordinate	8	1	82	6	4
Conferring with peers about future plans	4	13	76	4	3
Making a concession to someone else	12	19	60	5	5
Checking on whether a subordinate has accomplished a task	7	26	59	6	3
Dealing with customers you know	6	39	28	7	19
Dealing with vendors	17	38	23	6	16
Requesting information	35	41	12	9	3

Source: Ronald Westrum, Unpublished Ph.D. dissertation, Purdue University, 1972.

Figure 9—Executive attitudes towards media used in work situations

systems to meet the user needs. The penetration rates for business travel substitution will be almost totally dependent on behavioral factors rather than technological ones.

Computer and communications services into the home

Introduction—Bell Canada study design

The discussions of CAI and travel/communications substitution each reviewed the forecasts in those areas that impacted on the home. The general conclusions of the referenced material were quite conservative, compared to previously published forecasts. This prompted the Business Planning Group to initiate a study to examine consumer acceptance of a broad range of potential new computer based communications services in the home.

The choice of forecasting techniques was especially difficult here. The use of the traditional Delphi methods to elicit the opinions of recognized "experts" in relevant fields left a considerable gap in the information base. This gap was knowledge of the attitudes and feelings of the potential consumers, that is, the housewives, toward "wired city" services. This school of thought stated that when it came to rating potential acceptance of these potential services, the housewives were in fact the only group of "experts" on the subject. The conclusion of this debate was to develop a new modified Delphi technique that utilized both "Experts through research" and "Experts through experience" (housewives).[17] These two homogeneous Delphi panels examined and debated the market potential of various wired city services into the home. The emphasis was on services, not technology, and on analyzing the "comments" feedback from the panelists as well as their statistical estimates. This emphasis on comments helped get at the underlying reasons for the various conclusions reached by the two panels. The developer of this technique has called it SPRITE (**S**equential **P**olling and **R**eviewing of **I**nteracting **T**eams of **E**xperts).[18]

This study examined four main types of home services as well as ten types of information retrieval from the home. These were grouped by considering whether they were either a) fully interactive services, or b) limited interaction services. The services studied are shown in Figure 10. Detailed service definitions given to the panelists to facilitate their analysis will not be repeated here.

FULLY INTERACTIVE	LIMITED INTERACTION
Remote Shopping Services	Consumer Shopping Guide Consumer Service Guide Consumer Rating Service
Remote Banking Services	Demand News Service Demand Entertainment Service Demand Education Service.
Electronic Home Security Services	Household Guide Electronic Bulletin Board Personal Filing Service.
Programmed Education Services.	Home Calculator Service.

Source: Michael T. Bedford, "The Future of Communications Services in the Home," Business Planning, Bell Canada, Montreal, Nov. 1972. (Proprietary) pp 19-23

Figure 10—Future of communications services in the home—Services reviewed

Study findings[19]

An overall review of the findings of this research indicates that there were not very many significant differences between the experts and the housewives. The main differences that emerged were usually based upon fundamental attitudes in one group or another toward the acceptability of particular service features. The groups maintained their independent viewpoints although in less crucial cases, the experts sometimes agreed to shift their viewpoints toward those of the housewives. The study results are presented as overall evaluations of various service options rather than median estimates of the possible future dates of service penetration.

(a) *Remote Shopping Service*

The panelists felt that the facilities of a shop-from-home service would be used mostly for the purchase of articles such as grocery dry goods, perishable goods (other than meat), small appliances, and drugs and cosmetics. Meat, clothing, and large appliances were rejected by about half the panelists. They did not expect users of the service to pay much of a price premium (over and above the current cost of shopping) for the service. The main benefit of the service seemed to be the convenience aspect rather than any perceived cost reduction. Preferred methods of payment for this prospective service appear to be through charge accounts (favored by the experts) and the remote banking option (favored by the housewives).

Several panelists, both experts and housewives, made the comment that shopping trips fulfilled many more needs than the traditional purchasing of goods. It seems that if a remote shopping service is to become widely accepted, alternative outlets for housewives' social drives will have to become available.

(b) *Remote Banking Service*

Over eighty percent of the panelists expected that a checkless banking service (once developed) would be used for transactions involving retail stores, transportation tickets, contractual payments, and utility payments. Use of the service in restaurants was rated only slightly less likely (about sixty percent indicating it would be used). The experts on the panel felt that all of the possible optional features of the service were "very desirable" when ranked on a five point scale between "essential" and "definitely not desirable"; the housewives differed insofar as they felt that an overdraft privilege was less desirable and that an automatic payroll deposit feature was essential in any such feature. On the matter of costs for this service, both groups of panelists expected them to remain about the same as that for conventional banking transactions, regardless of what type of transaction was being considered. Hard copy of all transactions received at regular intervals was rated the most useful form of documentation for this service. In regard to soft-copy, or non-filable copy, the experts were generally more receptive to this form than were the housewives.

(c) *Electronic Security Service*

The panelists felt that the automatic detector/alarm would be most desirable for the detection of smoke and/or fire, natural gas fumes, and intruders in the home. The threats receiving lesser ratings were carbon monoxide, high levels of radiation, and dangers such as flooded basements and frozen pipes. In the event any of these threats were detected, the panelists thought that an alarm to the proper emergency group would be more desirable than an alarm to a neighbor's home or an alarm ringing outside the home to be heard (hopefully) by anyone in the vicinity. An alternative type of alarm system provided a set of push-buttons in the home which would act as normally operated fire alarms or burglar alarms, but would also be equipped to summon help for accidents in the home, poisonings, et cetera. Panelists indicated that this type of alarm service would be most desirable for signaling the presence of fires or intruders to the proper authorities. Householders would expect to pay more than their present insurance costs if they could obtain this service, according to the panelists. This reflects the idea that the cost of the service to the household would be slightly greater than the expected reduction in insurance premiums for a home with the service. One of the expert's comments covers this point well: "The differential insurance losses would probably be nil. The real pay-off is one's increased perceptions of safety."

An interesting observation throughout this part of the study was the housewives' greater acceptance of these security oriented services. They found all the services more desirable and felt householders would be willing to pay more for them than did the experts.

(d) *Programmed Education Service*

The results of the study regarding programmed education in the home were interesting because of the differences between panels. The housewives felt the service would be most useful for older (over 18 years) age groups and stressed the impact of the service on continuing adult education. The experts thought it would be most useful for school age (5 - 18 years) children. Both groups felt the service should offer a broad range of subjects in order to maximize the use of the service. In the words of one of the experts, "The consensus reflects the point that as much as possible should be available on the system so that it can be useful to the widest possible audience. On a per

sonal basis, only a few subjects would be of interest." The experts and housewives both felt that courses provided through the service would not be free, but would cost less than comparable institutional courses.

(e) *Limited Interaction Services*

The Limited Interaction Services were grouped into several categories. The first grouping was Consumer-Oriented Services (Consumer Shopping and Service Guides and the Consumer Rating Service). The statistical response of the panelists indicated that these services were fairly important improvements over the current non-electronic means of providing them. However, this estimate was qualified with a very low estimate of the monetary value of these services. The median payment estimates lay between "free" and "less than $1 per month." A review of the panelists' comments in this area revealed strong negative feelings toward these electronic services in the minds of many respondents who were quite satisfied with the current means of obtaining consumer services.

The second category was termed Information-Oriented Services (Demand News, Demand Entertainment, Demand Education). Both groups of experts regarded Demand Entertainment and Education Services as ones that would be used widely if available at a reasonable cost. These two services were regarded as significant improvements over the current means of providing the services. However, once again there were many negative comments directed toward the services, especially from the housewives. These centered around their low opinions of the potential quality of Demand Entertainment Services and the sterile and impersonal nature of Demand Education Services. Both groups also had many negative opinions toward the value of Demand News Service. The common point stressed here was that there are many alternative media sources for news information.

The third grouping of services was called Home Management-Oriented Services (Household Guide and Electronic Bulletin Board Services). These services were also regarded as somewhat questionable electronic duplications of current activities being handled quite well by the present media. The personal touch of the supermarket bulletin board was stressed by several housewives. The experts were more optimistic toward the services' acceptability since they were more efficient than current means. Both groups saw very little economic value in the services.

The final classification was Data-Oriented Services (Personal Filing Services and the Home Calculator). These services were the least attractive to the housewives. The experts also felt that these services were ahead of their time for normal homemakers and would only appeal to professionals working at home.

Study findings—Conclusions

Several points seemed to emerge clearly from this research effort. Planners, marketers, researchers, etc. in the computer and communications fields often appear to feel that the new capabilities they are creating technologically will be of great benefit in the home. However, most of the services that these "experts" are designing for the "wired city" are mere electronic substitutions for many low cost or free methods of obtaining the services today (note: the consumer considers advertiser supported or subsidized services as effectively "free"). The experts are planning for the use of these services by *other* people, that is, the impersonal thing called "the market."

On the other hand, we have the housewives, the potential consumers of these proposed services. The housewives represented on the panel were from an innovative modern community of upper middle class citizens, near Montreal, Quebec, (e.g., the most likely type of consumer to be offered these services in the near future). This panel reacted to the proposed services in two ways. Their statistical responses often indicated that they felt the various services would be widely accepted by "people in general." However, their comments which were carefully encouraged and analyzed, clearly indicated that these women felt that the services might be acceptable to "people" or their neighbours, *but not themselves*. The housewives appeared to reflect the conventional wisdom of many readily available futuristic visions of the computer/communications fields in their statistical responses. When their own feelings were probed, they were often less than enthusiastic toward the services.

The housewives' conservatism seemed to be based on a desire to maintain a personal relationship in many of their day-to-day activities. The new services were often regarded as efficient but impersonal. Many of the respondents volunteered fears of being shut off from the world in an electronic prison. The recreational nature of many daily events such as shopping was often overlooked by experts. Housewives also regarded the home as a place that did not have to be operated on a totally efficient manner. As one respondent commented: "Sometimes it is less convenient to be so well organized."

These findings do not mean that the various services outlined above will not be offerred to the public or eventually accepted by them. However, most services will start with professionals working in the home or with housewives who have many other activities beyond normal home duties to help maintain their contact with people outside of the home. As new forms of outside activities develop for housewives over time, they will come to depend upon electronic substitution for many of the old ones that seem so important today. Once again the trend is evolution not electronic revolution.

CONCLUSIONS

The theme of this paper is probably clear by this point. Development of computer based communications systems and their attendant technologies is going to continue at a rapid pace. Many of these advanced systems will be utilized in business, government and institutions. The devel-

opment and acceptance of systems that interface directly with the ultimate users is going to continue to be a slow and painful process. These systems will have to be designed and redesigned with the human element (and frailties!!) at the center. Emotional and "irrational" human considerations will determine the extent and timing of the use of CAI, travel/communications substitutions, and wired city services. The development of the more sophisticated systems will be based upon the structures financed by mass and demand entertainment services as well as similar educational services. There will continue to be centers of excellence in specific North American locations where pilot systems are trialed and refined. However, these systems can only be regarded as crude models of widespread systems to be in use toward the end of the century, not as mirrors of a future that will be upon us in a few years.

REFERENCES

1. Examples here include:
   Forecast 1968-2000 of Computer Developments and Applications, Parsons and Williams, Copenhagen, 1968.
   Baran, Paul, Lipinski, A. J., *The Future of the Telephone Industry*, R-20, Institute for the Future, Menlo Park, Sept. 1971.
   Salancik, J.R.,Gordon, Theodore J., Adams, Neale, *On the Nature of Economic Losses Arising from Computer Based Services in the Next Fifteen Years*, R-23, Institute for the Future, Menlo Park, March 1972.
   Communications Needs of the Seventies and Eighties, Bell System, April 1972 (Internal Document).
   Doyle, Frank J., Goodwill, Daniel Z., "An Exploration of the Future in Educational Technology," *Business Planning*, Bell Canada, Montreal, Jan. 1971 (Proprietary).
   Goodwill, Daniel Z., "An Exploration of the Future in Business Information Processing Technology," *Business Planning*, Bell Canada, Montreal, Oct. 1971 (Proprietary).
   Bedford, Michael T., "The Future of Communications Services in the Home," *Business Planning*, Bell Canada, Montreal, Nov. 1972 (Proprietary).
   Canadian Computer/Communications Task Force, *Branching Out*, Vols. 1 and 2, Information Canada, Ottawa, May 1972.
   Dunn, D. A., et al, *Policy Issues Presented By the Interdependence of Computer and Communications Services*, Stanford Research Institute, Feb. 1969 (for FCC).
   Berstein, George B., *A Fifteen-Year Forecast of Information Processing Technology*, Research and Development Division, Naval Supply Systems Command, Jan. 1969.
2. Airline reservation, ticket and hotel reservation, on-line banking and credit checking systems are examples where the individual has second hand personal contact with computer and communications systems.
3. Readers interested in technological forecasting are directed toward a number of excellent books on the subject which include:
   Ayres, R. V., *Technological Forecasting and Long Range Planning*, McGraw, Hill, New York, 1968.
   Bright, J. R., (ed.), *Technological Forecasting for Industry and Government*, (New York), Prentice Hall, 1968.
   Jantek, E., *Technological Forecasting in Perspective*, O.E.C.D., Paris, 1967.
   Martino, Joseph P., *Technological Forecasting for Decision-making*, American Elsevier, New York, 1972.
   Wills, Gordon, et al., *Technological Forecasting*, Pelican Books, London, England, 1972.
4. The most extensive review of Delphi to date may be found in:
   Linstone, H. A., Turoff, Murray, *Delphi and Its Applications*, American Elsevier, New York, 1973.
5. Dalkey, Norman C., *The Delphi Method: An Experimental Study of Group Opinion*, RM 5888-PR, RAND Corporation, Santa Monica, Calif., June 1969, page v.
6. A review of several of these studies may be found in:
   Judd, Robert C., "Use of Delphi Methods in Higher Education," *Technological Forecasting and Social Change*, Vol. 4, No. 2, 1972, p. 173.
7. Doyle and Goodwill, *Future of Educational Technology*, p.p. 18-9.
8. *Ibid* (foregoing two paragraphs).
9. Enzer, Selwyn, Little, Dennis L., Lazer, Frederick D., *Some Prospects for Social Change by 1985 and their Impact on Time/Money Budgets*, R-25, Institute for the Future, Menlo Park, March 1972.
10. *Forecast 1968-2000 of Computer Developments and Applications*, Parsons and Williams, Copenhagen, 1968.
11. Baran, Paul, Lipinski, A. J., *The Future of the Telephone Industry*.
12. Anastasio, E. J., Morgan, J. S., *Factors Inhibiting the Use of Computers in Instruction: Final Report to the National Science Foundation*, EDUCOM, 1972.
13. *Ibid*, p. 41.
14. *Ibid*, p. 44.
15. Readers interested in another hard look at CAI and educational technology are referred to Oettinger, Anthony G., *Run, Computer, Run: The Mythology of Educational Innovation*, Collier-Mac-Millan Books, 1971.
16. Goodwill, *Future Business Information Processing Technology*, p. 32.
17. Bedford, *Communications Services in the Home*, p.p. 7-11.
18. Unpublished memorandum, Bedford, Michael T., *The Value of Comments Analysis and an Analysis of SPRITE as a Planning Tool*. (Proprietary).
19. Selected material from: Bedford, *Communications Services in the Home*, p.p. 27, 35, 45, 55, and p.p. 63-71.

The social implications of the use of computers across national boundaries

by BURT NANUS

University of Southern California
Los Angeles, California

MICHAEL WOOTON

Southern Methodist University
Dallas, Texas

and

HAROLD BORKO

University of California
Los Angeles, California

Large time sharing systems and distributed networks of computers are already major factors in tying together decentralized national operations in both the public and the private sectors. In the public sector, the marriage of computers and communications is apparent in such systems as the ARPA Network, the Air Defense System, law enforcement systems, weather forecasting and the like. In the private sector, there are many such systems used for tying together sales offices and warehouses or ticket offices and data banks of reservations systems, as well as serving various other scheduling, financial control or logistics operations in large corporations. In fact, discussions on the design and development of massive national or regional information utilities have been appearing with increasing frequency of late.[1]

In light of these developments it appears to be only a matter of time before these kinds of computers and communication networks are in widespread use *across* national boundaries. In fact, as will be briefly discussed in this paper, a small number of such systems are already in use in multinational organizations and many more are being contemplated. But while the national use of computer networks is just a logical extension of current trends and capabilities, the multinational use opens a whole new realm of considerations in the technical, as well as the social, political and economic spheres. These issues have not yet even been identified, let alone explored or addressed in any meaningful way.

In order to begin an inquiry in this area, the Social Implications Committee of AFIPS sponsored a year-long Delphi study at the Center for Futures Research of the University of Southern California. Some of the major findings of the study, which was completed in February, 1973 are summarized in this paper.

PROBLEM AND METHODOLOGY

The use of computers across national boundaries can take many forms. We will use the designation "multinational computer system" (hereinafter abbreviated as MNCS) to mean any arrangement whereby computers in one country are directly linked to other computers, data bases, or computer users in one or more other countries. The use of computers in this manner at the present time is certainly not widespread. However, as one projects ahead ten to twenty years and contemplates on the one hand, the rising tide of multinationalism in both corporations and governmental organizations, and on the other hand, the rapid increase in capabilities and decrease in cost of computers/communications networks, one can conjecture that it is only a matter of time (and probably not very much time) when these kinds of applications will proliferate. To help understand how such systems can be employed effectively, a brief look at the literature describing current applications might be instructive.

In the public sector, one of the most dramatic uses of computers across national boundaries is in connection with the United States space program. NASA's Apollo Program, for example, employs a real time international (indeed, interterrestrial) network composed of computers linked to monitoring stations to provide instantaneous data for real time decision-making with regard to the landing of astronauts on the moon. A space ship on the moon may be the world's most remote terminal allowing man to interact with a central computer in a real time mode. NASA also uses a system called the Computer Assisted Network Scheduling System (CANS) to produce, modify and observe actual and simulated schedules for space flights tying together stations and equipment in a

worldwide network.[2] In addition, NASA's RECON System is an excellent example of a multinational on-line information utility. The network consists of a central computer facility in Germany linked to terminals in the Netherlands, Sweden and France for the purpose of permitting research users to operate in an on-line mode through access to a large collection of scientific and technical literature.[3]

The RECON System has served as a model to stimulate researchers to suggest that the United Nations sponsor a world science information system as a mechanism for permitting nations, universities and professional associations to share scientific information more effectively and avoid needless duplication of efforts. Most nations still have a proprietary policy regarding the flow of scientific information across national boundaries. For this reason, UNISIST (the U.N. acronym for the World Science Information System) does not envision a world system in the sense of a preplanned, integrated organization under a single manager; instead, UNISIST hopes to operationalize a flexible network of cooperating services among quasi-independent systems. The end goal technically of UNISIST is to have centralized processing of information, probably on a regional basis, so that users will have information available to them across national boundaries in an on-line time-sharing basis with display of information virtually anywhere via remote terminals. Part of the system would be a world register of scientific periodicals. At the Intergovernmental Conference for the Establishment of a World Science Information System, held at UNESCO House in Paris in October, 1971, the Conference recommended that the Director General of UNESCO take steps to make adequate budgetary provisions available in order to implement the first stages of UNISIST during the fiscal period 1973-74.[4]

In addition to UNISIST, the United Nations is currently in the process of establishing its international computer center in Geneva, Switzerland to serve as a centralized computer facility for all U.N. organizations, wherever they are located. The center contemplates being on-line to other U.N. centers around the world, serving such users as the U.N. Development Program and the World Health Organization, with an ability to develop and maintain international data banks containing inventories of economic and social statistics, and to develop multinational management information systems for use within the United Nations.

There are other public applications of MNCS as well. The World Meteorological Organization employs a worldwide computer network in its World Weather Watch program for monitoring and forecasting weather conditions at the global level.[5] The banking community represents another example. Most central banks are government run and are rapidly increasing their use of on-line computer services tying together remote branches. It is expected that by the late '70s, most central banks will be transferring funds and carrying out the bulk of foreign exchange operations via international computer networks linking chains of large multinational banks to create a global service industry in banking. Finally, in passing, we must at least mention the many multinational military uses of computers which have provided much of the technology that will be applied in non-military applications.

In the private sector, there are also numerous existing applications of the use of computers across national boundaries. The major international airlines and some travel agencies already have multinational reservations systems covering many activities. For example, International Reservations, Limited has a multinational real time booking service to confirm reservations for hotels, motels and car rentals with a central computer center in Virginia linked to remote terminals in the United States, Great Britain, Switzerland and Ireland.[6] The American Express computer network also is a worldwide system with links to remote terminals for reservations purposes. Similarly, such airlines as the Scandinavian Airlines System, Air Canada, Yugoslav Airlines and others have multinational networks serving reservations functions as well as management reporting, statistical analysis, passenger records and other accounting functions. In fact, there is already an example of an international cooperative effort to own and operate a multinational computer network performing similar functions for a large number of users. The Societe Internationale de Telecommunications Aeronautiques (SITA) is a multinational computerized communications network owned by the airlines and serving about 130 companies.[7]

Computer services networks for commercial purposes are relatively common within certain countries such as the United States and Great Britain but are now beginning to reach across national boundaries, particularly in Europe. For example, Honeywell, since its acquisition of General Electric's computer hardware functions, now has a vast network of on-line time sharing services. At present, Honeywell operates a European time sharing service consisting of seventeen time sharing systems covering fourteen countries from Denmark to Italy and serving more than 8,000 businessmen, engineers, scientists and students.[8] Similarly, University Computing Company, Limited has a multinational computer utility consisting of two Univac 1108 computers in London linked to regional centers in Paris, Dusseldorf, Frankfurt, Brussels, and The Hague.[9]

The publications and broadcasting industries have expressed considerable interest in multinational computers. For example, Triangle Publications, Inc. assembles eighty-one separate editions of TV Guide every week using its on-line computer communications system in the United States and Canada.[10] United Press International is installing a computer based multinational news network which will link its New York headquarters to a worldwide information storage and retrieval system.[11] Multinational stock quotations services are another example of computerized news services. Very extensive

systems are now in operation by Reuters and Ultronic linking together stock brokers in the United States, Canada, Europe, the Far East and South America.

Finally, there are several examples of multinational corporate information systems within companies but across national boundaries. For example, IBM has a system of interconnected computers which exchange engineering information among its laboratories in Europe and the United States. Ford Motor Company similarly links its British plants to facilities in Germany. Such insurance companies as the British American Insurance Company, Metropolitan Life, the Sun Life Assurance Company of Canada and others link their home offices to terminals in other countries for the purposes of transmitting policy data and other statistical transactions.

The Japanese have been very active in this area. In 1971 Mitsui installed a fully computerized global communications network linking together 115 Mitsui offices in sixty-nine countries, claiming to be the world's largest commercial computer telecommunications network. Mitsui claims that the company's management information system receives information daily, including a wide variety of environmental information regarding market conditions from all of its world-wide locations through its network.[12]

A number of companies have installed at least the rudiments of multinational information systems. For example, Monsanto has centralized its computer facilities in St. Louis but has tied that installation to remote facilities in the United States, Europe, and Latin America for management information system purposes.[13] Gulf Oil has tied its Pittsburgh computer center to twenty-five other computers located in the United States and overseas in order to monitor and control Gulf refining and transportation operations around the world.[14] Vickers International Division of Sperry Rand Corporation has a multinational inventory control system, as does Canadian Pacific Railroad and other companies.

This list of examples in the public and private sectors is far from complete. In fact, more than forty such multinational applications were discovered in the course of the brief literature search conducted for this study and there are certainly many more such applications than are reported in the literature. The point is that the multinational use of computers is no longer hypothetical; it is real and its influence is rapidly expanding. But what will its long term impact be? Nowhere in the literature is there a detailed examination of the likely future consequences of this new development. What will be the impacts upon intergovernmental relations and politics? What will be the impacts upon corporate management and operating practices? What will be the effects upon individuals and whole societies of this intimate linking together through computers and telecommunications of the peoples and nations of the world?

To begin to identify the important issues in these areas, the Social Implications Committee of AFIPS asked the authors to conduct a Delphi study of leading thinkers and researchers in related fields.[15] With the assistance of AFIPS officers, an outstanding Delphi panel was assembled consisting of corporate officers, government officials and computer experts. In all, fifty-seven people participated in the process as listed in the attachment. A carefully prepared list of questions was developed from an examination of the existing literature and extensive discussions with members of the AFIPS Social Implications Committee and other computer experts. Three iterations of the Delphi questionnaire were necessary to clarify issues and to develop a preliminary understanding of the reasons for the positions that the panel took on the issues.

One word of caution is necessary before summarizing the results. The study suffers from all the limitations of the Delphi approach.[16] Moreover, it was not intended to be a comprehensive or systematic (much less scientific) examination of the issues that can arise from the use of multinational computers. This will require a far more detailed and rigorous study than was possible here. The only intent was to have one particular panel help us to define issues so that the professional community would have some guidance as to where to look further in assessing potential impacts. Since the panel had to consist of computer-knowledgeable people, it would be surprising if the data did not reveal an optimistic bias favoring the spread of computer usage. Thus, the reader is cautioned not to consider the data presented here as predictions, but rather simply as the first crude attempt to define a new field of inquiry.

The results will be discussed under five headings which correspond to the five major areas of inquiry of the study. They are: technological aspects, socio-cultural implications, public policy and administration, multinational business implications, and impacts on the developing countries.

TECHNOLOGICAL ASPECTS OF MULTINATIONAL COMPUTER SYSTEMS

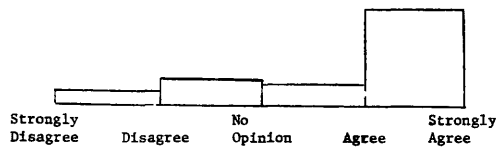
While the study was not concerned primarily with technological dimensions of the MNCS, it was necessary to explore a few of the technical developments that would most affect its use. The entire panel agreed that it would be necessary to increase the speed and reduce the cost of transmitting data through developments in telecommunications such as the following:

1. Improved facilities, particularly satellites;
2. Better switching systems for wide band communications; and
3. Improved and more widespread use of CATV for home communications.

Similarly, 97 percent of the panel agreed that in order to utilize computers across national boundaries on a large

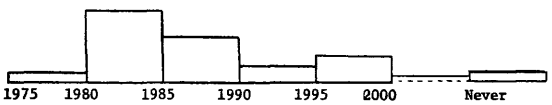
1A Scenario D-1

Global data banks will permit individuals everywhere in the world to share the latest technical knowledge. As a result, there will gradually be a homogenization of the way people understand and interpret problems, and the information they use to solve them.



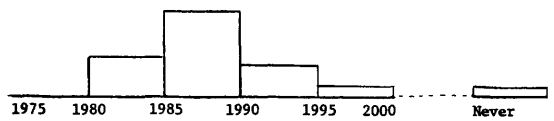
1B Question 2

In what year will a new industry emerge whose function will be to operate multinational computer systems for educational purposes that will be used by at least 250,000 people in five countries?



1C Question 10

In what year will educational data banks be established to service, with on-line capabilities, at least 25 percent of all multinational organizations throughout the world?



1D Question 15

In what year will the first university extend its course broadcasting to satellites, thus making possible a world-wide enrollment in a particular educational program?

Figure 1

scale it would be necessary to provide improvements in portable, on-line and inexpensive terminals as well as smarter and more flexible terminals. The same percentage agreed that it will be critical to improve computer storage systems through providing cheap, high speed associative memories; faster, larger capacity and cheaper secondary multiaccess memories; and safe, inexpensive storage to replace magnetic tapes and discs. Many (92 percent) focused upon the man-machine interface suggesting that improvements were needed in interface support facilities; in cheap, high speed hard copy printers with unlimited character sets; and in more sophisticated man-machine interaction possibilities.

With regard to software, the panel felt that for the MNCS to be widely applied, international agreements on software standardization would be needed, as well as improved large scale multicomputer operating systems and improved software support for "human intellect augmentation." The panel also agreed that cost-effectiveness improvements were required in the development of data management systems capable of handling large, decentralized multinational data bases; in automatic language translation; in automated mass education techniques and in the use of time sharing systems. However, with regard to automatic language translation, a large majority did *not* believe that we would see the "first practical application of voice-to-voice transmission with computerized translation and voice synthesis across the boundaries of at least five countries" until the end of this century if at all. Further, it was felt that once developed, this use of computers would likely be restricted to narrow technical domains with highly constrained query languages.

SOCIO-CULTURAL IMPLICATIONS

One of the main themes investigated in the study was that global technologies such as the MNCS have a tendency to support various forms of cultural uniformity and homogeneity. Most (75 percent) of the panel agreed with the statement that "the use of computers across national boundaries will contribute to the homogenization of cultural tastes and attitudes, although the process that each country goes through to attain this homogeneity may be different, and in some cases may lead to social upheaval." This was further reinforced by the panel's opinion regarding the question illustrated in Figure 1A. We must be careful about interpreting these findings however. While Country A may come to contain the same general spectrum of beliefs, behavior and values as Country B, the spectrum for both may greatly widen from that of today as technologies permit much greater diversity of life styles and actions. As one panelist pointed out, the linkage of computers and communications can individualize the mass media by permitting many more specialized messages to be sent to specialized audiences in place of a few messages sent to everyone.

On the psychological effects associated with multinational computer systems, 84 percent of the panel agreed with the statement that multinational computer systems will increase the level of fear that some people have regarding the mechanization of life they associate with all computer systems. On the other hand, as indicated in Figures 1B, 1C and 1D, the panel strongly anticipates the early use of MNCS for educational purposes, an application that might eventually go at least part way toward negating that fear. In all these questions, the panelists expect great activity in the decade between 1980 and 1990.

A somewhat less optimistic picture emerges with regard to multinational library and data bank applications, as shown in Figure 2. Many panel members felt that the use of these library networks would be so limited in the near

future, so expensive and so difficult to achieve international agreements on, that collaboration as proposed could not be foreseen before the end of the century, if then. Of course, they are more optimistic about scientific data banks (Figure 2C) because of the experience with the NASA RECON System and UNISIST, but even here, they are very sensitive to the potential economic and political barriers. On the other hand, at least several panelists expressed the opinion that there would be mounting political pressures to establish systems of this type since all countries would want to have access to the most current developments in science and technology, and thus political barriers may fall earlier than the majority are willing to anticipate.

Figure 2D illustrates the views of the panel with regard to the future of multinational man-machine interaction.

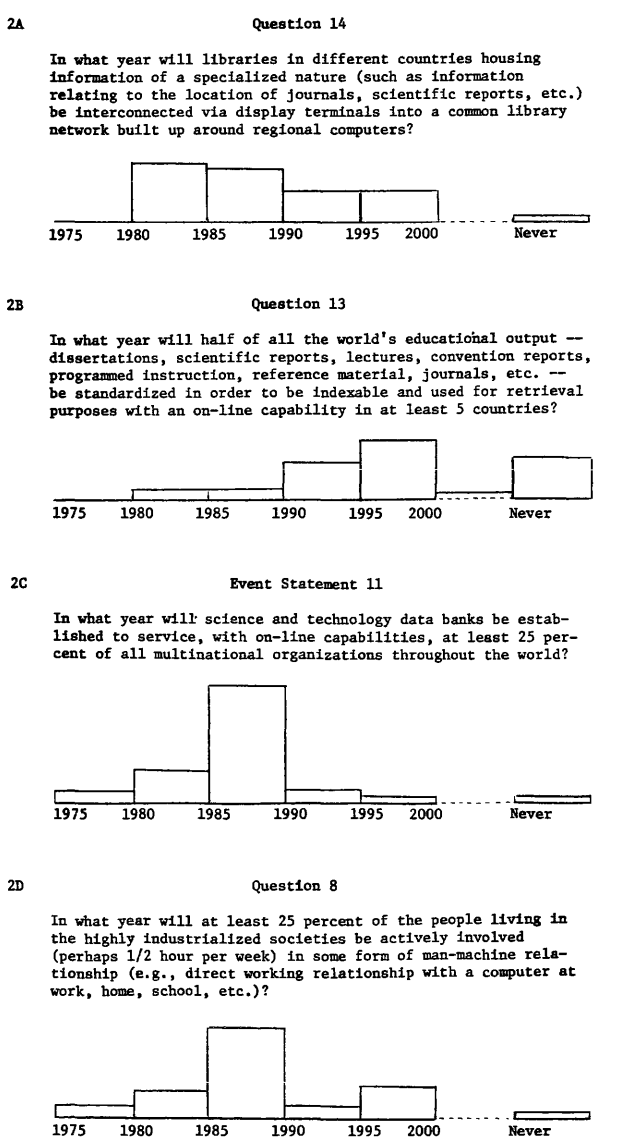


Figure 2

The panel seemed fairly confident that a great number of people living in industrial areas of the world would be interacting with computers by 1990 at the latest. In answer to another question, 72 percent agreed that "the inability to effectively interact with computers will be viewed as a particularly disabling form of illiteracy." Presumably this assumes a considerable simplification in the processes of man-machine interaction and a general movement toward the computer as a major means for augmenting human intellect through access to multinational data bases, on-line conferencing, etc.

PUBLIC POLICY AND ADMINISTRATION

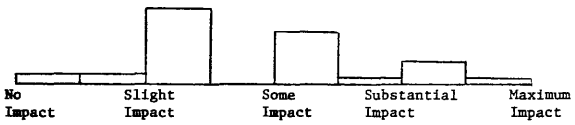
One of the major themes emphasized by the panel members throughout this study was that the use of computers across national boundaries is infused with a public character such that increased public regulation would be inevitable. In fact, one of the distinguishing features of institutions that may emerge from the use of MNCS is that they might tend to blur the distinction between public and private domains of action, since the use of such systems requires the utilization of public communications networks on a large scale. The effects of this interaction, however, are not entirely clear from the comments of the panel members. As shown in Figure 3A many respondents felt that the MNCS had at least some impact on aggravating tensions between nation-states and multinational corporations. Those who expect a high impact do so because they feel that MNCS will make multinational corporations more powerful, but a significant minority (35 percent) believe that the use of MNCS might actually reduce tensions, since it would make visible much information now closely held or hidden by corporations.

Figure 3B suggests that some kind of mechanism might be needed to resolve these tensions and this was confirmed by the fact that 91 percent listed as one of the most important barriers to the use of multinational computer systems "the lack of legal or political mechanisms to resolve conflicts over who controls multinational data banks." Moreover, 61 percent agreed "there is a lack of international coordination in all areas concerned with computer sciences, particularly hardware and software interfacing standards."

Probing further in this area, the panel was nearly unanimous that "the inhibiting role of political ideologies, particularly the notion of national sovereignty, but also the tensions between capitalism and socialism, multinationalism and nationalism, etc." was a major barrier and in fact, this was cited as the single most important barrier to the use of computers across national boundaries. The panel was less concerned with national language differences (only 40 percent cited this), but nearly 70 percent felt that an important barrier was that "nations will be preoccupied with the military significance of multinational data systems, both from the point of view of linking together military allies and from the point of view of protecting such systems from enemy infiltration." Thus,

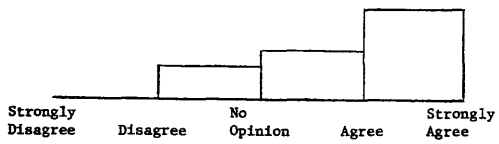
3A Question 34

The use of computers across national boundaries will accelerate the trend toward increased tension between nation-states and multinational corporations.



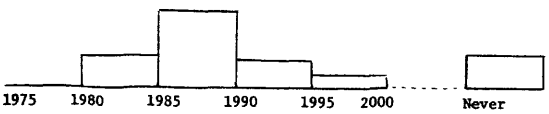
3B Scenario B-4

One result of the use of computers across national boundaries will be that regional "Development Agencies" or some other mechanism will be needed to resolve tensions between nation-states and multinational organizations that emanate from the manner in which these organizations use information to make decisions regarding the goals and objectives of economic development.



3C Question 25

In what year will a mechanism (e.g., an international ombudsman agency) be established among at least 5 countries to adjudicate disputes over issues arising out of the transmission of data across national boundaries?



3D Question 26

In what year will international conventions regarding information flows across national boundaries, including laws on wire-tapping, fraudulent uses of data, etc., be ratified by at least 5 countries?

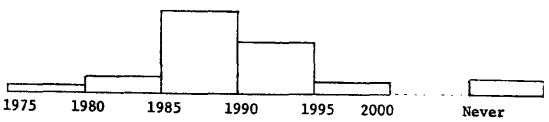


Figure 3

they feel that language and technical differences could be solved much more easily than political barriers, which seem to be the major constraints.

How could these political problems be overcome? Figure 3C may provide some clues in that a majority of the panel believes that some form of judicial machinery (some were very much opposed to the notion of an ombudsman for this purpose) would be needed to adjudicate disputes arising out of data transmission across national boundaries, and that such a mechanism will be developed by 1990 or shortly thereafter. Although some panel members felt that there are too many political barriers for this event ever to come about, others pointed out that the International Standards Association is currently formulating a proposal along these lines and that precedents are being set in other areas such as environmental monitoring and law enforcement. The picture in Figure 3D goes one step further and suggests that panel members do not see international conventions on information flows across national boundaries until at least 1985 and that, even then, there will be serious problems because people from diverse cultures do not necessarily share a common set of values concerning the nature of fraudulent use of data. For example, the United States and Western European countries have differing legal concepts about the nature of price fixing by corporate organizations and the types of collusion that are permissible.

The conflict between the individual's right to privacy and society's right to use computers and data bases for planning purposes is much in the news in the United States. The panel agreed that at the moment, "there is a lack of multinational agreement on the protection of individual privacy and the limits of government censorship on data transfers across national boundaries." When asked in what year there will be an international agreement to protect the privacy of individuals such that agencies of all types would not use data dossiers in ways considered to be repugnant to individuals, most of the panel responded that such agreements would not occur in the next fifteen years and 38 percent even indicated that international agreements on this subject would never be signed. Some of them pointed out that there might be limited agreements of this sort between a few very close neighbors such as the United States and Canada, but they were very pessimistic about larger scale agreements. Some said that differing concepts of individual rights and civil liberties would preclude the possibility of defining the issue at the multinational level, and others felt that even if agreement on definition were possible, it would still be politically unfeasible to design machinery at the global level to police the agreement. If such agreements cannot be formulated, this is likely to raise some very difficult problems in deciding what kinds of data can be transmitted

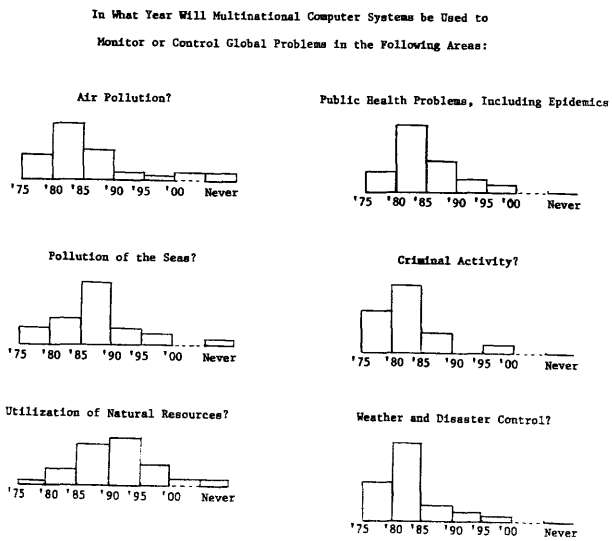


Figure 4

across national boundaries and how the flows could be regulated.

Some of the public areas in which the use of multinational computer systems may have an important impact are illustrated in Figure 4. In all the listed areas, the panelists expected very great progress before 1990, with the possible exception of the utilization of natural resources where the major stumbling block appears to be the national economic plans of individual countries and their reluctance to permit some form of global economic planning or decision-making regarding their resources.

In addition to the areas cited in Figure 4, the panel very strongly agreed that by 1990, at least half of all foreign exchange operations will be carried out by multinational computer systems linking together the large international banks. They were similarly highly confident (i.e., 95 percent agreement) that by 1990, a medical data bank will be established to service, on an on-line basis, at least 25 percent of all multinational organizations throughout the world.

MULTINATIONAL BUSINESS ADMINISTRATION

Earlier in this paper, it was pointed out that there are already many examples of multinational management information systems in operation for various purposes. To explore the proliferation of these systems, the panel was asked in what year at least 25 percent of all multinational organizations would use multinational information systems in dealing with certain problems. As indicated in Figure 5, very great progress is expected in these areas within the next ten to fifteen years. The researchers were particularly interested in the impact of these applications

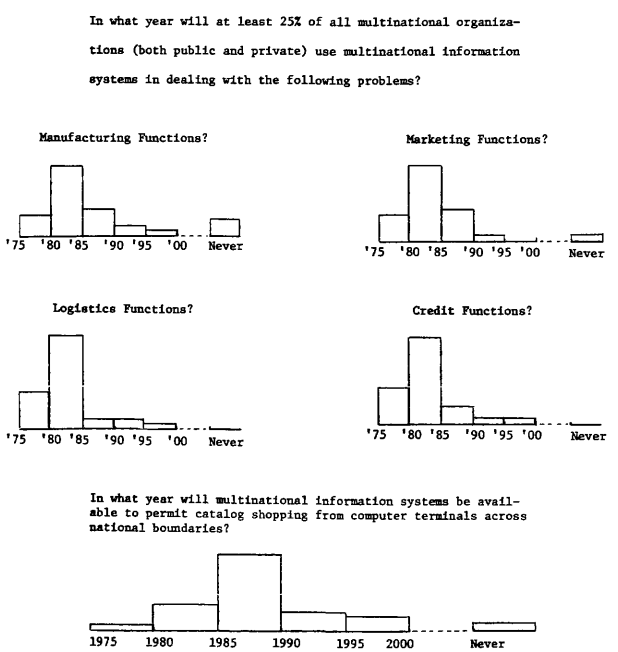


Figure 5

on management theory and practice. Some of the relevant findings in this area are summarized below:

1. There was strong agreement that multinational computer systems will make it possible to more closely integrate the management and research skills of one geographic area with the production and natural resource usage of another.
2. The majority of the panel members (78 percent) believed that computer technology per se, not necessarily the use of MNCS but also including it, would accelerate the trend toward professionalization of management around the world. That is, with the increasing use of computers in big business organizations, the management education required to utilize this technology would likely create uniformities in management behavior and practice in diverse cultural contexts around the world.
3. 78 percent agreed that multinational computer systems will help to enhance the power and influence of multinational organizations whose interests transcend national interests. Since these multinational organizations tend to be financed and dominated by the highly industrialized countries, their increased power and influence may tend to widen the gap between the rich and the poor nations.
4. 80 percent disagreed with the concern sometimes expressed in the literature that multinational computer systems will create large, hierarchical, highly centralized and unflexible management organizations. In fact, according to some panel members, the use of MNCS by a corporation might have quite the opposite effect since the widespread use of remote terminals throughout an organization to collect data, transmit it to a central computer facility, and then receive it back in a form useful for local decision-making, would have a tendency to decentralize the overall structure of multinational corporations.
5. Many panel members agreed that the spread of multinational computer systems will create problems of national citizenship and organizational loyalty. For example, employees of multinational corporations could be asked to make decisions that may not be consistent with the policies of their own governments. Management may be forced to deal with these problems of loyalty at all levels in the organization and some of them may involve legal considerations since individuals may be making decisions within conflicting legal systems (differing laws of nation-states, world law vs. nation-state law, etc.).
6. 60 percent agreed that multinational computer systems will help to redefine the meaning of work and leisure by freeing people who work through distributed computer networks from time and place constraints associated with traditional modes of work. This suggests that people might work in decentralized information centers or perhaps in their home or other choice of location by communicating in the

real time mode with people in other locations, even across national boundaries. Although no time perspective was attached to this statement, a number of panel members believed this alteration of work habits was possible for at least some sector of the population in the next twenty years.

7. 55 percent of the panel agreed that multinational computer systems might have some slight impact, if any at all, on the multinationalization of union activity. Many panel members believe that most union policies will continue to be made at the national level.
8. The panel, by a 66 percent majority, appeared to agree that the use of multinational computer systems would have, at best, a slight impact on the trend toward uniformity in multinational law, including tax laws, anti-trust laws and the chartering of corporations. The feeling was that these matters of law were deeply rooted in national legal philosophies and would unlikely be changed by the use of MNCS. Conflicting notions of monopolistic behavior and effective tax rates would seriously delay cooperation in defining a set of multinational laws regarding the behavior of multinational corporations.

IMPACTS ON THE DEVELOPING NATIONS

The issue of the relationship between MNCS and economic development was explored in several questions of the Delphi questionnaire. The panel strongly agreed that world economic development will be greatly affected by the globalization of information processing. They also agreed, by a 72 percent majority, that the use of MNCS should help the developing countries enormously by tying them into advanced technologies, suggesting that the MNCS itself may constitute a potential vehicle for acceleration of technology transfer. The suggestion was also made by 71 percent of the panel that the developing countries will be motivated to have their own large computers for prestige purposes, much as African countries have done in their investment in airlines. The majority (67 percent) believes that the use of MNCS will make the developing countries "more familiar" to corporations desiring to establish facilities in their country and thereby will accelerate industrialization.

All this optimism does not indicate that the panel expects the developing countries to be uniformly assisted by the development of multinational computer systems. On the contrary, most of the participants do not feel that the use of MNCS would narrow the gaps presently existing between the rich and poor nations. Some observed that a form of "information dependency" might be a likely consequence of this development. Moreover, 55 percent agreed that the developing countries will not have enough people with needed computer skills to benefit from the use of MNCS in the foreseeable future.

Some other positions of the panel with regard to developing countries are summarized below:

1. Most of the panel (90 percent) felt that the use of MNCS could be a strong force in bringing about changes in education, management, medicine and public administration in the less developed countries. Similarly, 92 percent agreed that the use of MNCS will provide opportunities for developing nations to improve public administration in such areas as obtaining statistics, planning and finance.
2. A majority (76 percent) agreed that multinational computer systems will help to institutionalize a method of socioeconomic change whose origins are multinational, thus competing with national strategies of change.
3. A majority of the panel (70 percent) felt that the use of MNCS will have little impact in the developing countries unless developed nations subsidize the development of computer technology in the LDCs and promote the development of the skills and capabilities required to fully utilize computer technology.
4. The impacts of MNCS on the less developed countries are likely to be asymmetric in nature and should be evaluated on a country by country basis. Nonetheless, it will likely become one of the major vehicles by which technology is transferred across national borders and will definitely help to further integrate the developing countries into the world economic structure.

SUMMARY AND CONCLUSIONS

In this paper, we have been able to present only the highlights of a much more detailed study of the possible social implications of the use of computers across national boundaries. The clearest lesson from the study was that there will indeed be some major impacts, and many of them will occur in the next ten years, but that the nature of these impacts are still only vaguely perceived. Nevertheless, certain conclusions seem safe to make at this time.

The use of computers across national boundaries in both the public and the private sectors will expand very greatly in the next two decades as the costs of computation decline, new applications are proven economical, and the scope and influence of multinational organizations increase. On the technical side, there appear to be few barriers to the development of MNCS that are not now already close to solution. The problems that do exist are more in the nature of political or socio-cultural and while there is no guarantee that all these barriers can be swept away in the next two decades, there are promising starts already.

In the public sector, the use of computers across national boundaries will strengthen multinational public

enterprises in such areas as public health, criminal activity, pollution, weather and disaster control, with major impacts before 1985. New institutions will be required at the multinational level to resolve disputes over the transmission of data across national boundaries, to develop regulations concerning the activities of multinational data banks, to provide individual safeguards, and to deal with problems of standardization of data transmission facilities and capabilities. In the private sector, progress may be even faster because much can be done within individual companies. As a result, the use of MNCS is already beginning to enhance the power of multinational corporations vis-a-vis the nation-state while at the same time contributing to a growing uniformity of business practices throughout the world.

These developments are likely to have their most profound and least understood impacts on the socio-cultural level. Within the highly industrialized societies, many people will find themselves in some form of man-machine relationship, often involving multinational communications, within the next decade. These interactions may be for educational, health, library, business, or other reasons but the net effects will be the enhancement of shared beliefs and values and a growing sense of interdependence on matters of the most fundamental nature. One effect of this, for example, might be to create new problems of national citizenship and organizational loyalty whereby individuals will be asked to make decisions in a multinational context that may not be consistent with the policies of their own governments.

In the developing nations, the widespread use of MNCS may be considerably delayed, perhaps for fifteen or twenty years, but when it happens it will have enormous impacts. In the short run, the use of MNCS may tend to enhance the economic interests of the information-rich, wealthier nations at the expense of the information-poor, but in the long run, the use of MNCS will increase the technological options available to the LDCs and speed up their ability to industrialize, and to take advantage of the latest developments in education, management, medicine or public administration. The danger to the developing nations is that the MNCS may distort their investment priorities or lead to policies that favor multinational as opposed to national socioeconomic change. This provides a new challenge to the developed nations to create international organizations and agreements that strengthen the position of the developing countries in regard to all flows of science and technology, particularly the use of MNCS.

In the long run, we may find that the use of computers across national boundaries will be one of the three or four most important factors tending to bring the world closer together through the creation of new multinational institutions and interdependencies. If this should happen, the impact on human society will have been truly revolutionary—perhaps equal to the impact of the invention of the printing press or of human language itself. The main contribution of the data developed in this study has been to suggest that these impacts may begin to be felt sooner rather than later—before 1985 for many of them—and that it is not too early to begin to plan how to avoid the obvious traps and to assure the greatest benefit for the world's peoples.

REFERENCES

1. Two recent examples are Sackman, Harold and Boehm, Barry W., (ed.), *Planning Community Information Utilities*, AFIPS Press, Montvale, New Jersey, 1972 and Greenberger, Martin (ed.), *Computers, Communications and the Public Interest*, Johns Hopkins Press, Baltimore, 1971.
2. Brewer, A. C., "Interactive Scheduling System," *IBM Systems Journal*, 10, 1, 1971, pp. 62-79.
3. "European Information Systems," *Data Processing*, 13, 3 May-June, 1971, p. 199.
4. *UNISIST: Intergovernmental Conference for the Establishment of World Science Information System, Final Report*, Paris: UNESCO Publications, October 4-8, 1971, p. 18.
5. Lusignan, Bruce Kiely, John, (ed.), *Global Weather Prediction: The Coming Revolution*, New York, Holt, Rinehart and Winston, Inc., 1970.
6. "International Reservations Limited," *The Computer Bulletin*, 15, 8, August, 1971, p. 314.
7. Fellows, Philip W., "International Communications Network for SITA," *Sperry Rand Engineering Review*, 21, 2, 1968, pp. 12-16.
8. "G. E. Time Sharing Network Expansion," *Computer Digest*, 6, 6, June, 1971, pp. 2-3. Also see Reehling, Bob, "Time Sharing—Continental Style," *Electronic News*, 14, 731, October, 1969, p. 46.
9. "UCC Inauguration," *The Computer Bulletin*, 15, 7, July, 1971, p. 270.
10. "On-Line Computer Based Composition System," *Computer Digest*, 6, 8, August, 1971, p. 7.
11. "UPI Computerization of News Service," *Electronic News*, 16, 824, June 21, 1971, p. 40.
12. "How Mitsui is Keeping in Touch," *International Management*, 25, 3, March, 1970, p. 51.
13. Harbison, Earl H., "Centralized or Decentralized," *Data Processing*, 12, 5, September-October, 1970, pp. 384-86.
14. "Gulf Oil Installs 370 as Host Unit," *Electronic News*, 16, 844, November 8, 1971, p. 39.
15. Delphi is a method of collecting expert opinion using a series of highly structured questionnaires designed to provide feedback to the panelists while avoiding undue peer pressures. See Helmer, Olaf, *Social Technology*, Basic Books Inc., New York, 1966.
16. Delphi has been criticized by some as not being statistically valid, of not weighing the relative degree of expertness of each participant in relation to each question, of not testing for reliability, etc. These limitations, while very real if the objective is a scientifically reproducible experiment in forecasting, were considered to be less important here where the object was simply to identify issues and explicate positions.

ATTACHMENT I—DELPHI PARTICIPANT LIST

Mr. Leland H. Amaya
Vice President, Data Systems
Pan American World Airways, Inc.
Dr. Arthur G. Anderson
Vice President
International Business Machines
Dr. Paul Armer
Program Coordinator
Center for Advanced Study in the
Behavorial Sciences, Inc.

Mr. Isaac L. Auerbach, President
Auerbach Corporation
Mr. Joseph C. Berston
Vice President
Com-Stute Corporation
Yokohama, Japan
Prof. Davis B. Bobrow, Director
Quigley Center of International Studies
University of Minnesota
Dr. Barry Boehm
Rand Corporation
Dr. Robert Boguslaw
Department of Sociology
Washington University
Mr. Launor F. Carter
Vice President & Manager
Public Systems Division
Systems Development Corporation
Mr. R. C. Cheek, President
Westinghouse Tele-Computer Systems Corporation
Division of Westinghouse Electric Corporation
Dr. Samuel H. Cleff
Vice President, Research & Development
ADP Personnel Data Systems Inc.
Dr. Irving K. Cohen
Rand Corporation
Mr. Joseph F. Cunningham
Dr. Ruth M. Davis, Director
Center for Computer Sciences and Technology
National Bureau of Standards
Dr. Melvin S. Day, Head
Office of Science Information Service
National Science Foundation
Mr. John Diebold, President
The Diebold Group, Inc.
Mr. Oscar A. Echevarria
Management Systems Specialist
Inter-American Development Bank
Mr. Anders Edstrom
Assistant Professor
INSEAD
Fontainebleau, France
Mr. Bob O. Evans
IBM Corporation
Dr. Franco Filippazzi
Director of Advanced Developments
Honeywell Information Systems Italia
Milan, Italy
Mr. R. B. Forest, Editor
Datamation
Mr. Harvey S. Gellman, President
DCF Systems Ltd.
Toronto, Canada
Mr. George Glaser, Principal
McKinsey & Company
Hon. Allan E. Gotlieb
Deputy Minister
Department of Communications
Ottawa, Canada
Dr. Joseph O. Harrison, Jr.
National Bureau of Standards
Prof. Manley R. Irwin
University of New Hampshire
Mr. Gilbert E. Jones
Chairman of the Board
IBM World Trade Corporation
Mr. A. J. N. Judge
Assistant Secretary General
Union of International Associations
Brussels, Belgium
Dr. Peter J. Judge
Head of Section for Scientific & Technical Information
OECD
Paris, France
Mr. Alan B. Kamman
Senior Consultant
Arthur D. Little, Inc.
Dr. Thomas E. Kurtz, Director
Kiewit Computation Center
Dartmouth College
Mr. S. L. Lacks
Assistant Comptroller
The Goodyear Tire & Rubber Co.
Dr. Harold A. Linstone
Senior Editor
Futures Research Institute
Portland State University
Mr. John McLeod
Executive Director
World Simulation Organization of Simulation Councils
Mr. Norman Macrae
The Economist
London, England
Mr. Howard S. Maynard
Systems Manager
ESSO International Division
Standard Oil Company (N.J.)
Mr. Antonio L. de Mesquita
Rio, Brazil
Mr. Roy Nutt
Vice President
Computer Sciences Corporation
Prof. Edwin B. Parker
Institute for Communications Research
Stanford University
Dr. Ithiel de Sola Pool
Professor of Political Science
MIT
Dr. Zenon W. Pylyshyn
Associate Professor
Psychology & Computer Science
University of Western Ontario
London, Canada
Dr. Joseph Raben
Queens College
Dr. Rudy L. Ruggles, Jr.
Hudson Institute
Dr. Harold Sackman
Rand Corporation
Dr. Judah L. Schwartz
MIT Education Research Center
Mr. W. W. Simmons
Planning Consultant
Mr. Gordon Smith
Executive Director
ACM
Mr. Irving I. Solomon
Vice President & Manager
Information Systems Division
National Retail Merchants Association
Dr. George A. Steiner
Graduate School of Management
UCLA

Mr. Ki Soo Sung, Manager
Computing Center
KIST
Seoul, Korea
Dr. Richard I. Tanaka
Senior Vice President, Operations
California Computer Products, Inc.
Mr. Bernard R. Tozer
The National Computing Centre Ltd.
Manchester, England
Mr. Carlos Villa
Management Systems Specialist
Inter-American Development Bank

Mr. F. V. Wagner
Executive Vice President
Informatics, Inc.
Dr. Donald E. Walker
Stanford Research Institute
Mr. Ulf Wennerberg
The Swedish Agency for Administrative Development
Stockholm, Sweden
Mr. Jack W. Wild, Director
Advanced Program Studies
NASA Headquarters
Dr. Robert F. Williams
California Polytechnic University

A new NSF thrust—Computer impact on society

by PETER LYKOS

National Science Foundation
Washington, D.C.

COMPUTER IMPACT ON SOCIETY PROGRAM DESCRIPTION AND OBJECTIVES

Program description

The recently formed (Nov. 9, 1972) Computer Impact on Society Section in NSF's Office of Computing Activities reflected a growing need to understand the wide and deep impact which computers and associated information technology are having on our social organizations and way of life. The program has two principal thrusts, computer impact on organizations and computer impact on the individual, to be implemented via studies and demonstrations. As experience is gained and the program matures, research projects of considerable scope and import may be reasonably expected which will develop ways computer science and technology can be creatively applied to meeting individual and social needs.

Program objectives

The primary significance of the computer for society is its function as part of information technology. The program objectives for the immediate future center on four areas; namely, the computer as a management support tool including computer simulation and modeling, management information systems, and computer-aided conferencing; the role of the computer in the financial systems of the country; citizen access to public data bases, supporting models and other analytical aids, and an improved human/machine interface; and an assessment of public attitudes and perceptions toward the computer, computer literacy, and computer impact on life styles.

Importance of the program in terms of need

The primary significance of the computer for society is its function as part of information technology. As societal systems become more and more complex, information handling and transfer becomes an increasingly important factor. Breakdown of such systems happens generally because of an information problem. In examining such systems, how they operate and how they might be improved, invariably underlying technical problems are uncovered which need to be solved. Formal interaction between the individual and society involves operating within society's legal structure and society's economic structure, both of which lean heavily on information technology. Deep penetrations have been made in: development of computer hardware, both very large and very small scale; computer software both in sophistication of information handling and in transparency to the user; interface with the larger information technology; and in sophistication of communication through the human/machine interface. A large gap exists between current computer science and technology capability, and its actual application supportive to the information technology which enables society to function.

The new program should lead to increased economic capacity and productivity through:

- Improved machine-based management techniques supportive to management of people in organizations and to management of traffic flow be it information, objects, or people.
- Improved techniques of information gathering, validation, storage retrieval, transmission and display, including use of remote servo and sensor devices.
- Improved modeling and simulation methods coupled with better optimization approaches with effectiveness enhanced by more comprehensive and accessible data bases.
- Normalization of laws particularly regulatory laws through machine-based analysis, consistency checks, and presentation of alternatives.

Other societal effects of the proposed program should be better matching of human needs and wants with social resources and opportunities such as students with schools, workers with jobs, goods with buyers, sick and health services, and so on. In addition access by citizen groups to public information, as an aid to intelligent decision making, can improve the common weal.

The NSF's Office of Computing Activities has developed an awareness and close working relationship with the nation's research scholars in the emerging disciplines of computer science and engineering. In addition OCA

has established good working relationships with other elements of the research community through other parts of the Foundation close to the Social Sciences, Engineering, and Mathematics. Accordingly NSF is in a unique position to identify and to attack problems arising at the society-computer interface.

SIGNIFICANT RECENT ACHIEVEMENTS

Although OCA's Section Computer Impact on Society was formed only recently (Nov. 9, 1972), there has been a long-standing and growing concern of OCA with this general problem area, and the following projects supported through OCA are examples of Office achievement in this regard.

1. A project to design a software clearinghouse for local municipalities. The HUD-led USAC is well into a $10,000,000 five-city municipal information system program. However, there are 12,000 local government units which do not have access to that high technology.
2. A project to upgrade a major city's machine-based public information system and to make it available to research professors and graduate students in a local university.
3. A pair of research projects to examine computer-augmentation of technology-forecasting through computer-based conferencing, and to examine in depth multi-participant decision making in a single organization.
4. An in-depth analysis, and generation of corresponding recommendations, regarding the competencies required to design and implement a management information system together with a corresponding description of an extended masters degree program to train corresponding professionals.

MAJOR CURRENT YEAR (FY 1973) PROGRAM EFFORTS

The major focus for the remainder of the current fiscal year (CIS was formed November 9, 1972) was on a set of four intensive two-day workshops addressing the following:

- Public Perceptions and Attitudes toward the Computer, and Computer Literacy
- Simulation and Modeling as an Aid to Decision Making
- Role of the Computer in Banking and Finance
- Role of the Computer in the Legal and Regulatory Processes

In addition to publication of the Proceedings, a major goal was to develop more structure for CIS.

LONG TERM PROGRAM DEVELOPMENT AND IMPACT

The Computer Impact on Society program was formed within the NSF, a part of the executive branch of the federal government and, by its creation, represents an awareness of the need for an action program to address that general problem area. Recent activity in Congress reveals a strong inclination on the part of Congress toward legislation to support civilian research and engineering programs directed at the problems of our society.

Although the program is in its developmental stage and is being funded at a modest level, from a long range point of view it seems reasonable to expect increasing support for projects designed to realize the potential, and to anticipate and avoid the problems of using, computers to help mankind. While it is impossible to describe specifically the problems which the program will address in the future, their general features are apparent, and it is important to consider the way in which this CIS program relates to both our national goals, and to the problems and needs of our individual citizens.

In the economic sector the program should improve the utility and acceptability of automated data bases, which are vitally important to our healthy economic development, by addressing and solving many important problems in accuracy, quality, controlled accessibility, and public perception and understanding. The correspondingly improved management techniques and systems, as well as the improved modeling, simulation and heuristic problem-solving methods, and computer-augmented conferencing which will result from the CIS effort, should lead to significant improvement of the efficiency, accuracy, and quality of decision making in academic business, industry and government. In addition the program should support research in use of real-time computing and distributed intelligence associated with sensor and servo devices.

In the social sphere the work supported under the CIS program will help to enhance significantly the public and professional understanding of computers and information systems and allow a much broader participation in decisions involving their use. Many direct benefits such as the more efficient and "humanized" delivery of services in welfare, medicine, law enforcement, education, maintenance, retail trade, entertainment and transportation should be stimulated by CIS supported activities. The clearer understanding of the social impact of computer systems which will result from CIS supported work may have an important indirect effect on the troubled, but rapidly developing, field of technology assessment. At the individual level, work on technical and attitudinal problems at the man-machine interface should lead to more flexible and natural audio-visual (and perhaps tactile) ways for both professional users and the general public to interact with and draw upon the great resources of information systems, particularly in the delivery of social serv-

ices. It should also result in progress toward directly applying the enormous power of automated systems to the problems of the physically handicapped and other disadvantaged groups.

Environmental concerns will not form a direct part of the program thrust, yet much of the work to be supported in management systems, modeling, and data bases will prove directly applicable. In later years, as the program begins to explore problems in improved techniques and strategies for collecting, reducing, and displaying remote sensor data, particularly in pattern recognition, the results should prove enormously important. Work in selected problems in long range social planning and cost-benefit analysis may likewise prove important from an environmental point of view.

"Social planning" and the "quality of life" are widely used phrases which, because of the diverse range of images they evoke, must be used with some care. Obviously the CIS program will not be, and shouldn't be, in the business of support for decision-making on future social organizational values systems and life styles. But it will be in the business of supporting the development of many of the information system tools which will be important in making such future decisions and it will be actively involved in encouraging a widespread public understanding of, and access to, these tools. From a long term point of view these activities may prove vitally important to the preservation and development of a democratic way of life based on individual freedom, knowledge and dignity.

The benefits which computer projects bring to society are inevitably mixed with new problems and frustrations. In order to minimize negative consequences, the implications of each project should be systematically evaluated. An on-going program emphasizing "computer *impact on society*" would serve to focus attention on both the computer potential and the critical social issues. An important by-product of this direction might be the enhancement of the computer systems involved as a consequence of greater attention to the people-computer interface.

While other areas of impact certainly exist, it should be clear that the potential significance for CIS supported work at both a national and individual level is enormous. Whether this positive impact is realized will depend upon the vigor with which the program is pursued, the adequacy of funding, and the quality of the investigators who can be enlisted to approach problems associated with the impact of the computer on society.

NSF'S COMPUTER IMPACT ON SOCIETY PROGRAM GUIDELINES

The National Science Foundation awards grants to support studies, research, and demonstrations which explore the social impact of computer science and technology and promote creative applications of these fields to meeting individual and societal needs. The primary significance of the computer for society is its function as part of information technology.

The areas for which support is available are indicated below. Projects may contain elements considered within several programs or deal with topics not explicity mentioned here. In addition projects are expected which will overlap with other NSF programs and may be joint funded. The primary focus of the study, research or demonstration will determine which program will consider the Proposal.

Computer impact on organizations

This program supports studies, research, and demonstrations aimed at assessing, designing, and developing creative use of computer and information technology supportive to management and decision making at all levels; anticipating, defining, and making more visible computer impact problems in such areas as law and economics, and encouraging application of real-time computer use in process automation, robotics, traffic flow, and other fields.

Computer impact on the individual

Grants in this program support studies, research, and demonstrations involving use by citizens of machine-based information resources with emphasis on ease of access, accuracy, intelligibility, confidentiality, and related problems; human and technical approaches to improving communication through the human-machine interface especially for non-technical users; and the development of computer and information technology which will service individual human needs and promote the growth of individual talents and interests.

ELIGIBILITY

Guidelines on eligibility and proposal preparation and other helpful suggestions are contained in the NSF pamphlets, Grants for Computing Activities (NSF 71-4), and Grants for Scientific Research (NSF 69-23), which may be obtained from the Foundation.

DEADLINES

Proposals may be submitted at any time.

The impact of technology on the future state of information technology enterprise

by LEE A. FRIEDMAN

Planning Research Corporation
San Antonio, Texas

PREPARING FOR TOMORROW

In the near future the cumulative impact of consumer demands for certain goods and services, eminently absent from current information system inventories, will surface in the competitive market place. At that time many information technology enterprises[1] (see Table I) that are not now prepared to meet the demands will be forced (or, maybe, freely enticed) to alter their courses of action, to hurriedly seek new endeavors and invest heavily in more diverse resources and capabilities. And such extraordinary measures that augment archaic practices, will be employed in order to maintain a survival posture. However, it is not necessary for the subject enterprises to wait for the consumer's broadcast before embarking on ad hoc augmentation adventures.

The intent here is to outline a few principal techniques and programs that can answer cogent questions about preparations for change and survival. As an introductory offer, several outstanding preparatory measures which the subject enterprises should, or will be seriously engaged with in the next 5 to 10 years are listed in Table II. It is almost certain, for the most part, that no radical or revolutionary transformations need occur within a specific enterprise, or even within the industry, especially if the initiation of preparatory programs anticipate or at least attempt to parallel consumer demands. The technological pursuits (Items 1, 2 and 3 in Table II) represent aggregates of expected consumer and market demands that also signal evolutionary trends or technological requirements. These, in turn, will require adequate and timely reactions by the information technology industry, which can only be effected by the initiation of the noted preparatory programs. Before further elaborations on the techniques to derive essential preparatory programs it would be helpful to first put them in perspective.

[1]Table 1 (derived from reference 9) presents a list of pertinent types of enterprises. The general category of enterprises includes profit and non-profit organizations engaged in hardware, software, systems engineering, R&D and allied services associated with the development/uses of information processing systems.

TIMELY POLICIES

Product or mission oriented

Frankly, acquiring and building the kinds of goods and services that can accommodate the impact of evolutionary trends will require no complex or innovative formula. Only a shift in enterprise policy and procedural emphasis is necessary, but a shift that is nonetheless significant because it entails the reallocation of resources and expenditures applied to the production of goods and services. The alternative is to stand still and continue the current operating pattern. Still, as the evolutionary signs grow stronger the subject enterprises will have to make a commitment deciding whether or not to remain with a policy of *product performance maximization* (PROMAX), which has of course been very profitable so far, or to introduce a policy feature that will divert some crucial energies to accomplish what can be labelled as *mission performance maximization* (MISMAX). And it is within the rationale of this latter policy that we shall also find the key thrust of the evolutionary trend.

Retarded robots

The MISMAX policy stems from justifiable and somewhat negative consumer attitudes toward information systems as a viable and effective commodity. MISMAX specifically represents unwritten yet unavoidable invitations *to fix known technological gaps or lags, weaknesses and faults* that hinder maximum application of available system products. Behind these invitations are reasonable people who, while they continue to adopt consecutive product increments, are nevertheless also becoming discontented about the costly imbalances now existing among supposedly complementary system components. Or stated more concisely, these claims are indicative of two prevailing conditions: (1) there are a few major system components (hardware or software or humanware) that seem too advanced for full utilization, i.e., they cannot be used at or near capacity, because complementary components are either inadequate or non-existent;

Table I—Major Enterprises Information Technology

•Equipment and Related Support	•Computer Services
Mainframe (CPU) Manufacturers	Regular Service Bureaus
Peripheral Manufacturers	Time Sharing
Mini-Computer Manufacturers	Network Management
Electronic Components	Libraries
Information/Computer Technology Research	•Other Services
Other Automated Processing Equipment Producers (OCR, COM, Process Control)	Facility Management
	Systems Management
•Software Engineering	Education, Training and Information
Independent Software Producers	Media Suppliers
University, Users	
Government Agencies (as producers)	

and (2) there are obvious and detrimental disparities between the system's automated and non-automated functions that clearly inhibit a system's ability to do what it must or should do (with non-automated functions impacting on the reliability, validity, accuracy, performance effectivity, etc., of the automated).

Shifting commitments

If critical deficiencies persist and expand the hope of achieving significant technological objectives—as mass information system network-utilities, or the intelligent man-machine system, among others noted later on—will remain as idle dreams for some time to come. Furthermore, if the consumer is not able to fully use what he has already, can information technology expect to continue selling product improvements or refinements that could compound the consumer's problem? This market approach, by the way, is representative of a PROMAX policy. Standard PROMAX practices are directed in the main toward iterative redevelopment or improvements of a current product line or technique (by improving or increasing a product's capacity and/or by reducing production costs). Yet, at some point in time a PROMAX program becomes problematically anachronistic and risky, especially a program that is not supported through supplemental programs such as new product/technique R&D. Now, these problem conditions will not impress the consumer as much as they will the technological enterprise. When faced with disconcerting conditions the enterprise will not easily be able to reallocate limited energies, resources and expenditures to cover both current product improvement activities and the development of new and perhaps replacement items being sought by the customer.

Table II—Primary Technology Enterprise Preparatory Program Elements

TECHNOLOGY/MARKET PURSUITS	TECHNOLOGY ENTERPRISE PREPARATORY PROGRAMS
A. Consumer	
1. To develop Lateral Processing System Capability. (Consists of CPU, mass analog/digital data storage, processing unit, and OCR data entry processor - see Table 4, Column IV.)	o Requires increase in complementary hardware, software, and systems engineering capabilities, either within house or between complementary enterprises. o Major Capabilities: - Hardware - optics, analog/digital devices, firmware, micrographics, mass storage devices, microwave/communications - Software - data management and operating systems, simulation-modeling, heuristics, auto-indexing - Humanware - decision-processes, information processes and science, experimental testing, and operations research o Support Capabilities: - Equipment - test/breadboard equipment capabilities; allocation of computer time (or loan of mini-computer for home use, e. g., Datapoint 2) - Other - library expansion, field investigations, professional society papers, joint enterprise/university endeavors o Suggested for enterprises currently engaged in two or more of above major capability categories.
2. To develop special transitional capabilities resolving certain system deficiencies (and also in support of 1., above).	o Requires increase in in-house R&D with associated marketing efforts. o Major Capabilities: - general problem solver (proprietary) packages for MIS - multi-form source media (analog to digital) conversion and message content auto-indexing - micrographics - mass data storage media operating and data management processing - computer network development and management - MIS planning and forecasting functions o Support Capabilities: - As in 1., above o Suggested especially for enterprises that have no hardware manufacturer affiliations.
B. Technology Enterprise	
1. To develop and market special user information system utility-networks.	o Requires increase/augment in professional staff with individuals having broad information science/engineering/communications education background and who also have detailed education and knowledge of user application functions (e. g., biomed, physics, chemistry, public utilities, financial, etc.). To assist in marketing, product development (as in A.2., above) and in system development and testing.
2. To develop technology forecasting and assessment capability.	o Requires staff function to produce technology forecasting, assessment and planning products; to review and develop goals and techniques required to implement preparatory programs; to monitor effectiveness and progress of preparatory programs.

SURVIVAL

Free market factors

The problem of making a timely commitment to more fruitful policies and programs is of course not always so easily resolved. Many technological enterprises cannot afford to engage in both a strict PROMAX policy and supplementary R&D programs that anticipate future market place demands. And in a free market environment how then can we induce prospective victims of evolutionary changes to shift their soon-to-be "fatal" policies? The normal route usually consists of impatiently awaiting the sudden felt signals of supply and demand pressures. But by the time the manifest pressures are felt and understood, by some PROMAX oriented firms it is too late to enable a quick and healthy recovery. So this problematic condition leaves us with a question as to how the subject enterprises can be aided in recognizing and accepting (near-future) technological trends, to assist in the timely reduction of all too risky PROMAX ventures and re-orientation toward MISMAX programs.

Internal factors

Essentially, survival in a free market environment will depend for the most part on how well an enterprise allocates, adjusts or commits its internal factors (resources, goods, services, investments) to accommodate anticipated or forecasted market demands, in the MISMAX sense. In order to identify and regulate internal factors—the degree of preparatory program elements implemented—the enterprise should also engage in systematic technology forecasting and planning programs that serve twin purposes. First, the forecasting program can structure a wholistic picture of relevant external factors that will impinge on the future state of an enterprise's intrinsic factors. Table III presents a succinct list of those factors. Second, the forecasting program can aid in generating plans for preparatory programs, continually structuring the means-end activities required for some period into the future.

Table III—Key Factors

MARKET PLACE			INFORMATION TECHNOLOGY INDUSTRY		
A. Factors Affecting Technological Change			B. Factors Affected by Technological Change		
1. User-Customer	SERVICE INDUSTRY STATES 2. Contr Services	3. Contr Sources	4. Gds/Svcs as Income Sources	5. Resources	6. Capital Expenditures
o Inform. system capabilities, uses: o Requirements, goals o Needs, motives o Investments, expansion o System developers, staff o Industry's plans, programs, priorities o Institutional philosophy, and ideological pursuits	o Complementary o Non-competitive o Competitive (Degree) between: Industry-Industry Industry-Customer Industry-University University-Customer	o Gov't agencies (Fed, State, Local) o Private (Company; Industry) o Non-Profit o In-House	o R&D (for customer and/or in-house) o Turn key Systems o Computer service bureaus o Facility and system mgmt o Hardware components and subsys o Software o Data communication and network management o Media suppliers o Education, training and information (publications)	o Sites, locations, facilities o Staffing (professional staff experience, education, qualifications, performance knowledge) o In-house equipment, laboratories, libraries, support services	o R&D (in-house funds) o Marketing o Dividends (distribution and purposes) o Promotional effort (e.g., advertising, sales), indirect (e.g., publications produced, professional societies) o Expansion, acquisitions (whether diversified, co-generic, vertical) to augment support R&D, to augment capabilities) o Salaries, overhead, administration, fringe benefits

TECHNIC

Attribute mapping

A profitable and cost-effective preparatory program can only be gained when adequate resource expenditures are made available to purchase the necessary baseline forecasting capability. Initial efforts of significant magnitude will therefore be required to create a detailed mapping of each inevitable extrinsic and intrinsic factor. The mapping tasks entail the identification and allocation of sufficient qualitative and quantitative attributes—time scales, probabilities, weighted events and priorities, dollar amounts, utilities, authentications, goals, life spans, among others—associated with each factor, for the essential purpose of generating an indispensable, real-time, forecasting data base. The price of building and maintaining this data base will be returned dollars for pennies, for with this rich vein of information we acquire a better understanding of the events impacting our livelihood and can better control our own forces to maintain a posture of survival.

Table IV—Trends in Information Technology

TECHNOLOGICAL ATTRIBUTE CAPABILITIES I. CURRENT (As Prime Candidates) →	II. TRANSITIONAL PERIOD →	III. FUTURE ACCOMPLISHMENT	IV. SOME REQUIREMENTS IMPOSED ON INFORMATION TECHNOLOGY
A. Data Base Concepts 1. Digital to Digital Conversion 2. (Fragmented/Limited) Digital Data Base	A. 1. Analog to digital data conversion; structural pattern recognition 2. Parallel analog/digital data base (with single source accessibility) (Graphics, pictorial, printed alpha-numerics converted into computer mediated common idiom.)	A. 1. Direct conversion (and auto-indexing) of source media; analog-digital-analog transformation, with analytic pattern recognition 2. Integrated, central data base, consummate data base 3. Integrated analog-digital data base	Development of: A. Lateral System Processing Concept 1. Source media converter, perceptual post-processor and DMS (OCR) 2. Data base storage processing and DMS (for pre-CPU processing) o to enable analog-digital data processing o to store $>10^{13}$ bits/digital and $>(5)10^{9}$ analog images 3. CPU's processing (cognitive) user controlled and generated software 4. OS to control 1, 2 & 3 5. Auto-indexing of message data content meaning, association, value, to provide a measure of info. content and language trans
B. Software (User-Oriented) 1. Algorithmic, proprietary packages 2. Discrete, determinate user programs; languages 3. Linear-serial processing (within boundary of each program) 4. Simplex, discrete and determinate data management systems (DMS)	B. 1. Emulation software; and generalized compilers 2. Generalized problem-solver software packages (rudimentary heuristic and vital processing tech) 3. Auto-indexing (subject meaning) 4. Quasi-parallel data association processing and access 5. Procedure-oriented DMS	B. 1. User generated/controlled heuristic - vital processing software as rudimentary, surrogate (mental) information processing functions 2. Universalized languages/compilers 3. Auto-indexing of message content meaning, idiomatic language transl 4. Question answering DMS	
C. System Structure 1. Fragmented information systems 2. Costly data communications 3. Limited data communications	C. Rudimentary computer networks linking parallel organizations, with non-standard CPU's and I/O languages; user controlled	C. Information system utilities	6. Special purpose packaging - system design, evaluation, application/modeling B. 1. Multi-network chaining and linkages 2. Multi-CPU/language 3. Multi-data base 4. Multi-media (analog-digital) data communication and transmission and display 5. Turnkey system modularity - open ended design to facilitate expanded uses, connectivity to networks and advances in hardware/software including modification of general problem solver software

The first areas to be mapped should be the extrinsic factors depicting technological capabilities, trends and presumed information technology requirements, as outlined in Table IV. Not only is it important to identify each topical subject of concern, but it is equally as important to apply appropriate qualitative and quantitative attributes, the sources of which are noted shortly. These attributes will eventually be applied as criterial elements within the context of a continual comparative analysis that is intended assess the internal status of a subject enterprise's factors in terms of extrinsic market conditions, trends and demands. The main features of these criterial attributes can be easily discerned by seeking complete answers to certain questions, using Table IV entries as topical guides. Procedurally, after deriving relevant "requirements imposed on information technology," one would evaluate the current position and capabilities of information technology and determine: what is and what is not being accomplished, what must be accomplished, when and how, to satisfy the aims of technology. This initial set of steps should then be followed by a determination of: what competitors (and consumers, and funding sources) can and cannot do, what is being done and planned and what they say has to be done. Subsequent to these studies, an enterprise must, of course, map the current state of its own set of intrinsic factors, as an extension of the Table III (Columns 5, 6 and 7) entries.

Valid sources and good indicators

Now, with our abundant supply of information media and channels it should not be too difficult to find proper sources containing the answers. However, the issue of source message validity and reliability or meaning will certainly have to be tackled, given the often intangible and moot nature of the subject matter. Or stated in more practical terms: How does one cull a source message for reliable and meaningful intelligence? And how can one select valid operational indicators that signify, to varying degrees of certitude, the extant state of extrinsic trends that might have potential impact? Providing methods to resolve these problematic issues, unfortunately, is not within the purview of this brief essay. There are, indeed, many ways to identify and measure meaningful information within the context of a well structured intelligence data analysis. But I wouldn't really be skirting the problem by stating that knowledge and application of such techniques are an integral part of a technological forecaster's span of abilities. In this particular case for example, a forecaster would stipulate whether or not singular overt

Table V—Public Sources for Attribute Acquisition

1. Public Sources	2. Message Parameters	3. General Indicators Provided
Notices of "Requests for Proposals" and Contract Awards	o Type of Contract (e. g., R&D, professional services, hardware acquisition, etc.) o Funding source and dollar amount o Types of services, capabilities, technology sought o Type(s) of enterprises bidding o Winners' capabilities (including sub-contractors) o Time period for acquisition/development	o Purchase of new technology desired o Whether contract for new or replacement components; for new or replacement system o Resources-capability: goods/services required-available o Competitor group identification: competitor capability-resource availability, types o Ability of winner to compete on similar contracts during development period o Funding sources
Notices of information systems, product development completed and/or cancelled (also include other new resources)	o Consumer and developer identification o Type/extent of system developed (or, e. g., new facilities built) o (Include parameters from 2., above) o Reasons for contract cancellation	o Source of competition (industry or consumer) o Types of systems/products developed o Types of new technology produced o Inadequate goods/services requiring redevelopment o Capabilities/resources of competitors/consumers
Notices of personnel requirements/changes (in periodicals and news papers)	o Employment advertisements and experience/education required o Notices of principal personnel transfers/re-employment and types of professional experiences	o Resources being sought by competitors/consumers for current, new or anticipated efforts o Resources sought and/or acquired by competitors/consumers indicating new ventures, capabilities, products, services
Reports on activities of industry, association, legislations, legal actions	For example: o Recent ADAPSO position paper on increasing competition by commercial banks which are marketing competing professional and computer services o Brooks Bill (funds for data processing R&D) o Industry program language standards committees o New copyright law o GSA decisions on computer acquisition policies	o Technology and resource trends among competitors o Funding sources o Product improvements o System acquisition policies o Impact on system development costs o Trends on distribution of professional resources

expressions, by several consumers, of either general dissatisfaction with a product or desire for a new product would constitute a reliable and cogent attribute—or an industry consensus—depending on the weighted levels of confidence, certitude or authority assigned (by a forecaster), and supported perhaps by confirmations of past judgments by these spokesmen. However, for the simple purposes of this essay, it is probably more important to disclose some of the intelligence sources, and indicate the purposes they could serve. Table V presents such a disclosure. Here again, unfortunately, the captured intelligence data will be frequently superficial in depth, thereby requiring further probes and investigations to gather a necessary level of detail for the forecast data base.

Analyses

A primary goal of forecast analysis is to reduce, to workable and understandable dimensions, the complexities of present market place conditions in terms of the directions and pathways dictated by selected future states. The forecast approach is intended to expose cogent economic and technological potentials and their contemporary constraints, as well as provide the rationale for crucial preparatory programs designed to equalize threats (competition, enemy technology, disequilibrium) and gain a prosperous market posture. Procedurally, a forecast analysis, which can be performed in several ways to serve many purposes (see References 1, 3, 7 and 11), interprets present conditions and potential in terms of two gross interrogative objectives, viz.: (a) what can we possibly achieve in the future with what we now have (an exploratory forecast); and/or, (b) what must we do now in order to achieve a future goal (a normative forecast)?

Yet, while the results of a forecast analysis might well provide a concise picture, so to speak, of things to come and things to do the picture will not be complete. An enterprise still needs to know how, when, why and with what, before it can make a commitment decision. A more complete set of answers to the latter group of interrogative objectives can be developed through the application of two techniques which are complementary to the forecast function, viz., technological planning and assessment. The utility of these two additive functions can be defined, briefly, in context, as follows. Planning, of course, refers to a method that generates discretely bound and ordered arrangements or sequences of actions (preparatory programs) meant to achieve a specified goal or target. Technological assessment,—a feedforward or ada-predictive operation—refers to a method that analyzes a (series of) preparatory program(s) and ascertains the anticipated benefits and risks of each program alternative in terms of extrinsic factor impact, exploring and exposing the probable consequences and potential of each program.

Synthesis

Although the orientation of this essay leans toward the implementation of a morphological forecast analysis, the attributes of the forecasting data base described earlier can also be applied for assessment and planning purposes (and to other forecast analyses as well, such as microeconomic, heuristic and intuitive techniques). The approach to the joint application of forecast-planning-assessment functions is relatively simple, straightforward and iterative. In essence, the following controlled path should be programmed:

1. Data Base Generation
   ↓
2. Forecasting
   ↓
3. Planning
   ↓
4. Assessment
   ↓
5. Commitment/Decision and Action
   ↓
6. Program Monitoring and Feedback
   ↓
7. Adjustments/Additions to Functions 1→5

To complete this discourse on technique a short description of elementary operational requirements associated with the forecasting, planning and assessment functions is presented in Table VI. Because of the potential complexity involved in accessing, manipulating and relating the range of data base attributes, among other (mental) information processing tasks, it is suggested that these requirements be translated into (EDP) computer software functions, to take advantage of automated information processing capabilities. (Although this latter bit of advice might seem obvious, it is nonetheless offered because quite a few information technology enterprises will not

normally build for themselves what they are paid to build for their customers.)

SURVIVAL STRATEGIES

Technology impact on technology

It should not be surprising that any organized enterprise can at times demonstrate unreasonable human-like qualities, such as manifesting provisional beliefs in immortality or omnipotence. And when operating under this pseudo-protective umbrella an obstinate veil will hide any inner signs of imminent decline or outer events signalling possible repudiation. In this final chapter certain negative (inner) and positive (outer) confirming signs of probable decline are presented. Their disclosure is intended to spark subject enterprise vigilance, and direct attention to signs of the times before the consequential impact of change is felt.

Table VI—Operational Requirements for Forecast Program

FUNCTION	REQUIREMENTS
TREND	Analyze extrinsic factor activities, trends, resources, etc., among competitive, complementary and consumer enterprises; structure in terms of principal goods and services, contracts, resources, percent of market dollar, funds available, captured, expected, etc., - for selected time periods.
FORECAST I: Match Posture -	Compare enterprise's current intrinsic factor levels with those of extrinsic competitor, consumer and other market place elements (e. g., funding sources, new applicable products achieved through research), for each selected time frame. Identify posture, position and pace of the enterprise at each time frame (given that current goods, services, resources, expenditures remain intact through each future period). Flag constraints, degradations, market capability potential and consequences.
FORECAST II: Planned Postures -	Generate alternative market postures (goals) for various potential goods, services, resources and investment levels for each time frame, per selected variations identified by MATCH operation. Identify levels/ranges of goods, services and capital expenditures per available and potential income/contract/funding source, as baseline for generation of multi-level venture plans.
PLANNING: Preparatory Programs -	Device possible (most feasible, cost-beneficial) enterprise preparatory programs - unique combinations of goods, services, resources, investments, expenditures - per each derived market posture for each time frame. Presented as multi-level venture plans.
ASSESSMENT I: Venture Plan Parameters -	Generate scales of payoffs, risks and utilities of selected venture plans given varied levels of enterprise capabilities and market place potential/postures for selected periods into the future.
ASSESSMENT II: Venture Plan Outcome -	Apply combinations of criterial attributes and derive indices of survivability; stipulate consequences (e. g., loss of income, personnel, fringe benefits, et al) if venture plan(s) not implemented at specified time frame, and indicate costs of adaptation if venture plan implemented at later time period.

To briefly illustrate the consequential results of inappropriate reactions to signs of change let us go back in time and review a particular telling event: In the mid-1930's American railroad industry executives produced and narrated a documentary film intended to promote public interest in transcontinental passenger travel. As a bonus offer, or what we might today call a sales gimmick or loss leader, the film's railroad executive-narrator suggested (not too seriously) the possibility of interlinking transcontinental railroad travel with short airline trips, at the passenger's option. Passengers could leave the train at one city where they would board an airplane to transport them to another city en route, at which point they would reboard the train, and so on through to the coast. In a concluding footnote to the film, in reply to the statement that this proposed joint travel arrangement could very well be destined to be *the* future of transcontinental passenger travel, a nodding consensus was expressed by the chorus of railroad executives present—as an indirect boost for a young airline entrepreneur sitting amongst them (it was either Rickenbacker or Lindbergh).

An epilogue to this situation occurred in the mid-1960's, when the film was shown again on national television as part of a series of comedy films. The film elicited a comic effect because the audience pictured Molierian executives slipping on many banana skins in the intervening three decades. Even when subsequent events indicated signs of change these executive decision-makers continued to manifest idée fixe attitudes that overvalued their services, overestimated the sureness of their position, underestimated potential competitors and continued to misinterpret the market place, resulting in the eventual decline of passenger railroad enterprise.

Signs of decline or death

Creating a believable causal link between extrinsic and intrinsic signs of change is often difficult because of a time differential that exists between the occurrence of the two. Extrinsic signs, which normally occur well in advance of intrinsic signs, will not normally have any significant effect on the intrinsic for some initial latency period. (See Figure I, area between t_2 and t_3, for a schematic illustration of this process.) And any enterprise that is a party to this hazy condition faces a problem, for they will indeed have difficulty measuring and confirming the strength and potency of the several diverse signs identified as being precursors of change, since intrinsic reactions will not as yet have occurred—as negative confirming evidence. Given an adequate multi-variate forecasting capability any positive interpretation of the instrumental nature of extrinsic factors could provide sufficient lead time to at least prepare a search for other possibly related clues, attesting further to the probability of change. However, as described earlier, the policy of

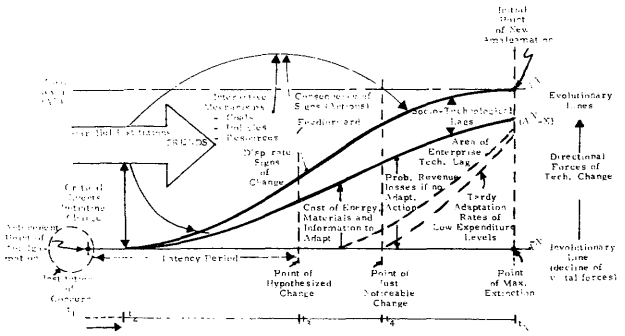


Figure 1—Processes of technological evolution and extinction

many enterprises in a free market is to await the felt pressures of supply and demand. Which is to say that they will await for more direct, certain and tangible signs, viz., uncontrolled impacts on intrinsic factors. But at that point in time extrinsic changes would be well under way, and (as noted in Figure 1, area beyond t_4) the subject enterprise's adaptation-recovery programs would be costlier and riskier.

The following is a listing of selected extrinsic and intrinsic indicators which, this author believes, will serve as the most significant signs of change. (The term significant, here, is, of course, used in a relative sense, its definition being a function of an enterprise's own stated condition, including the possible fact that the enterprise may itself be a major initiator of forecasted technological changes.)

Extrinsic:

- Resources—evidence of continual searches and acquisitions (by industries not listed in Table 1) of professional level personnel representing changes in required educational backgrounds, and/or representing the initiation of new endeavors (competitive)
- Consumer—evidence of continual changes in source of (contracted) services, and resources; in sources and types of information system goods and services acquired and developed
- Competitors—evidence of notable shifts by major competitors in types of resources acquired, released; types of contract services; facilities gained; enterprises acquired; and types of standard goods and services sold, or dropped from inventory
- Funding Sources—evidence of notable increases and decreases in planned and available dollars; allocations of dollars for specific goods, services and resources; shifts in types of winning contractors

Intrinsic:

- Resources—increasing loss (within specified period of time) of principal managers and professional personnel; reduced saleability of current professional capabilities; partial or total lack of resources to enable successful competition for major funding source contracts
- Goods/Services—reduced saleability of current line of goods, services and techniques; partial or total lack of goods, services, techniques to enable successful competition for major funding source contracts; decrease in number of consumer-based contracts and/or decrease in size of contract. (The ultimate test for significance in, for example, reduced sales, would be the occurrence of lost income, profits and higher operating costs.)
- Capital Expenditures—increase in expenditures to maintain market positions, especially those associated with increase in number of competitors for contracts in some goods and services category and/or reduction in contracts won (or RFPs received) in other categories. (The ultimate tests for significance in, for example, expenditure increase, would be the occurrence of extended, below-the-line profit/loss ratios and cash positions.)

Three strategies

This concluding section focuses on three general strategies, the execution of which depends on an enterprise's current posture and prevailing policies. To select an appropriate approach an enterprise would, in terms of its technological forecasting ability, review its current posture to identify strengths and availability of goods, services, resources, capital, income and market potential. In sum, the principal features of the three strategies include:

1. *Immediate*—to implement actions prior to initiation of trend (or to initiate trend) to act as precursory agent. This approach will require risk expenditures the extent of which will be a function of basic capabilities available and required, and the expected market payoff. This strategy would be initiated during the time period shown in Figure 1 as t_2 to t_3, and would rely mainly on the current scope of resources, goods, services and income during this period. Since this strategy entails the greatest gamble, significant changes to these intrinsic factors will probably occur in the post-t_3 period. The successful execution of this strategy, to its fullest, would also require the following types of posture-sustaining actions:
   a. investments in marketing efforts to sell anticipated new products
   b. investigating other marketable areas and capabilities (for periods beyond t_2 and t_3) plus inclusion of technology forecast engineering programs
   c. investments in enterprise promotional activities
   d. investments in management development programs and professional promotional activities (publications, professional societies)

e. investments in extended in-house applied research and development activities
f. investments in in-house basic research activities

2. *Immediate Future*—to engage in limited development/production activities, following-up precursor's achievements, (to save costs of initial development expenditures and risks) to either provide enhancement to precursor's products (a better mouse-trap) and/or to fill in goods and service gaps created by the introduction of new product. Payoff outcome expected to be smaller share of market (but if small enterprise this may be sufficient). The successful execution of this strategy, if engaged during the t_2 to t_3 period, would require some effective investment levels allocated to posture-sustaining action items a. through c., with some minimal attention to items d. and e. Expenditures of time, money and energy in this group of actions can be expected to stabilize in the post-t_3 period.
3. *Extended Future*—to engage in production activities when the market place and new technological products become established. Unless enterprise is able to serve as filler of gaps (as in 2 above), the consequences of a late arrival, with intent to join the stream, will be higher risk expenditures required to immediately improve selected capabilities (resources, goods and services) and adapt to the major changes that have occurred, and with payoff outcome being a small share of the market. Enterprises normally employing this strategy up through the t_4 period and beyond divert little or no expenditures to engage in any of the posture-sustaining action items. Any subsequent efforts, in post-t_3 period, especially, to "catch-up," will have a very high price tag. Enterprises which plan to employ this strategy at the t_3 mark will be required to expend additional investment capital in order to sustain its position (to, e.g., develop a better mouse trap, or acquire new (overhead) professional engineering capabilities, in non-hardware affiliated firms, to pursue and market new techniques).

CONCLUSION

Since most readers have little interest in perusing didactic conclusions this one will be short.

The underlying theme of this essay was to propose a viable class of attitudes, goals and elemental control techniques that could be applied to stabilize the two opposing forces affecting organizational processing—adaptation and extinction—as they are implied in the life and death cycle of a substantive (information) technology-bound enterprise. To be sure, such enterprises, as organismic parts of an institutional body, are not immune to the debilitating effects of budding evolutionary predicates of technological change. Because of the potential hazards in times of rapid external changes, the signs of the predicates must not be quickly interpreted as mere circumstantial evidence, although some may not yet be substantiated. However, the veracity of the signs will certainly be verified by tests of reality, over time, especially as we listen to the final calls by early victims. As a concluding proposition, the following is offered: that the enterprises of information technology must avail themselves of any useful and effective technology that can purchase and sustain a posture of survival.

REFERENCES

1. Ayres, Robert U., *Technological Forecasting and Long Range Planning*, N.Y., McGraw-Hill, 1969.
2. Bernstein, George B., *A Fifteen-Year Forecast of Information Processing Technology*, Naval Supply Systems Command Research, AD 681 752, January 1969.
3. *Information Technology: Some Critical Implications for Decision-Makers 1971-1990*, N.Y., The Conference Board, 1971.
4. Debons, Anthony, (ed.), *Proceedings, NATO Advanced Institute in Information Science*, Working Group Reports, "A Technology Assessment of Information Technology," "Information Utilities and Netting," et al., Pa: (University of Pittsburgh) Silver Springs Conference, September 1972.
5. Farber, David J., "Networks: An Introduction," *Datamation*, April 1972.
6. Hormann, Aiko, "Machine Aided Evaluation of Alternative Designs," in *Proceedings of Environmental Design Research Association* (EDRA) Conference, UCLA, January 1972.
7. Jantsch, Erich, *Technological Forecasting in Perspective*, Paris: Organization for Economic Cooperation and Development, 1967.
8. *Data Base Management System Requirements*, Joint Guide-Share Data Base Requirements Group, November 1970 (Chairman: W. D. Stevens, Skelly Oil Co., Tulsa, Oklahoma).
9. "Proposed Legislation for Computer Research and Development," National Bureau of Standards, *Technical News Bulletin*, 56, September 1972, No. 9.
10. Turmail, Richard L., "EIA Crystal Ball Shows Trends in Electronics to the Year 2012," *Electronic Design*, Volume 13, June 22, 1972.
11. Turoff, Murray, "Delphi and its Potential Impact on Information Systems," in *Proceedings AFIPS Conference*, Volume 39 (FJCC), 1971.
12. Turoff, Murray, "Meeting of the Council on Cybernetic Stability: A Scenario," *Technological Forecasting and Social Change*, Volume 4/2, 1972.

The home reckoner—A scenario on the home use of computers

by CLAUDE A. R. KAGAN and LAWRENCE G. SCHEAR

Western Electric Company
Princeton, New Jersey

PART 1—THE HOME RECKONER "H.R." SET—AN EXTENSION OF THE HOME ENTERTAINMENT CENTER

Certain similarities between the "H.R." set and General Purpose computers currently marketed, would indicate that the community of people engaged in the various phases of that business might be interested in a technically oriented description of the appliance in question, and its potential role in the home.

Currently available models of the "H.R." set make extensive use of integrated circuit modules, as well as a few transistors in those parts where high voltages and currents prevail. The "H.R." set is carefully designed to permit rapid and simple interconnection with the very broad range of models, types, and standards of units currently found in home entertainment centers.

It is hoped that, as time passes, incorporation of certain elements which are now duplicated in the several units shown in Figure 1 in the "H.R." set itself will cause significant price reduction of the entire system complex, and permit more convenient and broader application of some of the features to be discussed in this article.

Figure 1 shows in block diagram form the components of such a fully equipped, and admittedly, "De Luxe" system. It should be observed that the cost of the "H.R." set is commensurate with the investment value of the rest of the system illustrated.

Essentially the "H.R." set serves as a programmed input-output controller for a variety of signal sources and destinations. Furthermore the "H.R." set is capable of processing the signals in their transfer from the number of available input channels to their desired destinations.

In its present form the "H.R." set is modular in construction permitting the rate of growth and extension appropriate to the owner's desires and budget. (Membership in a proposed "Quarterly Module Club" will provide the members with the opportunity to expand their system conveniently, systematically, and economically.)

The basic "H.R." set is furnished in a variety of cabinets to match contemporary tastes in furniture. Custom installation is possible by discarding the attractive but inexpensive housing of the "H.R."

The standard configuration of the basic "H.R." set incorporates, in addition to the necessary power supply, interconnecting jacks, plugs, and connectors, a minimal logical capability which includes a specially wired command module designed to perform the scanning algorithm of a string language, and to execute its twelve most popular primitive functions.

In addition, all models are equipped with a minimum fast access memory pack with a capacity of 8000 words. This pack is divided into Memory Units, where the number of bits allocated to each MU is a function of the particular alphabet used, or the desired application of the moment. Command extension modules and additional memory packs may be added by the user at his convenience, and as required by the sophistication of the control scripts he wishes to use. (It is expected that lending libraries may possibly include the loan of specialized command modules when required to properly make use of the scripts being borrowed.)

Since the most frequent output is via the T.V. set, the basic "H.R." is equipped with a module capable of generating highly legible and pleasing characters in both upper and lower case, as well as arabic and roman numerals and a variety of frequently encountered punctuation marks and special symbols.

Anticipated offerings to members of the "Quarterly Module Club" will be of modules for additional fonts of characters, and to enable the display of these in a broad range of color and hue. The user may also, if he wishes, design and install fonts of his own choosing.

A suggested minimal configuration is shown in Figure 2. With the system shown it is possible to extend very greatly the scope of utility and enjoyment of the elements of the home entertainment center indicated.

Let us examine the elements shown and their function in the system illustrated.

As previously mentioned the T.V. set serves as the prime means of displaying messages originating from the "H.R." set (in addition to its basic function as a receiver of television signals), and also provides through use of the TVHR pointer a means of feedback to the "H.R." set of the user's wishes.

The 16 button keyboard is an inexpensive means of controlling or inputting information to the "H.R." set as well as serving as a channel selector for the T.V. set.

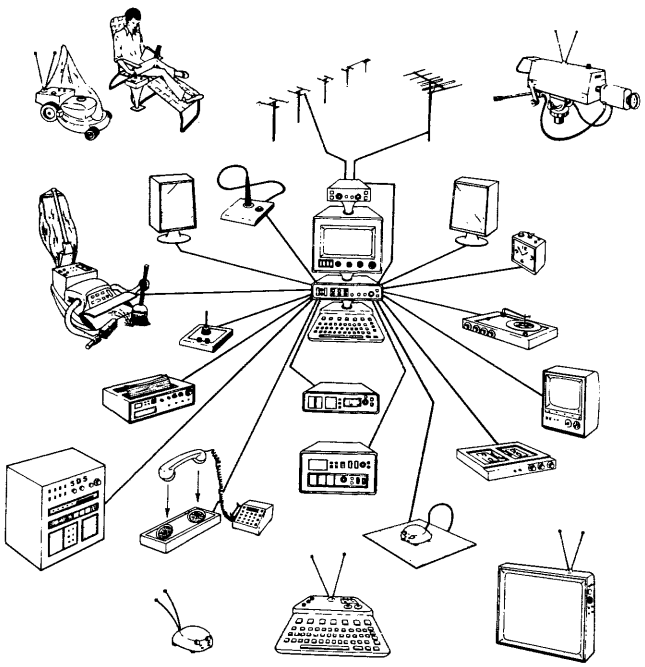


Figure 1

The pair of stereo arranged loud speakers needs no justification.

The HI-FI Stereo phonograph is used to play into the system the wide range of available musical and other offerings as well as the expected equally wide variety of material designed to fulfill the desires and needs of the "H.R." set owners.

One of the albums, popular with young and old, is "PARTY GAMES," which, when played into the "H.R." set will condition the system with the ability of playing literally dozens of games, single player as well as team. This album includes such old time favorites as TIC-TAC-TOE, ONE-ARM-BANDIT, Pin the tail on the BUNNY (Moving)—using the TVHR pointer, BATTLESHIPS, and educational games such as GABBLE, FINANCIER, POST-OFFICE, and furthermore the user can receive coaching in games as CANASTA, BRIDGE, and CHESS.

It is obvious that, although relatively slight, the investment in an appliance such as the "H.R." set should provide a modicum of use other than pure FUN. To this end a variety of educational albums will be offered, initially to charter members of the "SCRIPT of the Month Club" and later through the usual retail consumer outlets.

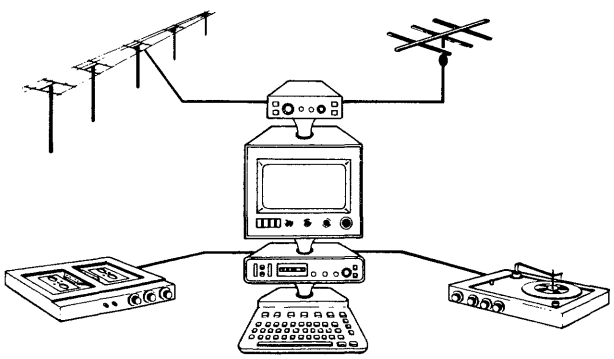


Figure 2

Description of the broad range of amusement and educational albums may be found in Figure 3.

It is expected that as the number of "H.R." sets increases throughout the homes of T.V. viewers we will find wide demand for a future module for the "H.R." set. This module will permit the reception and storage in the memory packs of the "H.R." set of a variety of information to be broadcast either as a sub-carrier or in "QUICKIE" or subliminal burst mode by the T.V. stations. Such information may include weekly program listings, details of news items too lengthy or of such limited interest as not to warrant more than a fraction of a second

Figure 3

of broadcast time, as well as a host of other useful information, including contests, sales promotion and miscellaneous educational items.

In fact it has been suggested that one of the UHF channels in each community be reserved and devoted to the broadcast in a recycling mode of nothing but INFORMATION, educational aids, and other scripts solely for use in "H.R." sets. The user need only select the UHF channel that serves this need in his community, depress the INDEX button, and within a minute or so, when the index is being broadcast, a display of available scripts or "H.R." loads in the current broadcast cycle will be displayed on the screen of his T.V. set. Selection is achieved simply by pointing to the desired item with the TVHR POINTER and again within a minute or so the desired one from a choice of 1800 or so items will reside in the memory pack of his "H.R." set ready for use.

Purveyors of proprietary "H.R." control and other information may also be called upon to sell their offerings through use of the hard-wired telephone switched network. Typical telephone line holding times of the order of ten seconds should fulfill most domestic users' needs, thereby obviating the need for a third telephone line merely to provide home information system needs. Computer Utility Companies will find this feature of the "H.R." set particularly convenient in their attempts to provide ever improving service to the general public.

The modularity, relative small size, and low power consumption of the "H.R." set opens up new vistas to the automobile owner. Sports car enthusiasts will find the combination of the "H.R." set and appropriate interconnections to the automobile's odometer, direction finder, and other navigation aids invaluable in the generation and display of maps and other rally aids.

International travellers will also find the "H.R." set invaluable, in combination with a small T.V. set and their automotive cartridge player set as a conversational mode real-time aid to the understanding of foreign road signs, and even in direct communication with the natives.

Recognizing the fact that neither the schools nor the universities will be able to provide suitable instruction in time to meet the rapid acceptance and acquisition by the nation's technically minded and highly receptive citizenry of this new addition to their homes, a special effort is being made in the interim, by the AOHRSM (Assoc. of Home Reckoner Set Manuf.) to design reasonably standardized but attractive and effective learning aids to enable all "H.R." set owners and users to enjoy to the utmost his latest addition to the home entertainment center.

PART II

Technological forecasting, the Nostradamus of today, shows us the most probable future based on an estimate of the impact of changes. It is a study of trends and probabilities, of psychology and social science. Usually such forecasting is oriented toward product lines, manufacturing techniques, or our changing materials technology. Rarely, outside of the realm of science fiction, does it attempt to relate to the daily life of the so-called average family. Occasionally, however, a serious attempt is made to predict in reasonable detail the effect of one or more facets of changing technology on everyday life in the not-too-distant future.

The Home Reckoner* Set may be thought of as a computer with many peripherals or as a very smart terminal, using today's terminology, in that it may work perfectly well in a stand-alone mode or can be augmented by external storage and/or processing unit. It may be tied

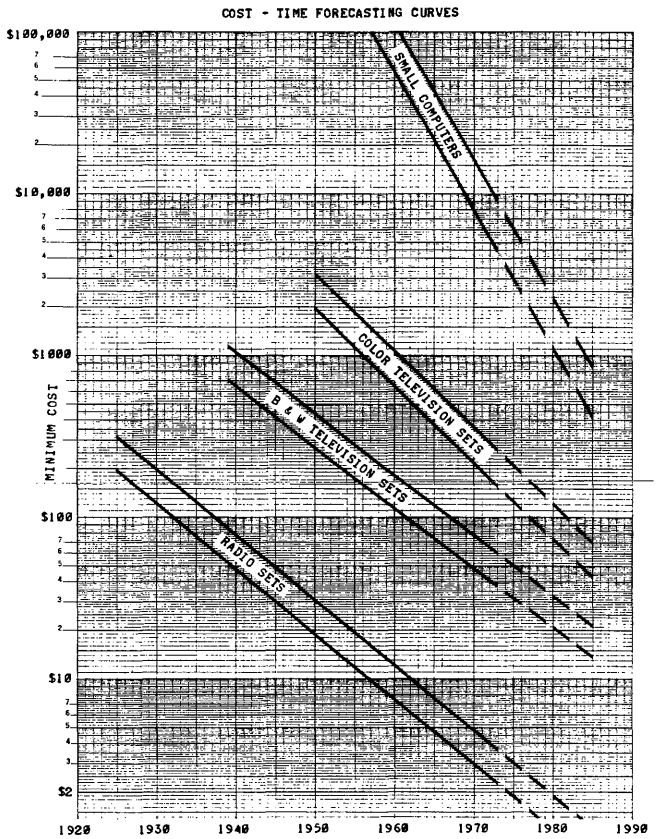


Figure 4

to a central computer, not only for the rapid transfer of data from one memory to another, but also for increased processing power. This large central computer could be located at a service bureau, with different rates depending upon use or time. The Home Reckoner Set as envisioned would be completely modular, being able to operate effectively with whatever peripherals happen to be connected at that time, assuming that they are adequate for the job. Moderate-speed serial interfacing would probably be the least expensive, and would facilitate interconnection of devices from a variety of manufacturers. Each Home Reckoner Set would probably be different, depending on the owner's existing electronic equipment and on whatever peripherals happened to be on sale at his local appliance discount store. Since the concept of the Home Reckoner Set makes as much use of consumer electronics as possible, emphasis is on low cost, and today this favors the moderate-speed serial mode of data transmission among the Various units.

The first part of this paper was written in 1967 by Mr. Claude A. R. Kagan of Western Electric. It was based on a series of graphs tracing through the years the minimum retail price of three items—radios since 1925, television sets since 1945, and "small" general-purpose computers since 1957. These graphs have since been updated.

As can be seen from the curve showing the trend of computer prices in Figure 4, the $2000 computer should become available to the consumer in 1977 or 1978. How-

* The use of the term "reckoner" was originated by Jules Verne in his story entitled, "One Day in the Life of an American Journalist in the Year 2889," first published in English in the October, 1889, issue of the "New York Forum" magazine. In this story, Jules Verne makes a distinction between a computer, which he visualizes as a mathematician's tool, and a reckoner, envisioned as an information processing tool.

ever, the Intel MCS-8 micro-computer is available now with 3K of memory for about $1000. This is in anticipation of the predicting curve. As we know, technological forecasting usually gives us trends, not exact dates. The Home Reckoner Set, as originally envisioned, is not far from reality today. Of course, all the audio-visual equipment is available, including the high-speed data transmission equipment required to enhance the local memory or local capabilities. The possibility of a local service bureau or a Cassette-Of-The-Month Club has as its basis one of the record or tape clubs, or, for instance, the monthly information tape put out for shop foremen by the American Management Association. We already have Learn-By-Cassette tapes covering a wide variety of fields—everything from speed-reading to automobile tune-up to accounting mathematics.

Character generator chips are available today that allow graphic display of characters of all types to be shown on a standard television receiver, and this type of chip is in use in a number of teledisplay terminals, such as the Digi-Log Telecomputer 109. It is already possible to use the television receiver for games and learning with a device called Odyssey, introduced last year before Christmas by the Magnavox Corporation. Odyssey consists of a video modulator, character generator, a pair of positioning mechanisms, a detecting device, and a series of patterned, colored overlays for the television screen. Plug-in cards allow different effects to be generated or controlled on the screen, allowing such games as football, tennis, and name-the-states to be played. Odyssey came with 12 games for $100, with additional games being available for from $3 to a high of $25, that being a light "rifle" capable of across-the-room control of the picture on the screen.

Input and output for the Home Reckoner Set may use the asynchronous serial mode, easily achieved with the Western Digital Systems, Inc. Tr1402A transmitter/receiver chip. The speed, up to 9600 baud, depends on an external clock pulse, and may be driven from either the computer clock or a multiple of the line frequency. The Philips cassette was chosen as the local storage medium because of its size, attractive package, tape protection, and its capability to record medium-speed frequency-shift-keying tones with inexpensive equipment, improvable with a Dolby noise reduction system.

However, program libraries may also be enlarged with pressed vinyl stereo records, available at the local supermarket, which may have data on one channel and music, commentary, or instruction on the other, or data on both. Being ultra-conservative, implying monaural, medium-fidelity records, about one million ISO 8 bit characters may be stored on each disc. High-fidelity quadrasonic recording may more than quadruple this density. An average record collection today may consist of about 50 albums, enough to store about 50 million characters of data. By comparison, an average IBM 360 Operating System may use about 2.5 million characters, with about 60 million available on a standard Disk Pack.

A few years ago, Neiman-Marcus advertised a computer for use in the home kitchen to keep running inventory of the family larder, to make up shopping lists, and to prepare interesting and varied menus. A number of engineers today have computers in their homes, where they are used for everything from income tax computation to student homework problems to architectural designing. Many more people have remote access to a computer via the telephone network. The home computer is still a very rare exception. It has not yet become a status symbol. As with other electronic devices, when it does achieve this position of desirability, as we are sure that it will, the supply of inexpensive, easily interfacable computers will increase, because it will become profitable for manufacturers to make them available. Already the stored-program version of the Standard Logic Inc. C.A.S.H.-8 is available for $600 and a complete Intel MCS-8, including 1 K of RAM for about $1000. Figure 5 shows the Intel CPU and Figure 6 shows the complete computer—only one card!

Fifteen years ago, color television sets cost about $1000 and were the extreme status symbol. Today a color set is available for little more than a seventh of that. The audio cassette tape recorder, main mass storage device of the Home Reckoner Set, sells for as low as $14, including tape. Very few aspects of the Home Reckoner Set require standardization in order to make the concept work. One is the information interchange code, of which the ISO 8 bit code is probably the most logically suitable. The other is the language of the information processor—the command structure which the untrained user interacts with, not the machine language. This is the language in which the centrally available scripts are written, and may very probably be a string language. Libraries today lend books, records, tapes, pictures, and other works of art. It is only a small step to include tapes of computer programs or scripts. Of course, computer libraries already do this, but we are talking about a different medium and a different potential user population than they have.

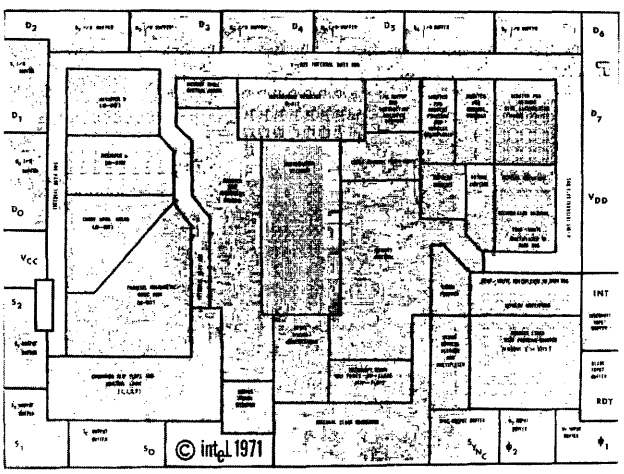


Figure 5

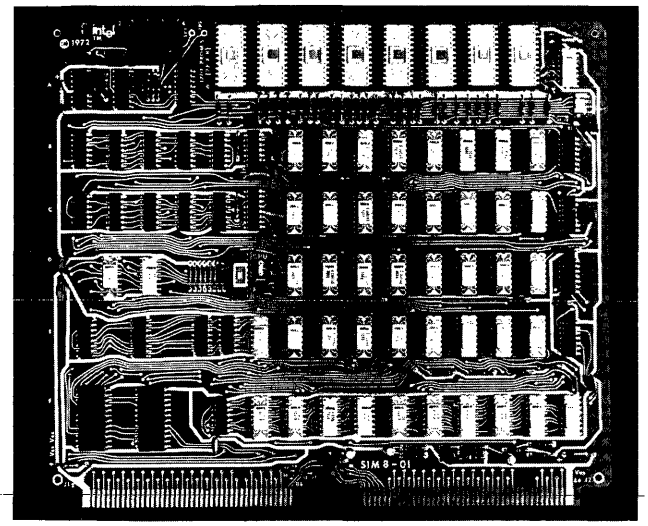


Figure 6

Inasmuch as the Home Reckoner Set allows the user to be an active participant rather than a passive viewer, there would probably spring up a number of users groups, which may or may not be affiliated with the various manufacturers, and at least 2 competing magazines in which new programs, new processors and new perpiherals would be advertised, experimented with, and expounded upon. Eventually, of course, there would be a convention held at least once each year, perhaps in the spring and in the fall, in opposite sides of the country. Lest you think that I am referring to the Spring Joint and Fall Joint Computer Conferences held in days of yore, let me state that similar conferences exist for science fiction fans, camera collectors, stamp and coin collectors, Airstream travel trailer owners, and for just about any other special interest group you'd care to name. The proliferation of CATV and the "wired city" opens additional possibilities to the Home Reckoner Set concept. We then have in the home a potential wide-band communications port which may provide enhanced local communication. Could not some of the useful bandwidth be used for 2-way video communication on a demand-interrupt basis? As time passes, the possibilities increase. As you can see, it is now economically possible to have a useful computer at home. It may be used with a variety of peripherals for business and pleasure, for education and recreation. Computer logic and programming is now taught in high school. More and increasingly younger people are viewing the computer as a tool and as an enrichment to life, not as the manifestation of the Big Brother concept. As the demand increases, the cost of these small but capable machines will decrease further, necessitating a reconstructuring of the forecasting curve. Perhaps the small machines will warrant their own curve! The computer is not yet all-knowing. Only time will tell!

Examples of costs—
asynchronous transmitter/receiver chip Western Digital Systems, Inc. TR1402A—$15
keyboard—Controls Research Corp.—$49
black & white television set—$50 up
color television set—$150 up
Magnavox Odyssey game—$100 up
Data Set/modem—$200 up (high, due to low production)
IBM Selectric typewriter—$550
Panasonic converter—JK-102K—makes a data terminal from a Selectric typewriter—$250
Intel MCS-8 micro-computer—2K PROM, 1K RAM —$1000 up
Hi-Fi stereo set—about $29.95 plus obligation to buy 10 records or tapes during the next 2 years

What's in the cards for data entry?

by GEORGE B. BERNSTEIN

Naval Supply Systems Command
Washington, D.C.

INTRODUCTION

The need for data entry

A data entry system should have three major objectives.

Enter data:

1. in a timely manner;
2. accurately; and
3. economically.

Today, we measure the speed of a computer in microseconds or nanoseconds; however, normally we measure the speed of data entry in hours, days, weeks or months. We measure the accuracy of the computer to 10^9 power and accuracy of data communications to 10^6. However, we measure the accuracy of data input to one error per one hundred or thousand key strokes or to one error per ten or twenty documents.

When people talk about the data entry cost they normally limit cost considerations to the cost of the key entry devices and the people who operate these devices. We tend to ignore the cost of the people at the source creating the documents and the cost of communicating the documents and/or data to the central processing department. Also, we tend to ignore the costs incurred because of inaccurate and untimely information in our computer system. If we are to have effective computer systems for providing management data, business data or technical data, we must have efficient and cost effective methods for entering data into the system in a timely and accurate manner.

Even though data entry generates 30 percent to 50 percent of the total cost of a computer system, the data entry techniques used today are essentially the same as those used in the early 1960's. During the intervening decade, all that has been gained is a small amount of machine efficiency. An International Data Corporation study estimated the data entry workforce for the U.S. at 700,000 people representing a payroll of $3,800,000,000. Virtually all of these people are retranscribing data. The total cost of data entry is approximated as $4.5 billion per year. These figures, while large, do not include the millions of people and billions of dollars spent generating data at the source. Obviously, the driving cost element in data entry is labor. Since data entry is a labor intensive function, there is little to be gained by reducing the cost of a key station a few dollars a month or by adding a few modifications to improve efficiency. Source Data Automation is where it's at.

In the decade of the Sixties, the cost for storing a unit of data in the computer was reduced approximately twentyfold; the cost of processing data in the computer was reduced approximately threefold; and the cost of communicating data to and from the computer was reduced approximately fivefold. Meanwhile, the cost of entering data to the computer has nearly doubled.

It is true that the keypunch has been improved via buffering. It is also true that devices have been substituted to replace the keypunch on a one to one basis. However, the data normally is still retranscribed at the central data processing location and rekeyed for verification. This new equipment has only improved the hardware cost factor by 10 percent to 30 percent. Meanwhile the cost of labor has almost tripled. After debiting and crediting these changes, we find that the total cost of data entry has essentially doubled. Meanwhile, the accuracy and timeliness of data entered into the computer have remained relatively unimproved. While we spent ten years improving the keypunch, we might better have concentrated on eliminating the retranscription function from data entry.

What is data entry

The steps for data entry are as follows:

1. The data is recorded on paper at the source using a typewriter, pencil, accounting machine, cash register, or other device.
2. The paper or data is communicated to the central data processing organization.
3. At the central data processing organization the data is:
   (a) edited and checked;
   (b) keyed;
   (c) key verified;
   (d) errors are corrected; and
   (e) finally it is entered into the computer.

As currently practiced, data entry involves a series of transactions and data handling that requires a combination of people and an elapsed time period that varies from a few hours to a week or more.

Let's carefully differentiate between a data entry device and a data entry system. A data entry device is a piece of equipment used to transcribe data from paper or a keyboard into computer language. A data entry system should combine the generation of the data at the source; communication of the data to the data processing center or computer; and the personnel and equipment involved.

We began by talking about the cost of data entry. But those figures only reflected the cost of data retranscription. They fail to reflect the cost of data preparation in the field and the cost of communications. They do not reflect the cost of the total data entry system. This is true because the responsibility of data operation and communication normally fall outside the responsibilities of the data processing manager.

When most people talk about a data entry system, they emphasize the hardware. However, in most cases when you consider the entire system and system costs, the driving cost factor is the operator. For example, when we compare the cost for the keypunch or teletype, a stand alone system for magnetic tape, and an intelligent terminal against the cost of the operator for various number of hours of useage per month, it is interesting to note that if the terminal is only used a few hours a month, the terminal cost dominates the cost of an operator. However, if the terminal is used 50 percent or 100 percent of the time on just one shift the cost of the operator is three to four times the cost of the terminal. In most data entry situations, we should concentrate on obtaining the equipment that optimizes the efficiency of the operator rather than seek out the cheapest hardware available. We see that data entry must be approached in systems terms. Evaluating any one element of that system in a vacuum may obtain lower initial capital cost—but only at the sacrifice of higher total systems cost, reduced accuracy, and reduced timeliness.

Improving a data entry system

There are a number of ways of reducing the cost of data input. In order of importance they are:

1. Eliminate retranscription. This will probably bring the biggest savings of all. This can be done by capturing the data at the source either through a pencil or typewriter or intelligent terminal.
2. Edit at the keyboard. This removes a considerable amount of clerical functions from the computer main frame and therefore should make a major reduction in overall computer system cost. Additionally, editing data at the keyboard makes it possible to catch errors at an earlier date and makes it possible to correct these errors at a lower total cost.
3. Improve operator efficiency. This enables the operator to key more data in a given period of time with reduced errors.
4. Use data compression and blocking. The amount of data is reduced and only the essential data sent to the computer blocked in a format suitable for rapid entry into the computer system.
5. Improve the equipment used for retranscription. The functions of the keypunch could be improved by making it more efficient to skip, duplicate, correct errors and by utilizing one machine for both keying and verifying the data.

During the Sixties and early Seventies, little has been done to implement the four most effective approaches. The major attempts at cost reduction in data entry have been to improve the keypunch itself or the functions offered by the keypunch. This has been done by improving the keypunch with a buffered keypunch and by attempting to replace the keypunch at the central data center on a one-to-one basis. Normally, this has resulted in a ten to thirty percent improvement in the cost of retranscription.

EVOLUTION OF DATA ENTRY EQUIPMENT

Introduction

Now let's look at how data entry systems have evolved historically. My approach will be from a systems concept viewpoint rather than a chronological viewpoint. This approach will allow us to develop some trends to project the future of the data entry industry.

The evolution of data entry equipment can be divided into five phases:

1. the keypunch;
2. keypunch replacement;
3. keypunch improvement;
4. optical character recognition (OCR); and
5. the intelligent terminal.

Keypunch

The original keypunch was invented in 1890 by Dr. Herman Hollerith. It was part of a punch card system for processing U.S. Census Data. The keypunch inventory grew throughout the decades of the Thirties, Forties, Fifties and Sixties. This led to the development at IBM of the 80 column card and at Univac of the 90 column card.

Key to paper tape

The first attempt to replace the keypunch was made with paper tape equipment. Paper tape development started in 1875 when paper tape was combined with a

printing telegraph. By 1914 paper tape in a typewriter was the basis of most teleprocessing equipment. In 1945 United States Steel Corp. and several other companies stimulated the use of paper tape equipment for entering data at the source in an office environment. This was the first attempt at source data automation.

Direct entry to magnetic tape

In 1951, Univac invented the unityper, a very basic incremental recorder on computer compatible magnetic tape. In 1964 a group of individuals broke away from Univac Corp. and started Mohawk Data Sciences Corporation. The stand alone key-to-tape developments of Mohawk Data Sciences Corporation were quickly followed by other manufacturers such as Sangamo Electric Company, and Minneapolis Honeywell Corporation.

Clusters

In the late Sixties and the early Seventies a number of companies including Computer Machinery Corp., Minneapolis Honeywell, Inforex and Entrex developed a cluster system that combined a number of key stations around a central processor for editing, formatting and pooling data on computer compatible magnetic tape or disc.

Buffered keypunch

In 1967 Univac pioneered improvements in the original keypunch by introducing a buffered keypunch, the Univac Model 1701 followed by the 1710. IBM followed this development with an improved keypunch of its own in 1971, the Model 129. The major features of the improved keypunch is a buffer which permits error corrections to be made by the operator and provides for high speed duplicating and skipping along with limited arithmetic logic for check digit and batch balancing. In addition, the same piece of hardware can be used as a verifier.

Review of one-to-one replacement

It is interesting to note that the improved keypunches and one-to-one keypunch replacements do very little to change the system. While they do simplify error correction and speed up skip and duplicating functions, the total improvement in speed of data entry averages less than 20 percent. This means that we are still left with the problem of carting the documents to the data center and re-editing, keying and reverifying the data.

The major advantage of the clustered key processor systems is the fact that they are based around a minicomputer which can perform the following editing functions: batch balancing, check digit verification, range checks, sequence checking, field validation blocking, and the development of operator production statistics.

While the data entry throughput rate may be the same as that for an improved keypunch, the edit functions offer the added advantage of removing overhead functions from the main frame. The major disadvantage of clustered key processor systems is that the documents still have to be transferred to the central data processing center.

Optical character recognition (OCR)

The fourth phase of evolution in the key entry field was the development of the family of optical character and mark readers which consisted of document readers, journal tape readers and page readers. The optical reader was the second attempt at source data automation. The basic principle of the system was to read documents, tape or pages generated at the source by a pencil or on slightly modified office equipment.

The principal advantages of optical recognition are:

1. a hard copy document is generated;
2. data can be captured at the source, therefore, eliminating retranscription; and
3. it is possible to search the documents.

The major disadvantages of optical recognition are:

1. data cannot be key verified (it must be sight verified);
2. edit features are available at the keyboard; and
3. the principal method of communication is still carting the paper from the source to the central OCR reader.

Intelligent terminal

The latest phase of the evolution in the data entry field is the development of the intelligent terminal. It combines a keyboard, a communications adapter, a mini-computer or intelligent buffer and a typewriter or printer.

The principal advantages of the intelligent terminal are:

1. data is captured at the source;
2. data can be key verified; and
3. data can be edited at the terminal at the source;
4. hard copy is provided;
5. the terminal can be combined with disks, cassettes or tape to create an off-line system;
6. the data can be searched automatically; and
7. it is relatively simple to correct the data.

The principal disadvantage of the intelligent terminal is the cost of the system. Prices range from $2,500 to

$12,000 for purchase and $150.00 to $400.00 per month for rental.

FACTORS RETARDING THE EVOLUTION OF DATA ENTRY

Introduction

In summary, we have improved data entry over the last 20 years by:

1. improving the keypunch;
2. replacing the keypunch on a one-to-one basis;
3. using clustered systems for keying and editing; and
4. using optical character recognition.

With the exception of OCR, source data automation and the elimination of data retranscription have been largely ignored. Also very little emphasis has been placed on improving the efficiency of the operator.

The evolutionary change in the field of data entry has been slowed by a variety of factors which are a combination of equipment, human and market shortcomings:

Equipment

The major equipment shortcomings could be found in the hardware, software, and communications areas. Technical or economical reasons have limited the hardware and software for data entry. For example, the typewriter-to-computer compatible magnetic tape had the disadvantages of lack of hard copy and the high cost of the tape transport. The typewriter-to-paper tape had the disadvantages of no key verifying, as well as difficulty in searching files and correction of errors.

It was difficult to interface the data entry system at a remote site with both the communications and computer system due to different software requirements for the communications system, computer and terminal. For example, even though the ASCII code was pushed as interstate international communication standard, most code systems involved with IBM equipment were the EBCDIC code. Communication systems were expensive and error prone.

Human

People related problems include human resistance to change, empire building, and split authority. It is natural for a human being to be resistant to any form of change. People are set in their ways. In spite of education, desire for improvement, and other factors, they are basically resistant to any form of change.

Empire builders—Data processing managers are notorious empire builders and one of the biggest segments of their empire is the keypunch room. In most cases the data processing managers remuneration is based on the number of people and the amount of equipment under his management. If the data entry function is removed from the data processing department and transferred to the user in the form of source data automation, the data processing manager will lose a large percentage of his people and equipment empire.

Split authority—The data processing manager has the responsibility for retranscription. Line managers have the responsibility for generating the source document. In almost all organizations, no one person has total responsibility for information management. Information is the only major resource of an organization that is not properly managed.

Market

IBM has over 500,000 keypunches and key verifiers in use throughout the United States. The magnitude of this valuable rental base is a major reason why the largest supplier of data input equipment, IBM, has not pioneered new and more efficient devices. If the keypunch were not as reliable and long lasting as it is, I am confident that data entry would not be in its present state of evolution.

A NEW BALL GAME

Introduction

Today the ball game is changing:

1. new technology has made it practical to capture data at the source;
2. over 150 companies with 275 different products have incorporated the new technology to give the potential user a practical input device for any given situation;
3. necessity and education are overcoming the human reluctance to change; and
4. we no longer can tolerate the high cost, inaccuracy, and lack of timeliness of data.

Equipment

There have been several major changes in technology in the last five years that when combined will, or at least should, radically change data input terminals. These changes can be found in:

1. the data storage media;
2. the communication system; and
3. in the size and cost of mini-computers.

The major change in the storage media has been the use of cassettes and floppy discs to replace punch cards, paper tape and direct entry into computer compatible magnetic tape or discs.

Paper tape expanded the record length of the punch card and was easier to correct. Computer compatible magnetic tape and discs gave the advantage of duplicating, skipping and fast data transfer; however, drives and storage media was expensive. Cassettes and floppy discs have the advantage of computer compatible magnetic tape and disc and yet their cost is compatible to punch card punchers and readers and with paper tape punchers and readers. In other words, the user can have all of the advantages of computer compatible magnetic tape at a greatly reduced cost. The only problem with cassettes and floppy discs has been their reliability; however, this has been steadily improving.

A summary chart comparing punched cards, paper tape, computer compatible magnetic tape, computer disc, cassettes, and floppy discs is shown below:

	Punched Cards	Paper Tape	Computer Tape	Computer Discs	Cassettes	Floppy Discs
Error Rate	E	E	G	VG	G	F
Cost of Storage	F	G	F	P	VG	F
Error Correction	P	F	G	E	E	E
Duplicating	G	G	E	G	E	G
Communications	F	G	G	VG	E	E
Searching	G	F	P	E	E	E
Cost of Drive	$500	$800	$3000	$10,000	$1,500	$1000

E = excellent G = good P = poor
VG = very good F = fair

It has always been economical to communicate data from one point to another by utilizing the telephone lines. For example, if an operator key 7,200 strokes an hour during a 6-hour day, this data can be transmitted on a voice grade line and a 1200-baud modem in 6 minutes. This means if data is key punched and key verified at the source, the average operator's normal production can be transmitted in 3 minutes. If we use data compression techniques, the data can be transmitted in even less time.

Recently communication adapters have been developed for individual terminals which makes it practical to use the telephone lines to transmit data from the source to the computer and back. Some of the features of these communication adapters are:

1. hardware interfaces;
2. code converters from EBCDIC, ASCII or Baudot to the code of the terminal;
3. auto answer or polling;
4. parity checks by character and/or blocks;
5. built-in modems; and
6. software interfaces with the communication and computer systems.

Communication adapters having all of the features shown above, can be purchased from $800.00 to $1800.00 and rented from $40.00 to $90.00 per month.

In the past, editing functions were done at the central computer or in a mini-computer in the cluster system. Today, mini-computers with most of the functions listed above, can be built into the terminal for a cost that ranges between $300 and $2,000. This makes it possible not only to enter data at the source, but also to edit and check the data at the source.

Human

The barriers of resistance to change begin to tumble when the people who are resisting the change begin to understand what the change is all about. Education and experience with data processing has resulted in a change in attitude on the part of the people in the organization outside of the data processing department. The computer is no longer considered to be a magic black box and the people who run it are no longer metaphysical magicians. The users can now begin to identify with data processing and they are emotionally prepared to carry out the data entry function.

Simultaneously, we have had budget restrictions placed on the organizations from two points of view. Personnel cuts are hitting the operating divisions of the users. The users are going to have to do their work with fewer people. Meanwhile the data processing budget is being slashed. The data processing budget is no longer a sacred cow immune to top management's cost cutting scalpel.

Today's management can no longer tolerate the high cost, inaccuracy, and untimeliness of data entry. Therefore, data entry must be removed from the data processing department and moved to the source.

Meanwhile, the data processing empire builders have been placed between the crunch bunch—higher volumes of data to process and a lower budget to carry out their function. The data processing manager must now make up his mind to expand his responsibility to become an information manager. In the past he has accepted data and returned processed data; now he must take responsibility for information management rather than data management.

Market

While the keypunch continues to "die on an upslope," IBM has been forced by competition to develop and market alternate methods of data entry. These include an improved CRT system, the 3270 cluster and the 3275 standalone unit; several OCR equipments, the 1287, 1288, and the new 3886 reader; and several recent remote batch announcements such as the 3740 floppy disc system.

In the January issue of Modern Data, L. A. Feidelman spoke of the data entry industry "hearing the footsteps of IBM." However, these announcements indicate that while IBM continues the strategy of attempting to protect its extremely profitable keypunch base, it has also chosen to

slowly introduce a new line of data entry products as a hedge for the future. A hedge against what? These new product lines are designed to meet the challenge of over 150 new companies with 275 different products who are beginning to nibble at that profitable keypunch base.

MARKET FORECAST

Progress in data entry can best be described as evolutionary rather than revolutionary. New innovative data entry techniques and devices are being developed, but there are now over 600,000 keypunch units installed performing 80-85 percent of all data entry work in the U.S..

Short haul

Under these conditions, we can expect the data processing manager to continue an attitude of low risk for change in data entry during the next two to three years. We might call this a period of consolidation. During this period the data entry market will be characterized by the following changes:

1. Small 80-column card keypunch installations will be replaced by buffered keypunch units. New and growing small businesses getting their first experience with data processing will account for a substantial percentage of the keypunch market of this period. Thus, we can see that keypunch equipment will still maintain significant sales.
2. The higher volume keypunch installation (8 or more keypunches) will be penetrated by the central collection keyboard to tape and processor keyboard to disc systems. These systems can be expected to show significant sales volume in 1973 of about $150 million. However, these sales will tend to level off during the 1974-1975 period as many larger users choose to use SDA approaches.
3. Keyboard to tape (cassette/cartridge) units will be widely employed in a source data automation environment. Present keyboard to tape/disc sales are not sufficient to maintain the current large number of companies. There has been and will continue to be an extensive shakeout of vendors. However, the keyboard to tape/disc terminal system will experience marked growth during this period.
4. Keyboard-to-magnetic tape cassette units, with communication capability, will show significant sales growth. In this respect, the keyboard-to-tape unit will be gathering data at the source and transmitting data in bulk during off hours. This application can demonstrate sizeable economic savings and will show significant growth throughout the decade of the 1970's.
5. Smaller installations can be expected to employ the electronic buffered card reader and keyboard-to-tape stand alone systems.
6. The optical mark reader will experience sizeable sales, mainly due to the lack of a reliable alphanumeric handprint reader. OMR sales will progress up to $50 million a year by 1974. However, a low cost alphanumeric hand print reader by 1975 will seriously restrict sales.
7. The portable data recorder is still in its initial phase. The portable data recorder will be employed in selected applications such as production control, sales inventory, and surveys.
8. OCR progress has been slow. This market is dependent upon the low cost OCR reader. The low cost OCR reader, less than $20,000 (and possibly under $10,000) will be commercially available by 1973. These low cost-single font readers will permit typed data to be automatically read. These low cost readers will also be used as remote scanners. The advent of such low cost OCR devices will result in a significant downward trend in keypunch and keyboard-to-tape/disc systems sales. In fact, most data entry equipment will be a combination of low cost reader with a keyboard and display in a multi-media system.
9. We will see a substantial increase in use of terminals, especially the alphanumeric display terminal. Keyboard to tape terminals and remote scanners are just starting. This growth will reflect the trend toward decentralization of data processing activity.
10. Electronic integrated point of sales (POS) systems represent a new data entry category with a potential 250 million dollar annual sales volume. The total POS market holds a potential sales volume of 675 million dollars per year.

Long haul

A business normally uses three types of keyboard devices:

1. the typewriter;
2. the teletype; and
3. data entry equipment.

Traditionally these three types of equipment are made by different industries and sold to different people in the organization. The office manager purchases typewriters, the communication manager purchases telecommunication equipment and the data processing manager purchases the key entry equipment.

By sheer coincidence all three key devices have gone through stages of evolution to where they are almost the same device. IBM pioneered the modification of the typewriter to include a keyboard, hard copy and a dual cassette. Recent studies have indicated that the replacement of the typewriter with the CRT and printer increases the speed and effectiveness of the typewriter MTST dual cassette system by approximately 40 percent.

All the new models of the teletype include a cassette and the more advanced models of the teletype include a CRT. The data entry field is rapidly moving to combination of CRTs, cassettes and printers for source data automation. As a result, the more sophisticated word processing machines, dual processing equipment, communications equipment and data input equipment include the following:

1. the keyboard;
2. a CRT;
3. cassettes or floppy discs;
4. a printer; and
5. a communications adapter.

Even though the hardware equipment has evolved to the same piece of equipment three separate and distinct industries remain to manufacture and market it and in most organizations three separate and distinct people make the basic decision to purchase it. The word processing industry was pioneered by the typewriter people, the teleprocessing industry was pioneered by people who make teleprinters, the data processing industry was pioneered by a multitude of equipment manufacturers who make data entry equipment. Few, if any, companies that make equipment for one of these three fields makes equipment for all three fields. For example, in IBM the data processing division is completely separated from the office products division. Additionally, IBM does not sell equipment to convert cassettes or make any cards in computer language.

It is obvious that if one piece of equipment can perform three separate functions in an office and it is not required fulltime for any one of these functions, then the cost for all three functions can be reduced by sharing the same piece of equipment. For example, in an insurance office terminals are needed for data base inquiry, automatic typewriters are needed, and data input equipment is needed. Also, telecommunications are needed with the home office. Then one piece of equipment can easily perform all four of these functions.

It is reasonable to predict that the companies who make keyboard equipment will start to realize that equipment can and should have multiple functions. Also, it is reasonable to predict that people who purchase keyboard equipment will centralize this purchasing so that the equipment can perform multiple functions and the advantages of volume purchasing can be recognized in the form of discounts. The word processing, teleprocessing, and data processing industries must be merged.

Assessing the regional economic impact of pollution control—A simulation approach*

by J. R. NORSWORTHY

Office of Emergency Preparedness
Washington, D.C.

INTRODUCTION

This paper presents a computer simulation approach to assessing the major economic dimensions of the regional impact of pollution control technology. The framework for the analysis is a regional economic input-output model. Impact measures are estimated in terms of price, employment, and output in the affected region. A 22-sector input-output model of the St. Louis area** is used as the basis for the demonstration. Summary economic impact measures combining price and employment effects are developed for the region and for the rest of the nation. The simulation model is applicable to any set of wage, price, or technology changes which can be specified as changes to an input-output model, and has been implemented as part of the regional economic impact analysis program of the Office of Emergency Preparedness. The full algorithm has been implemented on MCL's UNIVAC 1108 computer for application with the data base for the multiregional input-output model developed by the Harvard Economic Research Project. The price-change portion of the algorithm has also been implemented for the 1963 OBE 478-order interindustry model of the U. S. economy.

An ideal basis for assessing the economic dimensions of environmental protection would compare costs and benefits; however, public policy is leading economic science in that assessment of benefits is lagging far behind pollution control decisions. The simulation procedure presented here is directed only toward the effects of control expenditures; although, as we shall see, there are often benefits in the form of increased income which accrue as the direct result of control expenditures. A particular advantage of a simulation procedure in this case is that several critical variables, most notably demand elasticities, are not known, or are known only with great uncertainty so that repeated simulation for various "scenarios" adds considerable dimension to an application.

We first present a simulation approach to estimating the major economic effects of significant technology changes brought about by pollution control expenditures in the second section. The use of the simulation procedure is demonstrated for comparing two hypothetical pollution control strategies in the third section. We also show in that section an important difference between regional and national perspectives in the choice of environmental strategy; this is highlighted in the demonstration. Generalizations of the model for other purposes, such as assessing the effects of price or wage changes, are discussed briefly in the fourth section. An appendix describes the mathematical basis of the simulation procedure.

The basis of the simulation procedure is an algorithm for estimating the price impact multipliers associated with changes in technology, resulting—in this application—from pollution control expenditures. These changes are expressed in terms of alterations to the interindustry transactions matrix. The price changes are considered as the chief motivation of resource reallocation in the region, and in the nation at large. The price changes induce changes in usage by regional industries and consumers, and by extraregional purchasers as well; that is, output changes for regional industries result from price changes. Output changes are accompanied by changes in the incomes of regional households, and hence in regional consumption spending. (Capital spending by regional industry may also be changed, although we ignore such effects in this demonstration.) Changes in consumption spending induce further changes in output, incomes, and consumption, and so on until a new equilibrium allocation is reached.

THE SIMULATION PROCEDURE

Simulation procedures for the impact model may be summarized as follows:

Step 1. Represent pollution control (or other regional economic phenomenon) expenditures as changes in interindustry transactions matrix.
Step 2. Estimate new interindustry transactions matrix based on changes introduced in Step 1.
Step 3. Derive estimates of regional price changes from new transactions matrix (an iterative procedure).
Step 4. Derive estimates of new income and consumption patterns from employment changes.
Step 5. Estimate new output levels for regional industries from (a) new regional consumption pattern, and

* The views contained in this article do not necessarily reflect the policy or position of the Office of Emergency Preparedness.

** The model is from Liu.[1]

(b) extraregional response to price changes. Loop through Steps 4-5 until changes in income and output converge to zero.

Step 6. Compute price and income impacts for region and rest of nation.

(The appendix shows more generally how new price indices for each industry sector are derived, and how a new transactions matrix is inferred.) We have assumed in this demonstration that

(1) Elasticities of demand for exports from the region are unitary; that is, that the dollar value of expenditures remains the same. Consequently, a one percent increase in price results in a one percent reduction in the quantity sold to extraregional customers.
(2) Elasticities of demand for local consumption are also unitary.
(3) Elasticities of demand for regional interindustry trade are zero; that is, regional production processes are assumed to be characterized by fixed input coefficients. This is the standard assumption underlying input-output analysis.
(4) Increased demand for labor in the region leads to new employment at (implicitly) existing wage rates.

The elasticities of demand are critical to the results of the demonstration; practically, since information regarding demand elasticities at the regional level is typically limited or non-existent, repeated simulations for a range of elasticity assumptions are in order, particularly since substitution possibilities, and hence demand elasticities tend to increase over time. The labor market assumption is less critical; in a tighter labor market, some labor would be reallocated from other industries, but higher wages would be necessary to attract it. Consequently regional incomes would still increase.

The processes involved in obtaining the new transactions matrices and the new prices may be understood intuitively in the following example. In this example, we represent the interindustry transactions matrix as shown in Table I.

TABLE I—Interindustry Transactions Matrix

X_{11}	X_{12}	X_{13}	$\cdots X_{1n}$	C_1	E_1	F_1
X_{21}	X_{22}	X_{23}	$\cdots X_{2n}$	C_2	E_2	F_2
X_{31}	X_{32}	X_{33}	$\cdots X_{3n}$	C_3	E_3	F_3
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
X_{n1}	X_{n2}	X_{n3}	X_{nn}	C_n	E_n	F_n
$X_{n+1,1}$	$X_{n+1,2}$	$X_{n+1,3}$	$\cdots X_{n+1,n}$	C_{n+1}	E_{n+1}	F_{n+1}
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
$X_{n+k,1}$	$X_{n+k,2}$	$X_{n+k,3}$	$\cdots X_{n+k,n}$	C_{n+k}	E_{n+k}	F_{n+k}

where the column index represents the purchasing industry and the row index represents the supplying industry. Thus X_{23}, for example, represents the dollar value of sales from industry 2 to industry 3. Entries in the lower part of the table, indexed $n+1, \ldots, n+k$, represent "value added" activities: payments for labor inputs, state, local, and federal taxes, imports from the regions, profits, etc. In the right hand side, the vector C represents final consumption demand by regional households, the vector E represents external demands (or exports) for output from the region, and the vector F represents all other regional final demands (e.g., investment by regional businesses, purchases by state and local governments, etc.).

The transactions table is constructed so that the row sums (total output for a sector) equal the column sums (total expenditures by a sector) for all n sectors. This balance sheet condition is used in our simulation as a convergence criterion.

The technology of production for an industry is determined by the column vector of its inputs: we simply divide each element in the vector by the column total to obtain

$$a_{ij} = X_{ij} \Big/ \sum_{i=1}^{n+k} X_{ij}$$

and a_{ij} is the expenditure on output from industry i required to produce one dollar's worth of output in industry j.

Step 1. Suppose we wish to study the effects of major environmental protection expenditures in industry 1 in the region; and suppose that these expenditures are used entirely to buy inputs from industry 2. Then in the "first round," we increase industry 1's purchases from industry 2.

Step 2. Two results are immediate:

(a) industry 1 now has a different production process, characterized by greater expenditures per unit of output, and hence a higher price. The new price, relative to the initial price, is simply the ratio of total expenditures after the technology changes to total expenditures before the technology change. That is

$$P_1^1 = \sum_{i=1}^{n+k} X_{i1}^1 \Big/ \sum_{i=1}^{n+k} X_{i1}^0$$

where P_1^1 is the price for industry 1 output in round 1 relative to its price in the base period,

X_{i1}^0 is the dollar value of sales from industry i to industry 1 in the base period

X_{i1}^1 is the dollar value of sales from industry i to industry 1 in round 1 after the technology change is introduced.

(b) Industry 2 now must produce more output to satisfy the additional demand by industry 1 for inputs; industry 2 must therefore buy more of all inputs according to its own technology to produce more output.

On the next round, we must take into account the two effects: higher price for industry 1 outputs, and greater output and inputs for industry 2.

We do so in the following way: since the first row in the transactions table represents sales from industry 1 to all other industries, we can reflect industry 1 passing along the price increase to its customers by multiplying the dollar

transactions in row 1 by P_1^1. (Recall that P_1^1 is an index of price change.)

Thus

$$X_{1j}^2 = X_{1j}^0 \cdot P_1^1$$

where

X_{1j}^0 is the dollar value of sales from industry 1 to industry j before the technology change

and

X_{1j}^2 is the dollar value of sales from industry 1 to industry j after the technology change.

Thus we reflect higher prices for industry 1's output.

To reflect the greater output for industry 2, we construct the "production index"

$$R_2^1 = \left(\sum_{j=1}^{n} X_{2j}^1 + C_2 + E_2 + F_2\right) \Big/ \left(\sum_{j=1}^{n} X_{2j} + C_2 + E_2 + F_2\right)$$

where

X_{2j} represents the dollar value of sales from industry 2 to industry j in the base period.

and

X_{2j}^1 represents the dollar value of sales from industry 2 to industry j in round 1.

In the first round, the only change is in X_{21}^1, reflecting the new environmental protection purchases by industry 1.

We now use R_2^1 to obtain new purchases by industry 2 to increase its own production:

$$X_{i2}^2 = R_2^1 \cdot X_{i2}^0, \qquad i = 1, \ldots, n+k$$

where

X_{12}^0 in the dollar value of sales from industry 1 to industry 2 in the base period

and

X_{i2}^2 is the dollar value of sales from industry i to industry 2 in round 2.

In the second round of transactions (or calculations) two kinds of effects are induced on other industries in the region arising from the initial change in a single cell in the transactions matrix:

(a) price changes resulting from higher prices for output from industry 1, affecting all customers of industry 1

and

(b) output changes resulting from greater purchases by industry 2, affecting all suppliers to industry 2.

In general, then, we must account for these possible effects in all industries. We do so by first adjusting for price changes. We form the ratio of price changes for each industry j

$$P_j^2 = \sum_{i=1}^{n+k} X_{ij}^2 \Big/ \sum_{i=1}^{n+k} X_{ij}^1 \qquad j = 1, \ldots, n$$

and then multiply the *corresponding row* by the price change index

$$X_{ij}^3 = X_{ij}^2 \cdot P_i^2 \qquad \text{for} \quad i = 1, \ldots, n$$

Thus each industry is assumed to pass along price increases to its customers.

For simplicity, we suppose that price and quantity changes take place in different rounds. Now we form for each industry the production ratio

$$R_i^3 = \left(\sum_{j=1}^{n} X_{ij}^3 + C_i + E_i + F_i\right) \Big/ \left(\sum_{j=1}^{n} X_{ij}^2 + C_i + E_i + F_i\right)$$

and obtain new purchases for each industry by multiplying the *corresponding column* by the production ratio

$$X_{ij}^4 = X_{ij}^3 \cdot R_j^3$$

Step 3. We continue this process until the production ratios and price ratios converge to prespecified tolerances; that is, both

$$R^{s+1} - R^s < d_1$$

and

$$P^{s+1} - P^s < d_2$$

where d_1 and d_2 represent the convergence criteria selected in advance. At the end of the process, we have a new transaction matrix which reflects higher prices and greater outputs throughout the system, reflecting the initial change in technology in industry 2. Final price indices may be computed from

$$P_j^f = \sum_{i=1}^{n+k} X_{ij}^t \Big/ \sum_{i=1}^{n+k} X_{ij}^0$$

where t is the index of the computation round when convergence was achieved.

Based on our assumptions up to this point in the simulation algorithm, we have estimates of payments to labor in row $n+1$ of the new transactions table. Suppose that employment is proportional to expenditures on labor. Then we may calculate the relative increase in employment as the ratio

$$W_j = X_{n+1,j}^t / X_{n+1,j}^0$$

Incomes to regional households have increased in the proportion

$$Y = \frac{\left(\sum_{j=1}^{n} X_{n+1,j}^t + C_{n+1}^0 + E_{n+1}^0 + F_{n+1}^0\right)}{\left(\sum_{j=1}^{n} X_{n+1,j}^0 + C_{n+1}^0 + E_{n+1}^0 + F_{n+1}^0\right)}$$

Step 4. In the process of adjusting outputs of each industry by the production ratios, we have changed purchases of all inputs, including labor, based on the assumptions that (a) final demand (C, E, F) remain unaffected by price changes, and (b) that the changes in labor (household) incomes result in no changes in final demands. We now successively relax these assumptions.

The elasticity of demand measures the response in quantity demanded by a commodity to a change in price. Analytically,

$$e = -\frac{\Delta Q}{Q} \Big/ \frac{\Delta P}{P}$$

Thus, for known (or assumed) values of e, we can derive relative quantity changes from the relative price changes derived in Step 2. The simplest assumption, and that which we adopt here, is that $e=1$ for each element of C, E, and F; consequently, the dollar values of C, E, and F remain unchanged in response to the price increases brought about by environmental protection expenditures. Because prices have increased, the same level of expenditures by final purchasers will result in lower physical quantities of output. In other terms, total expenditures by each industry will now exceed total receipts; column sums will exceed row sums. Under these circumstances, we would expect producers to scale down their production; we simulate this effect by forming for each industry the ratio of receipts to expenditures

$$S_j = \left(\sum_{l=1}^{n} X_{jl}{}^{t} + C_j{}^{0} + E_j{}^{0} + F_j{}^{0}\right) \Big/ \sum_{m=1}^{n+k} X_{mj}$$

(For simplicity, we now reindex successive rounds of computations $s+1, \ldots, t$)

On the other hand, increases in household incomes within the region will generate increased consumption, so that we compute

$$C_i{}^{1} = C_i{}^{0} \cdot Y^{1} \qquad \text{for all } i$$

where Y^1 is computed as in Step 3 above. Our procedure corresponds to the simple assumption that consumers spend additional income in the same pattern as before.

Step 5. We now construct a new sequence of operations which adjust at each round of computations for both of these effects.

First, we adjust for the excess of expenditures over receipts. For each industry j, we compute

$$S_j{}^{1} = \left(\sum_{l=1}^{n} X_{jl}{}^{0} + C_j{}^{1} + E_j{}^{0} + F_j{}^{0}\right) \Big/ \sum_{m=1}^{n+k} X_{mj}{}^{0} \qquad j=1, \ldots, n$$

This differs from the expression in Step 4 only in that we replace C^0 with C^1. Because we have done so, $S_j{}^1$ may be greater than or less than 1. We then scale purchases by industry j

$$X_{ij}{}^{1} = X_{ij}{}^{0} S_j{}^{1} \qquad i=1, \ldots, n+k$$

In so doing, we create new income patterns, from which we compute

$$Y = \frac{\left(\sum_{j=1}^{n} X_{n+1,j}{}^{1} + C_{n+1}{}^{1} + E_{n+1}{}^{0} + F_{n+1}{}^{0}\right)}{\left(\sum_{j=1}^{n} X_{n+1,j}{}^{0} + C_{n+1}{}^{0} + E_{n+1}{}^{0} + F_{n+1}{}^{0}\right)}$$

At each subsequent round we obtain

$$C^{s} = C^{s-1} \cdot Y^{s-1}$$

which is entered into

$$S_j{}^{s} = \left(\sum_{l=1}^{n} X_{jl}{}^{s-1} + C_j{}^{s} + E_j{}^{0} + F_j{}^{0}\right) \Big/ \sum_{m=1}^{n} X_{m-j}{}^{s-1} \quad j+1, \ldots, n$$

to obtain

$$X_{ij}{}^{s} = X_{ij}{}^{s-1} \cdot S_j{}^{s}$$

We iterate these procedures until convergence for Y^s and $S_j{}^s$ is obtained in terms of prespecified tolerances. When convergence is achieved, we have obtained the following information:

(1) A new interindustry transactions matrix which reflects the new technology, and consequent new levels of output, income, employment and consumption demands for each industry and for the region as a whole.
(2) A set of price indices which show the effects of the new technology on prices throughout the interindustry model.

Step 6. We can use this information to summarize the effects on the region in terms of impact multipliers based on the original technology change and our assumed values of the elasticities of demand:

(1) a vector of indices of relative (or absolute) production changes for each industry;
(2) a vector of indices of relative price changes for each industry;
(3) a vector of relative (or absolute) employment changes for each industry;
(4) a vector of relative (or absolute) changes in consumption for each industry.

These measures, together with the new technology and Leontief inverse matrices, whose derivations are described in the Appendix, provide a complete description of the regional economic impacts brought about by whatever set of changes are initially put into the transactions matrix.

Policy-makers are likely to be most interested in the effects of policy decisions on incomes (or employment—which in this example is proportional to income) and—perhaps to a lesser extent—on prices. We can readily construct summary measures of these impacts at the expense of industry detail, although this use should be considered only indicative.

The dollar value of the change in regional income may be expressed as

$$W_1 = \sum_{j=1}^{n} (X_{n+1,j}{}^{t} - X_{n+1,j}{}^{0})$$

where t represents the final iteration in Step 5 and 0 represents the original transactions matrix.

Price indices may be computed for each component of final demand (in this case C, E, F) by

$$\mathrm{P1} = w_i \cdot p_i{}^{f}$$

where P_i^f is obtained as in Step 3 above and where

$$w_i = \frac{G_i^0}{\sum_{i+1}^{n+k} G_i^0} \quad \text{and} \quad G = C \text{ or } E \text{ or } F$$

That is, base period weights are used to measure the effect of price increases.

The effects of income changes and price changes may be compared by expressing both in dollar terms. For regional consumption

$$W_2 = \sum_{i=1}^{n+k} (P_1^f - 1) C_i$$

Estimates of the effects of the technology change on the rest of the nation can be derived from (a) changes in "imports" from outside the region and (b) changes in the price of goods shipped outside the region.

Changes in imports from the rest of the nation affect incomes elsewhere; a dollar measure of this effect may be obtained by

$$W_3 = \sum_{j=1}^{n} (X_{n+2,j}^t - X_{n+2,j}^0) \cdot K$$

where t is the index of the final iteration in Step 5 and 0 represents the initial transactions matrix. K is the income multiplier for the nation at large; that is, each dollar of final demand in a national model generates K dollars of income throughout the system. (The value is about 2 for the 1963 OBE interindustry model of the U.S. economy.)

Finally, we may construct a dollar measure of the effect of price changes for goods shipped outside the region:

$$W_4 = \sum_{i=1}^{n} (P_i^f - 1) E_i$$

These summary measures, although crude and heavily dependent on assumed values of elasticities, may be useful in determining the distribution of the effects of a given policy between the region under study and the rest of the nation. It can be argued, for example, that a regional authority will, because of its narrow constituency, tend to favor policies which raise prices of "exported" goods and thereby generate greater incomes within the region, whereas a different policy with less favorable local results would have lower national costs.

A DEMONSTRATION OF THE MODEL

The model has been used to demonstrate the difference between regional and national perspectives in environmental pollution control, based on hypothetical alternative strategies for two industrial sectors.* Some of the results are reproduced and briefly discussed here.

Table II summarizes the industries defined for the St. Louis region.

* These results are taken from a longer study, Norsworthy and Teller.[6]

TABLE II–Sectors in the St. Louis Input/Output Model

Sector No.	Description	Total Production*	Corresponding SIC Code
1	Food, Tobacco & Kindred Products	1,283,957	20–21
2	Textiles & Apparel	197,447	22–23
3	Lumber and Furniture	83,890	24–25
4	Paper and Printing	506,693	26–27
5	Chemicals, Petroleum & Rubber	1,584,148	28–29–30
6	Leather Products	106,859	31
7	Stone, Clay and Glass	219,898	32
8	Primary Metals	851,358	33
9	Fabricated Metals	535,675	34
10	Machinery (Except Electrical)	439,483	35
11	Electrical Machinery	386,357	36
12	Transportation Equipment	3,555,604	37
13	Ordnance & Miscellaneous Manufacturing	293,673	19–38–39
14	Agriculture	88,291	01–09
15	Mining	78,669	10–14
16	Construction	922,382	15–17
17	Transportation, Communication & Utilities	1,481,926	40–49
18	Wholesale Trade Services	870,227	50–51
19	Retail Trade Services	1,350,862	52–59
20	Finance, Insurance & Real Estate	1,656,675	60–67
21	Business, Personal & Other Services	1,483,274	70–89
22	Households	8,065,226	
23	Local Government	601,055	93
24	Other Exogenous Payments	5,511,801	
25	Imports	6,097,453	

* Thousands of 1967 dollars.

The scenario on which this demonstration is based may be described as follows: A regional air pollution control authority must decide between two pollution control strategies of equivalent effectiveness for the local chemical, petroleum, and rubber industries (Sector 5 in the St. Louis input-output model). The sector serves an external market: only 5 percent of total output is old within the region. Table III shows the distributions of expenditures among input-output sectors for the two strategies. Strategy 1, while more expensive, results

TABLE III–Hypothetical Technology Change for Pollution Control in Sector 5*

Sector	Sector No.	Expenditure Change** Strategy 1	Strategy 2
Machinery	10	$14,400	$ 0
Construction	16	18,000	4,400
Utilities	17	6,120	5,400
Households (Labor)	22	25,380	4,700
Local Government	23	900	1,100
Other Exogenous Payments	24	7,200	8,800
Imports	25	12,600	50,600
TOTAL COST		$84,600	$75,000

* Sector 5: Chemicals, Petroleum Refining and Rubber. Production is primarily for export to other regions (95%).
** In thousands of dollars.

TABLE IV–Impact Multipliers for Strategy 1 in Sector 5

Sector	Relative Price Changes for Regional Production	Change in Regional Production	Change in Regional Employment*
1	1.003	2290.	507.
2	1.001	434.	168.
3	1.0020	102.	41.
4	1.0002	200.	45.
5	1.0564	3223.	8158.
6	1.0010	86.	−11.
7	1.0001	−375.	−174.
8	1.0001	956.	306.
9	1.0001	1310.	463.
10	1.0001	14148.	6762.
11	1.0002	19.	−17.
12	1.0001	478.	39.
13	1.0001	138.	36.
14	1.0000	108.	49.
15	1.0001	513.	230.
16	1.0001	17186.	9383.
17	1.0000	7988.	3560.
18	1.0006	729.	133.
19	1.0002	6378.	3598.
20	1.0000	5734.	2057.
21	1.0001	4883.	2937.

* Thousands of 1967 dollars.

in far greater expenditure within the region. Strategy 2, less expensive, depends primarily upon imported equipment.

Table IV shows the impact multipliers on prices, regional production, and regional employment or income for Strategy 1. The major price effects are within Sector 5* where the expenditures take place. Major changes in output occur in Sectors 10 and 16, the major suppliers for the given pollution control technology. A major part of the initial increase in regional incomes in Sector 5 is dissipated ($25,380 down to $8,158) because customers outside the region reduce somewhat their purchases of Sector 5 output. Impact multipliers are not shown for Strategy 2; however, they can be inferred to some extent from Table V, which presents summary measures of regional and national impacts of the two strategies. Here the difference in the distribution of the strategies' impacts is clearly shown.

TABLE V–Summary Measures of Regional and National Impacts

Strategy 1.		
W_1: Change in Regional Income	$38,471	
W_2: Dollar Value of Price Increases in Regional Consumption	$ 5,864	
Price Index: Regional Consumption		100.07
W_3: Change in Extraregional Incomes	−$15,718	
W_4: Dollar Value of Price Increases in Exported Goods	$78,911	
Price Index: Regional Exports		100.68
Strategy 2.		
W_1: Change in Regional Income	−$11,154	
W_2: Dollar Value of Price Increases in Regional Consumption	$ 5,138	
Price Index: Regional Consumption		100.06
W_3: Change in Extraregional Income	$32,374	
W_4: Dollar Value of Price Increases in Exported Goods	$69,955	
Price Index: Regional Exports		100.60

* There are two reasons for this: most of Sector 5's output is exported, and in highly aggregated input-output tables such as this, the diagonal is almost dominant.

First, the price increases fall largely outside the region in each case, as shown by the price indices for regional consumption and regional exports.

Second, suppose we make the tenuous assumption that a dollar increase in prices is offset by a dollar increase in incomes. Then a regional perspective would evaluate Strategy 1 at $38,471 − $5,864 = $32,407, ignoring the negative effects outside the region amounting to —$94,629. Again from a regional perspective, Strategy 2 has a value of $16,292, so that Strategy 1 is clearly preferable. From a national perspective, however, which would include both regional and extraregional effects, both strategies have negative value, but Strategy 2 (at —$52,873) is preferable to Strategy 1 (at —$62,222).

The conclusions from this demonstration are twofold: first, that the characterization of proposed policy-guided technology changes in terms of impact multipliers can provide a useful basis for evaluating policy alternatives; and, second, that a regional model can be used to infer the distribution of impacts between the region studied and the rest of the nation. As a subsidiary observation, we also observe that an authority with only a regional constituency may have incentive to adopt a "beggar-my-neighbor" policy in circumstances similar to those presented here.

GENERALIZATION OF THE MODEL

The simulation procedure which we have described here derives from a simple case of technology change for a single industry in the interindustry model. The procedure can be—and in several instances has been—applied to a variety of circumstances. Several of these applications are outlined here.

National basis

Clearly the simulation procedure does not depend upon the regional nature of the input-output table. Technology change at a national level may be particularly interesting in connection with, for example, substitution of fuels to meet the "energy crisis" forecast for later in this century. At the national level, there is better information about consumption behavior and demand elasticities, so that the results of the macro-economic analysis can be brought into the model. In addition, national tax and other fiscal policies may be studied as adjuncts to the technology change under study.

Estimating the effects of final demand changes

Certain national fiscal policy changes have widely differing regional effects. For example, the SST cutback had its

greatest effect in the Seattle area, and to a lesser extent, on the West Coast generally. Traditional regional input-output analysis estimates only the first round of impacts on regional incomes. Our procedure takes account of the feedback from changes in regional incomes to changes in regional consumption, and so on until a new equilibrium is achieved. Effects on regional consumption and hence on regional incomes are slight for short term disturbances such as strikes; however, major changes in procurement policy have long term effects which are much larger than "one round" analyses imply—as the Seattle merchants can testify.

Our procedure is adapted to study regional effects of this kind by beginning the simulation with Step 4 where changes in final demand are introduced. There are, of course, no price changes brought about by changes in final demand.

Estimating the effects of price changes

During the recent wage-price-rent freeze, there was considerable interest in the effects of price increases in individual industries on prices in other industries, and ultimately on the Consumer Price Index and the Wholesale Price Index. We used the simulation procedure described here to estimate the potential inflationary impacts,* and the actual inflationary impacts of price changes in each of the industries in the 1963 OBE 478-sector interindustry model.** The impacts were estimated only in terms of prices in other industries, and in terms of the GNP deflator, and deflators for GNP components: personal consumption expenditures, residential investment, equipment investment, exports, federal government purchases, etc.

In more general terms the impetus for price changes may be from wage, productivity and/or technology changes provided only that these can be expressed as changes to the interindustry transactions table as demonstrated in Step 1 above.

Estimating time-phased effects

The effects of a time-phased sequence of changes to the transactions table can be estimated by applying the procedure successively. The new transactions table resulting from the set of changes for the first time period serves as the starting point for the changes corresponding to the second time period, and so on. The impact multipliers and other measures of interest are computed as a sequence of impact measures.

Estimating the effects of several simultaneous changes

The simulation procedure as demonstrated above deals with only a single change in technology; however, (1) a technology change may affect several entries in the industry's vector of inputs, and (2) one may wish to consider simultaneous technology changes in several industries at once. The simulation procedure as actually implemented accepts a set of changes to the transaction matrix and computes relative price and production change ratios for all sectors at each round of computations.

APPENDIX: TECHNOLOGY CHANGE IN INPUT-OUTPUT ANALYSIS

Traditional input-output analysis permits derivation of new (relative) prices for an input-output system based on changes in the prices of primary inputs.† This paper demonstrates a simple process which permits inference of the new equilibrium input-output system, including price and technology changes, from stipulated initial changes to an input-output transaction matrix. The effects of these changes may be measured in terms of relative price changes by sector, and change to the technology and Leontief inverse matrices. The technique has been applied to demonstrate the regional economic impact of pollution control technology,[5] and to estimate for the Price Commission the potential and actual effects of price changes on GNP component deflators[6] in the 478-order 1963 OBE interindustry model.

The classic input-output model defines prices in terms of the direct and indirect labor requirements for production of the output of each activity. The analogue to labor requirements in applied interindustry analysis is value added, which is composed primarily of the payments to labor and capital and direct and indirect taxes levied by the various levels of government. These inputs are described in "primary" inputs. An additional primary input—which is not classified as value added in OBE interindustry models—is imports.

Prices in a Leontief input-output system are typically described as follows:

$$P = (I - A')^{-1} A_0'$$

where

- P is a vector of (relative) prices
- I is the identity matrix
- A is the Leontief Coefficients Matrix (excluding the labor or value added row).
- A_0' is the (transposed) value added row of the matrix expressed in dollars per unit of output.

Consequently a given set of changes in the price of value added, dA_0, will result in a set of price changes, dP, in accordance with

$$dP = (I - A')^{-1} \cdot dA_0' \qquad (1)$$

To determine the impact of price increases for any set of industries (the proceeds are assumed to be distributed to value added only) on equilibrium prices in the system (measured in terms of the GNP and GNP component deflators), construct a diagonal matrix D whose components transform price changes into value-added changes.

$$D = \text{diag}\ (1/r_i)$$

* Norsworthy.[7]

** National Economics Division.[5]

† See, for example, Dorfman, Samuelson, and Solow.[1] An application may be found in Leontief.[3]

Thus

$$dP = (I - A')^{-1} \cdot D \cdot dP^*$$

where dP^* represents a vector of first round price changes.

Computationally, we may avoid direct inversion of $(I-A')$ by using the identity*

$$(I-A')^{-1} = I + A' + A'^2 + \ldots . \quad (2)$$

$$= I + A' + A'^2(I+A') + A'^4(I+A'+A'^2(I+A') + \ldots \quad (2a)$$

This series converges when the standard input-output assumptions hold, i.e., when the Hawkins-Simon[2] conditions are satisfied.

We now express equation (1) in terms of the transactions matrix. The algorithm used to estimated dP is based on the power series expression in equation (2a) above. We rewrite equation (1) as

$$dP = [I + A + A2(I+A) + A^4(I - A + A^2(I+A)) \ldots]'.D \cdot dP^* \quad (3)$$

We analyze equation (3) in the following form

$$dP = ID \cdot dP^* + A' \cdot D \cdot dP^* + A'^2 D \cdot dP^* + \ldots \quad (3a)$$

and interpret the first term on the right hand side as the change in the price vector P induced by all activities (industries) "passing through" the first round of price changes; the second term as the change in P induced by passing through the second round of price change, etc.

Let T represent the matrix of interindustry transactions. Then T is related to A by

$$A = D_1 \cdot T \cdot D_2$$

or

$$T = (D_1^{-1} \cdot A \cdot D_2^{-1})$$

where

$$D_1 = \text{diag } (1/P_i)$$

and

$$D_2 = \text{diag } (1/V_i)$$

where

V_i is total output for the ith industry.

Following the above formulation,

$$dP = (I - A')^{-1} \cdot D \cdot dP^* \quad (4)$$

$$= (I - (D_1 \cdot T \cdot D_2)')^{-1} \cdot D \cdot dP^*$$

$$dP = (I + D_1 \cdot T \cdot D_2 + (D_1 \cdot T \cdot D_2)2 \ldots)' D \, dP^*$$

The new transactions matrix T^* (assuming that the quantities of output demanded are unchanged) is

$$T^* = D_3 \cdot T \quad (5)$$

where

$$D_3 = \text{diag } (1 + dP_i)$$

and the new matrix of technological coefficients is

$$A^* = D_1^* \cdot T^* \cdot D_2^*$$

A more general procedure is used to accommodate changes in technology which may be expressed as changes to the transactions matrix.

Let dT represent expenditure changes in interindustry transactions and let dA_0 represent changes in value-added expenditures.

Then

$$A^* = D_1(T + dT) \cdot D_2 \quad (6)$$

and

$$dP = (I - A^{*\prime})^{-1} \cdot dA_0$$

A computation algorithm may be framed in terms of (4)-(7).

* See Waugh.[7]

REFERENCES

1. Dorfman, R., Samuelson, P. A., Solow, R. M., *Linear Programming and Economic Analysis*, New York, 1958.
2. Hawkins, D., Simon, H. S., "Note: Some Conditions of Macro-Economic Stability," *Econometrica*, July 1949.
3. Leontief, W., "Environmental Repercussions and the Economic Structure: An Input-Output Approach," *Review of Economics and Statistics*, August 1970.
4. Liu, Ben-Chieh, *Interindustrial Structure Analysis: An Input-Output Study of the St. Louis Region*, 1967, St. Louis Regional Industrial Development Corporation, St. Louis, 1968.
5. "Input-Output Structure of the U.S. Economy: 1963," National Economics Division, *Survey of Current Business*, Vol. 49, 1969.
6. Norsworthy, J. R., Teller, A., *Estimation of the Regional Economic Impact of Pollution Control*, paper presented at the winter meetings of the Econometric Society, December 1971.
7. Norsworthy, J. R., *Estimated Inflationary Impact of Price Changes in Input-Output Industries*, Office of Emergency Preparedness, 1973.
8. Waugh, F., "Inversion of the Leontief Matrix by Power Series," *Econometrica*, April 1968.

An automated system for the appraisal of hydrocarbon producing properties

by KIM D. LEEPER

PURPOSE OF PAPER

This paper is presented for the following reasons:

(1) To inform petroleum engineers that there does exist an automated system for the appraisal of hydrocarbon producing properties.
(2) To inform the public that one more application of computers to an area that has traditionally been the domain of human decision has been accomplished.
(3) To inform the computer industry of another market (i.e., petroleum producers) for software and hardware.
(4) Stimulating further research in the area of hydrocarbon production prediction and hydrocarbon property appraisal.

CONCERNS OF THE DEPARTMENT OF INTERCOUNTY EQUALIZATION

A major element of the California State Board of Equalization, Property Tax Department Intercounty Equalization program is the completion of approximately 5500 property appraisals per year. Because all locally assessable property is not reappraised annually by county assessors' staffs and because the appraisals that are made by them are not always made to uniform standards, there is disparity among the counties in the relationship of county-appraised to true, current full cash value.

For use in supplying equalization aid to school districts, providing for school construction loan repayments, County Medical Contribution, and other purposes for which equalized assessed values constitute the primary test of ability to raise revenue locally, it is essential that an unbiased state agency measure the assessment level of each county each year. In general this is done (1) by selecting and appraising samples of locally assessable properties and estimating the aggregate full value of the universe of such properties as of the preceding lien date* in each of 19 or 20 counties each year, (2) trending each of these aggregate forward to the succeeding years' lien dates until they are replaced with new ones, and (3) comparing each county's current total locally assessed value with the aggregate full cash value for the corresponding lien date.

The properties identified in the samples are inspected analyzed, and appraised by the Intercounty Equalization Division's appraisal staff, using accepted professional appraisal procedures. The sample results are then ascribed to the universe of locally assessable property to produce, at three-year intervals, an estimate of the full cash value of all locally assessable property in each county.

The types of property appraised include residences, vacant lots, farms and ranches, commercial and industrial enterprises, oil and gas fields, and timber holdings, as well as unsecured personal property. The properties chosen for appraisal constitute a randomly selected sample within assessed-value strata, except for those developed petroleum and water rights which are a significant part of a county's economy. All or substantially all developed hydrocarbon mineral rights in the counties where they are deemed to be a significant part of the economy are appraised in every three-year cycle, and their values are trended between appraisal years separately from the trending of other property values. Appraising all these properties in a county (and designating this group as Stratum 10), instead of a random sample is considered more reliable than appraising only a sample because an oil and gas field can be appraised more accurately as a unit then as a group of separate wells and undeveloped well sites.

One part of the Intercounty Equalization Divisions program is to determine the value of all hydrocarbon producing properties in those counties where the value of such properties is greater than 3 percent of the total county roll. Originally, 2 percent was recognized as the statistically significant limit for sampling purposes (12 of California's 58 counties were in this category). In succeeding years, the number of fields and secondary recovery projects have increased, with a corresponding increase in the appraisal workload. In June 1969, the statistical limit was raised to 3 percent of the county roll to compen-

* This date is March 1 of every year.

CALIFORNIA STATE BOARD OF EQUALIZATION
INTERCOUNTY EQUALIZATION DIVISION

ESTIMATED VALUE OF PETROLEUM MINERAL RIGHTS
IN STRATUM 10 COUNTIES 2/

Estimated Petroleum Mineral Rights Value

County	Market Value
Fresno	$ 175,683,200
Glenn	10,377,200
Kern	917,695,100
Santa Barbara	184,982,400
Solano	115,224,200
Sutter	31,619,800
Ventura	183,107,500
	$1,618,689,400

Note: Petroleum resources subject to valuation in counties other than Stratum 10, i.e., where petroleum assessments are less than 3 percent of entire local assessment roll, are presently valued as part of the randomly selected appraisal workload.

Major petroleum producing counties presently not in Stratum 10:

Colusa
Contra Costa
Los Angeles
Monterey
Orange
Sacramento
San Joaquin
San Luis Obispo

2/ Stratum 10 counties are those counties whose present petroleum mineral rights constitute more than 3 percent of the counties market value.

Exhibit A

sate for this increased workload (7 counties are not in this category; See Exhibit A). Since then, the trend in fields and secondary recovery projects has continued to increase to a point where it is becoming very difficult to continue a quality appraisal program for petroleum properties with the present staff. (See Exhibits B, C, D and E).

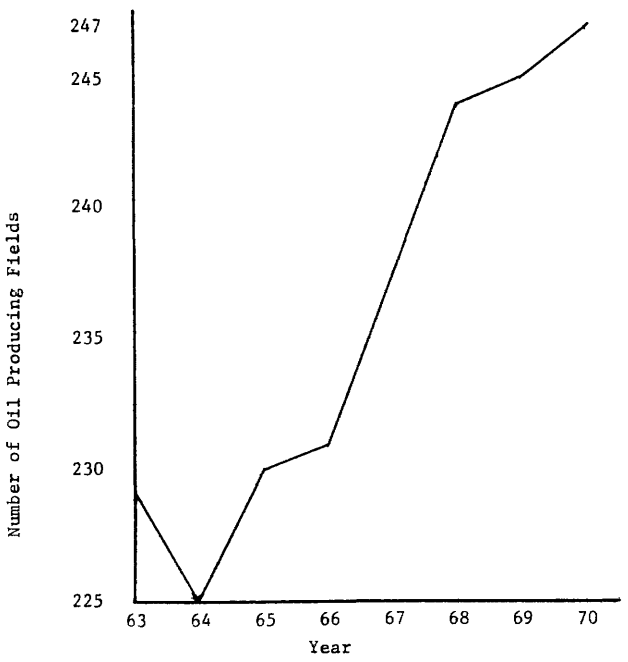


Exhibit B

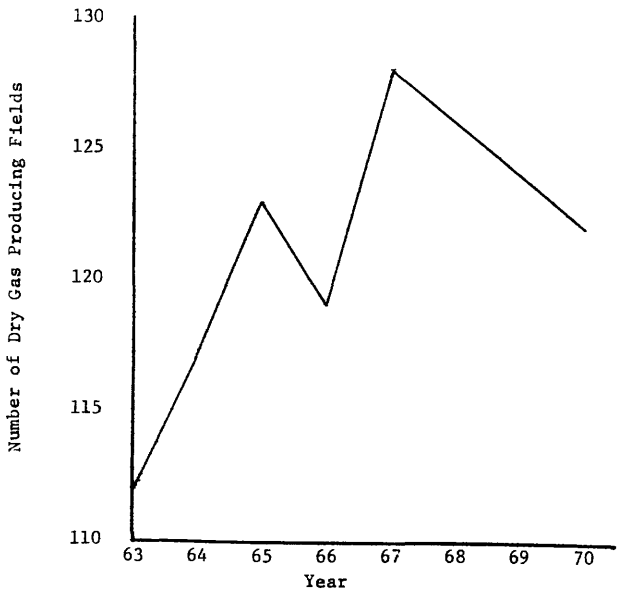


Exhibit C

The Hydrocarbon Property Appraisal System (HAS) described in the remainder of the paper allows the Intercounty Equalization Division to maintain acceptable appraisal quality standards and remain within the present budget limitations.* Moreover, the system's ability to handle a greater volume of current data will allow for a more accurate and flexible value-trending program in succeeding years. The program creates a high level of confidence within the counties that the petroleum property appraisals are well documented and professionally done.

The Board of Equalization is making this system available to counties with the hopes that it will result in

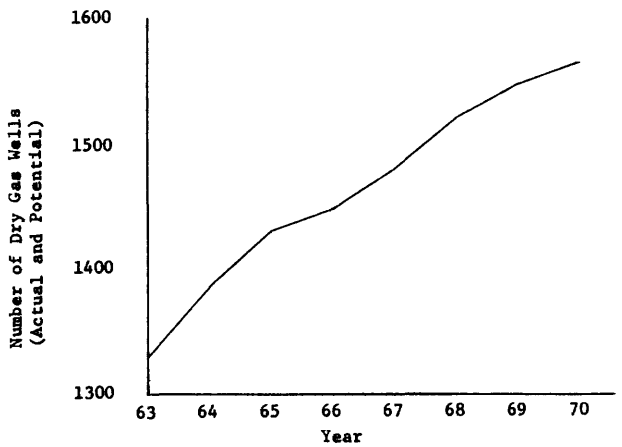


Exhibit D

* An increase in the market value of several counties is expected to occur when HAS is used to value the petroleum mineral rights in the counties. The program reduces the time needed to appraise petroleum mineral rights so the Senior Petroleum and Mining Engineer can devote more time to other tasks (i.e. appraising other types of mineral rights).

improved property tax equalization and greater appraisal standardization for oil and gas producing properties.

The remainder of the paper is devoted to describing the traditional method of hydrocarbon property appraisal, how the traditional method was automated, short comings in the present automated system, and overall system improvements to be made in the future.

MANUAL METHOD

Producing hydrocarbon properties to be appraised in the triennial survey are selected by two methods. First, as previously stated, all hydrocarbon producing properties are appraised in each county where these properties exceed 3 percent of the county assessor's roll. Second, in those counties where the value of such properties is less than 3 percent of the county assessor's roll, the hydrocarbon properties are randomly selected as are all other properties,

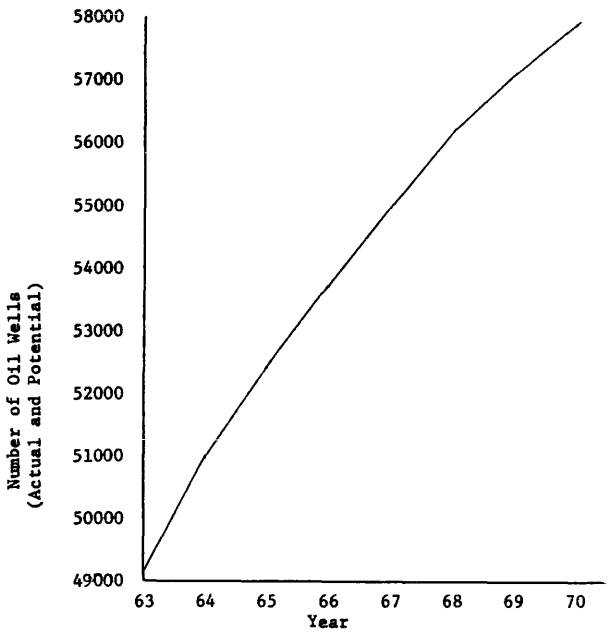


Exhibit E

Once a specific property or field is selected from the assessor's roll, the appraisal of the property follows this step-by-step procedure.

1. Identify property to be appraised.
2. Collect historical production data on subject.
3. Collect historical expense and cost data on subject property.
4. Collect historical capital expenditures on subject.
5. Discuss properties operations with company engineering personnel.
6. Review items 2 through 5 and select the best method (volumetric, material balance, or decline curve extrapolation) for predicting reserves and future production rates.

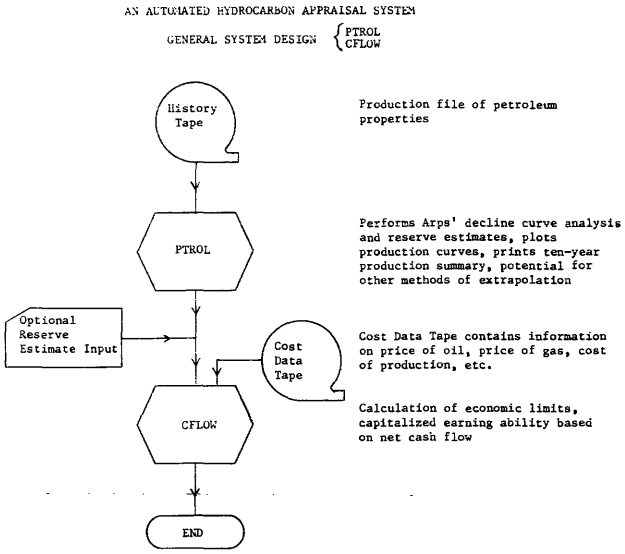


Exhibit F

7. Review items 2 through 6 and select the appropriate operating costs and capital expenditures for the reserves and production rates selected in step 6.
8. Determine future net income by multiplying the estimate future production rate by the price of oil and subtracting the estimated future costs.
9. Capitalize the net income to a present worth indication at the appropriate interest rate to derive a market value indicator. Separate studies are required to identify an appropriate rate of capitalization. (See Appendix A).

AN AUTOMATED SYSTEM OF HYDROCARBON APPRAISAL - SUBSYSTEMS WITH BRIEF DESCRIPTIONS OF GRAPHS AND FUNCTIONS

PETROLEUM PRODUCTIONS GRAPHS, EXTRAPOLATION AND CASH FLOW

HISTORY TAPE	Storage and retrieval of production data
PTROL	Printing of 10-year production history
	Extrapolation by Arps Decline Equation
	A. Graph A B. Graph B C. Graph C
	Graph plotting routine (11)
	A. barrels per day vs. time (semilog) B. Barrels per day per well vs. time (semilog) C. MCF per day vs. time (semilog) D. Gas oil ratio vs. time (semilog) E. Number of wells vs. time (cartesian) F. Number of wells full time producing vs. time (cartesian) G. Percent cut vs. time (cartesian) H. Cumulative gas vs. cumulative oil (log log) I. Monthly oil production vs. cumulative oil production (one semilog and one cartesian) J. Graph of percent cut vs. cumulative oil production (cartesian)
CFLOW	Calculation of economic limits
	Calculation of capitalized earning ability based upon net cash flow
	Comparison of capitalized earning ability value indicator with other value indicators derived from sales

NOTE: Graphs are plotted on simulated semilog, log log and cartesian coordinate paper as noted.

Exhibit G

10. Compare the value indicator derived in step 9 with value indicators derived from sales.
11. Estimate the market value for subject property.

AUTOMATED METHOD

The Hydrocarbon Appraisal System, HAS makes major changes in the above job stream. (See Exhibit F and G). The system can only provide meaningful reserve estimates for those fields that are amenable to ARPS' decline curve analysis.[1] If the appraiser is aware that a specific field is not amenable to decline curve analysis he may, by a control card modification cause the computer to plot a suite of production curves for the field in question. These curves are not as good as an estimate of reserves but they do provide the appraiser information which would otherwise have to be found elsewhere with a resulting large amount of effort expended. These curves are of great importance to the appraiser, who uses them to estimates reserves. Once the reserves are estimated he inputs this information into CFLOW, the cash flow modules.

Although the fields amenable to decline curve analysis contain only about 25 percent of the state's oil and gas reserves they constitute 60 percent of the total number of fields, and therefore 60 percent of the workload. The other 75 percent of the state hydrocarbon reserves may be appraised by use of CFLOW, a subsystem of HAS.

County petroleum production figures, before the advent of PTROL, had been trended manually by estimating reserves and production rates, using over-simplified straight-line extrapolation of the historical production curve manually plotted on semilog paper for each field. The production figures when multiplied by the price of oil would result in the future gross income. The gross income stream thus derived is adjusted to net income using costs collected during the last survey year. Such a method does not respond well to technical innovations, new discoveries, and extensive well repair programs, and therefore, left much to be desired.

The mere extension (by use of a straight-line projection) of past production level into the future until the economic limit of production is reached on a time frame is an expedient method, but in many cases, not viewed by industry, or government as an accurate measurement of future production. One technique generally accepted and used by industry and government is known as the ARPS Decline Curve analysis. Due to human error and time constraints, hand calculation of the curve is impractical in light of the volume and difficulty of calculations required. The practical use of this equation (together with other uses based upon the needs of each user) has been made possible by the advent of automated data processing.[2,3,4,5] The primary objective of PTROL is to plot production curves and use the ARPS Decline Curve analysis in the prediction of future production. The main objective of CFLOW is to extend PTROL by using the predicted production as input for the automation of summary capitalized earning ability that is required to value hydrocarbon producing properties and the printing of a final appraisal report consisting of a summary of value conclusions derived from the entire system.

In order for HAS to be effective, it must have a data base of production history. The Board of Equalization is grateful to the Conservation Committee of California Oil Producers for releasing through the Service Bureau Corporation of Inglewood, California, once a month, a copy of their monthly hydrocarbon production statistics. This tape of production statistics is the heart of the data base as without it none of the system except CFLOW would be able to function. The data base has a capacity of ten years of monthly production data for every field in California. There are approximately 1250 named producing fields in California. The monthly production data consists of oil, gas and water production figures as well as total numbers of wells and total number of fulltime producing wells.

The Board of Equalization originally was able to capture only 3 years of data for the data base. This was not viewed as a serious drawback because the PTROL subsystem which requires the data base will not be needed until next year when four years of data will be available. It is felt that four years at a minimum, of data will suffice for curve plotting purposes. The extrapolation routines in PTROL do not require all ten years of data for accurate fitting. The ten years of data are there primarily for the appraisers benefit so he might have historical information which he can use and override the predicted hydrocarbon production with values of his own before allowing the information into CFLOW.

With the advancements provided by HAS, the job flow is reduced to the following:

1. Identify specific property to be appraised.
2. Collect historical expense or cost data on subject. (Used in CFLOW)
3. Collect historical capital expenditures on subject. (Used in CFLOW)
4. Input items 1 through 3 into HAS system.
5. Discuss properties operations with company engineering personnel using the PTROL graphs and CFLOW report as an information source.
6. If disputes arise or errors are found in the graphs or the report, correct errors and go to Item 4.

The PTROL subsystem extrapolates the data base explained earlier (i.e., the oil production decline curves) using the method developed by ARPS and later tested on California oil fields by Higgins and Lichtenberg. The extrapolated curve is integrated by year to determine the volume of oil produced for each year in the future.

The subsystem CFLOW takes as input either manual extrapolations or machine fitting from PTROL. The volume of oil is multiplied by the price of hydrocarbon products to give the gross income per year. The future value of the hydrocarbon products are balanced against the future expenses and costs required to produce them.

The economic life of the field is the number of years required for the expenses to overcome the income earned from production hydrocarbon substances. Given the economic life, the net income is capitalized at five different fixed rates as well as by one rate specified at input. (See Appendix A) Various other value indicators are calculated and printed. These indicators are used to check the value of the property derived by machine.

If errors arise as mentioned in Item 6, above, machine time is not required to extrapolate the future production again. If the appraiser feels the extrapolations are correct he may use these values for input to CFLOW and change only the cost and expense value to correct the errors in Item 6.

The inspiration for CFLOW came after perusing a copy of the Kern County, California appraisal program. Theirs is a modified version of a still older Standard Oil of California Program. After discussions with the Intercounty Equalization Senior Petroleum and Mining Appraisal Engineer we came upon the calculations and format which we thought would express the value of the hydrocarbon mineral rights best. It is embodied in CFLOW.

FUTURE IMPROVEMENTS

Future improvements in the hydrocarbon property appraisal system (HAS) would include a cost history subsystem for cost and expense data, a new system flow when aforementioned subsystem is completed and more methods of predicting future production (i.e. material balance, volumetric, etc.).

A problem encountered during the implementation phase of HAS was the staggering amount of detailed cost data required for the system to function. A manual system was developed to reduce the cost data to a useable form for the machine. This manual process was very slow and due to the turnover of clerks not very accurate. The solution to the manual system of cost data reduction would be to design and implement a cost history system to reduce the data for the appraiser.

A cost history subsystem would consist of a file of cost and expense data and file maintenance system for adding and deleting data. The file for each hydrocarbon producing field would be of sufficient length so that as much as ten years of cost and expense factors could be stored. These could be printed so that the appraiser could review and correct them if necessary before using the factors as input to HAS.

Taking cost history subsystem and integrating it into (HAS) would result in a far more sophisticated appraisal system. Once triggered by the Petroleum Engineer, the automated cost factors produced by the data reduction program would be directly input into HAS. PTROL would then process as it does now.

This improvement would not only allow the Petroleum Engineer to modify the appraisal data and reenter it into the process at any point but would also allow the system to completely appraise a property without appraiser intervention. This would shave the budget even more, because a significant amount of cost is contributed by the hand processing of cost data by the appraiser.

Another improvement would be to allow PTROL more methods of predicting future production. This would require more research into the state of the art of hydrocarbon production prediction. During the research required for PTROL and CFLOW design several sources were found that indicated that methods dealing with volumetric and material balance were already developed. It would not take too much effort to acquire copies of these sources so that evaluation of their methods could be made in the light of information present in the data bases of this hydrocarbon substance appraisal system.

SUMMARY

HAS is an example of a cooperative effort between a system analyst and a technical engineer. The analyst knew nothing about petroleum engineering and the engineer knew nothing about systems. The system was designed so that if any new technical innovations were published they would be incorporated into HAS with a minimum of down time. This flexibility of the system was due to the report established between the analyst and the engineer.

In conclusion the following reasons summarize the arguments and suggestions for the implementation of PTROL and CFLOW.

1. More accurate appraisal of petroleum properties in California.
2. Better standardization and equalization in petroleum counties throughout the state.
3. Greater accuracy and flexibility in trending oil and gas property values.
4. Quality appraisals at a substantial savings to the state.

ACKNOWLEDGMENTS

I would like to thank the following people: Harold Bertholf, Senior Petroleum and Mining Engineer, California State Board of Equalization, for his valuable advice on appraisal of petroleum properties. Abram F. Goldman, Chief of the Intercounty Equalization Division, California State Board of Equalization, for his support of the project in the face of adversity.

REFERENCES ON PETROLEUM FIELD DECLINE CHARACTERISTICS

1. Arps, J. J., *Analysis of Decline Curves*, TP 1758, Petroleum Technology, September 1944.
2. Higgins, R. V., Lechtenberg, H. J., *Production Decline Curves Using Data From California Oilfields*, U.S. Department of the Interior, Bureau of Mines Report RI 7547, 1971.

3. *How to Predict Oil-Field Performance*, Oil and Gas Journal, September 14, 1970.
4. *Merits of Decline Equations on Production History of 90 Reservoirs*, Paper No. SPE 2450, Society of Petroleum Engineers of AIME, 1969.
5. Shea, G. V., Higgins, R. V., Lechtenberg, H. J., *Decline and Forecast Studies Based on Performances of Selected California Oilfields*, Paper No. SPE 833, Society of Petroleum Engineers of AIME, 1964.

ADDITIONAL REFERENCES ON PETROLEUM FIELD DECLINE CHARACTERISTICS

McCray, A. W., Comer, A. G., *Statistical Basis for Choice Among Hyperbolic Decline Curves and Computer Application in Calculation Confidence Limits of Reserve Prediction*, Paper No. SPE 1930, Society of Petroleum Engineers of AIME, 1967.

Locke, C. D., Schrider, L. A., *A Unique Approach to Oil Production Decline Curves Analysis with Applications*, Paper No. SPE 2224, Society of Petroleum Engineers of AIME, 1968.

APPENDIX A

Capitalization is the process whereby present worth is calculated. The present worth mentioned here is concentrated at midyear. Present worth (P.W.) is defined by the following equation:

$$\sum_{j=1}^{e} \frac{n(j)}{(1+i)^{j-1/2}}$$

where

e—the economic life of the property
$n(j)$ the net profit of the jth year
i—the interest rate desired

The value that the above process provides is the amount of money one must pay today for a value described in tomorrow's dollars.

Example: If a property made a net profit of \$5 per year for 5 years in an economy that was expanding at 8 percent a year, how much would that property be worth in today's dollars?

YEAR	NET PROFIT	AMOUNT
1	$\frac{5.00}{(1.08)^{0.5}}$	\$ 4.81
2	$\frac{5.00}{(1.08)^{1.5}}$	4.45
3	$\frac{5.00}{(1.08)^{2.5}}$	4.12
4	$\frac{5.00}{(1.08)^{3.5}}$	3.82
5	$\frac{5.00}{(1.08)^{4.5}}$	3.54
		\$20.74

Therefore, in an expanding economy that is expanding at 8 percent the twenty-five dollars in the future will be worth \$20.72 in today's dollars.

What is different about tactical military operational programs

by GEORGE G. CHAPIN

Litton Systems, Inc.
Van Nuys, California

INTRODUCTION

The increasing complexity of tactical military operations has required a corresponding increase in information processing using systems built around general purpose, stored program digital computers. The operational programs for these systems have attracted increased attention because cost overruns and missed scheduled deliveries have been common. The cost referred to is the one-time total programming cost (personnel, equipment, facilities, etc.) per instruction for the definition, design, production, test, installation and documentation of the program. Typically, the initial development takes from three to six years and often includes at least two iterations before a useful product is obtained, while the programming cost runs from $60 to $100 an instruction.

In the author's opinion, these difficulties are partly caused by the different data processing problems encountered in tactical vs. other systems. Clearly the differences are not absolute, for there are degrees of similarity between tactical and certain non-tactical systems as well as differences between tactical systems themselves.

In order to discuss these differences, it will first be necessary to define tactical systems in terms of equipment, operating personnel and problems being solved. This will provide insight into the kinds of data interchange between the computer and the rest of the system, into the "response time" requirements, and into the requirements for continuous 24-hour per day operation. This in turn will lead to a discussion of the nature of the data processing, the parallelism of processing to implement system tasks and the resulting high rate of communication between the major elements of the program.

PURPOSE AND NATURE OF TACTICAL MILITARY SYSTEMS

Tactical military systems are generally housed in ships, aircraft, transportable shelters or vehicles. Their purpose is to support operating military personnel by:

- coordinating the collection of data from own-site sources and from external sources and systems;
- correlating the data to obtain a clear picture of the tactical situation;
- processing the data required for the decision-making function; and
- communicating the decisions and actions to weapons, other users, or other systems.

The objective of the man/machine relationship in the system is to remove from the operator, to the maximum practicable extent, tiring and repetitive operations in order to concentrate his effort in areas requiring decisions based on judgment and experience. Figure 1 illustrates the relationship between the man/machine elements and the operational tasks which the system must perform.

Many tactical systems are for the purpose of establishing a clear picture of fixed and moving objects—in the air, on the surface of the ocean or under the surface. Tracks are developed and, in conjunction with amplifying information, are used for decisions related to air traffic control, the assignment and control of weapons, the control of rendezvous and strike missions, etc. Other systems are for the purpose of deriving and processing information for artillery fire planning and coordination or for the control and switching of communications data. Still other systems combine several or all of the above purposes.

There are other military and commercial systems which may or may not be called "tactical" but which perform similar functions. Examples include fixed site air defense, air traffic control, process control and communications switching systems, as well as the many applications of inter-active, on-line terminals. Further, many tactical systems now solve "business" data processing problems in such applications as logistics and intelligence. Thus there is no clear line of demarcation between the problems solved by tactical military vs. other systems, and hence between their computer programs.

Even though we cannot define tactical systems exactly, we shall consider programming for the most common type of tactical system—a "real time", man/machine system which develops "tracks" from sensor and data link inputs for ultimate assignment to and control of weapons. Such programs must cope with inaccurate and/or false data inputs, short response times to a large number of essen-

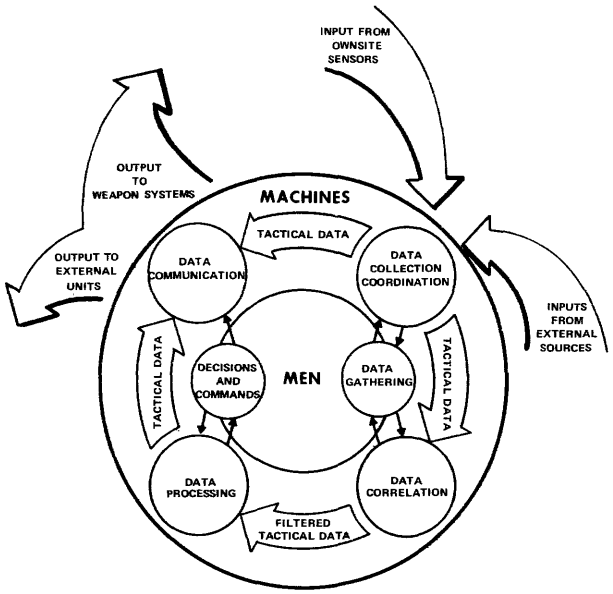


Figure 1—Man/Machine relationship in a tactical system.

tially unpredictable inputs, operator errors and equipment failures, in such a way as to handle peak processing loads while operating reliably 24 hours per day. The typical characteristics of such a program will be discussed and compared with those of non-tactical programs.

DESCRIPTION OF TACTICAL MILITARY SYSTEMS EQUIPMENT AND PERSONNEL

General purpose digital computers

Computers now in use range from mini-computer capability to large capability. Most of the less capable computers are used either in rather specialized applications (such as a small digital communications switching system) or to perform pre-processing or special functions in a larger system. Many systems have multiple computers (unit computers), multiprocessors, or combinations for the basic central processing functions, augmented in some cases by specialized processors. Typical systems with interconnected computers have from two to six processors, from 32K to 192K of main memory with cycle times from 1μsec. to 8μsec. and from 8 to 48 input/output channels, employing two cables per channel or bussed data techniques.

Sensors and interfacing equipment

The most common sensors are radar, radar beacons (Identify Friend or Foe) and sonars. In addition there are a variety of sensors employed for electronic countermeasures and for determining platform motion and position. Other sensors are remote in aircraft, helos, satellites, sonobuoys, etc. All sensors measure an analog quantity, and the corresponding electrical signal is usually analog. To be used for processing by digital computers, the analog signed must be converted to digital within the sensors or by use of special interfacing devices.

Examples of such interfacing devices are processors which receive radar or radar beacon video and automatically extract target position information in digital form. Resulting target reports, whether from radar or radar beacon, are inaccurate and often false. The computer program must be designed to cope with these false and/or inaccurate reports, which are termed "clutter" from radar and "fruit" or "garbles" from IFF.

Some sensors are controlled in a simple manner, requiring little control or processing by the computer. In effect, they just run. The computer receives only simple information such as sensor azimuth or own-site speed and heading. Some more modern sensors are much more complex and require substantial processing with severe time constraints, for example, to direct the steering of the sensor beam hundreds of times per second.

In interfacing sensors with the computer, there are usually trade-offs between use of special processing logic vs. the general purpose computer. Examples include radar and radar beacon video processors and sensor beam steering processing.

Since the digital inputs received from sensors are derived from analog information, the accuracy (and hence precision) of the inputs is limited by the physical process of conversion. Typically, the precision of a single quantity is 12 to 14 bits (for example, a position coordinate).

Operator-manned equipment

A variety of operator-manned equipment are used for input and output of data to/from the computer. The most common type is a display including a Plan Position Indicator (PPI) cathode ray tube augmented by readouts and data entry devices. Other equipment include alphanumeric cathode ray tube displays, special data readout devices using film projection techniques, light-emitting diode devices, keyboards, etc. Still other devices are made up of panels of lights, indicators, and entry buttons. Often sensor data is presented in conjunction with computer derived data.

The PPI display console utilizes a cathode ray tube for display of radar, radar beacon (IFF) or sonar data, augmented by data readouts, keyboards, and data entry switches. Most tactical systems have from three to thirty displays of this type. Sensor data, such as radar video, are distributed to each of the displays. Usually the data from several sensors are available, and the display operator selects the one required for his function. Computer-generated data are displayed as special symbology, alphanumeric data or lines.

There is considerable programming required to service displays. Again, there are trade-offs between programming vs. special logic built into the displays. Because of the importance of displays as the prime man/machine

interface, a later section discusses display processing in some detail.

Operators

Military personnel are an integral part of all tactical systems. Systems must be designed to assist real operators, many of whom are enlisted men with limited training. System designers and programmers too often overlook the fact that the tactical system is there to assist the operators and is not an end in itself. The proper involvement of the ultimate user in system design and programming is one of the more important ingredients in the development of tactical systems.

Communications equipment

A variety of communications terminals may be directly connected with the computer to permit site-to-site communications of digital information with other fixed or moving platforms. Typical speeds of transmission vary from 75 to 4800 bits per second and, through use of multiplexing techniques, much higher speeds. In addition, digital data from remote sites such as satellites enter (and leave) through terminals. Also computers are interconnected with other equipment in some systems to assist operators in circuit establishment, equipment turn on and tuning, performance monitoring, and message processing.

Computer peripheral equipments

Standard (but ruggedized) peripherals include magnetic tapes, disk files, card readers and punches, paper tapes, etc. Normally such equipments (with the possible exception of disk files) are not used in the mainstream of operations. Rather they are used for program entry, system restart, simulated entry of data for training and system debugging, recording of extracted data and recording of critical data.

Weapons systems and interfacing equipment

Such systems include surface-to-air missiles, torpedoes, anti-submarine rockets, and (indirectly) interceptor and other aircraft. Since outputs from the computer are used to control some physical process, the digital data must ultimately be converted to analog. This limits the required outputs to 12 to 14 bits per quantity in most cases.

Since most weapons systems are complex, considerable processing is required for control. Special purpose interfacing devices are often used to assist in the processing. Analog computers are also used in some cases. When digital techniques are used for control of guns or missiles, critical timing requirements are imposed on the program to accommodate a many-times-per-second feedback loop involving data sampling, computation and orders.

EFFECT OF MULTIPLE INSTALLATIONS

The number of installations and the similarity of installations significantly affect both the development and maintenance of tactical system programs. Some systems, particularly airborne, have very similar if not identical installations on many aircraft. Other systems, such as shipborne, tend to be one of very few of a kind, although employing very similar data processing hardware. Usually the sensors, weapons and ship mission cause the major changes. Still other systems have multiple installations which are generally similar but have different radar locations, geographical constraints, and perhaps weapons.

Even identical systems have a way of changing with time, causing more than one program to be in use at one time. A careful assessment must be made early in development of the number of similar but different programs and the method of program maintenance, for quite different program design concepts may be employed based on the results. For example, a system for thirty "identical" aircraft can emphasize highly efficient programming. Conversely, the ship requirement for many similar but different programs has lead to a "modular" program design approach which is tailored to ease of constructing new programs but at the expense of computer storage and execution time.

CHARACTERISTICS OF COMPUTER INPUT/OUTPUT DATA TRANSFER

The input/output problem consists of a concurrent transfer of data with up to twenty equipment groups consisting of up to fifty or more equipment items. The word concurrent means that in a short period of time (say two hundred microseconds) data may be transferred between the computer and four or more equipment groups. Typical data transfer rates total 10,000 to 300,000 words per second.

The programming associated with set up and control of input/output is highly computer dependent. Well designed computers make the I/O problem quite easy, even though there are many peripheral devices (including other computers) and considerable data transfer. Most systems employ buffered transmission on multiple I/O channels or data busses with timing controlled by the external equipment once the transmission is started. A single computer instruction initiates either an input or an output buffer mode. Once established, buffer transmissions employ independent access to memory, and the entire buffering operation proceeds to completion with no additional program instruction executions. As a result, the buffer mode of data exchange provides an input/output operating asynchronously with the main computer program so that the computer can continue execution of program instructions in the normal sequence. In most cases it is desirable for the program to be informed of the completion of a particular buffer transmission. Some

computers have an internal interrupt which is generated within the computer when the buffer mode terminates. The internal interrupt provides the signal to the Executive program indicating completion of the I/O buffer, thus eliminating the programming problem of monitoring the status of the I/O transfer.

On an average, the computer should spend only a small percentage of its time in getting data in and out of the computer. Occasional peak input/output loads will occur, lasting for a period of a few milliseconds, during which the time used for input/output increases significantly. The system should be designed, however, so that even during peak loads, it is not possible to miss a transmission of data because of equipment overload. The program processing the I/O should be more flexible in that under peak load conditions, certain outputs (such as display) can be temporarily deferred and highly repetitive inputs can temporarily be ignored.

In general, many items of data are received from or sent to the same equipment. All data words must either explicitly contain information identifying their contents or have positional (implied) encoding. The data transferred (other than control information) are of two basic types. One type is normally sent only once. Care must be taken to have positive acknowledgment of such data so that they can be sent again if not received properly. For example, entry of information from a keyset or keyboard is done only once. Such data are normally retained by the system until a new entry is made. Another example is a request for computation made just once by an operator and acknowledged by displaying the results of the computation.

The second type of data is somewhat "cyclic" in nature, that is, similar data are sent on a more or less periodic basis. Data derived from search radars, for example, essentially repeat once each radar scan time. Data on tracks received from a data link are repeated as the track position changes.

Since most tactical systems data are cyclic, the ratio of data received by the system to data stored by the system is quite high compared to commercial systems. Therefore, it is usually possible to store most, if not all, data in the computer. As a minimum all rapidly changing data should be stored in main memory because of the many accesses to it that are required.

EXAMPLES OF COMPUTER PROCESSING

In order to provide a "feel" for the type of processing required, several examples are discussed in this section.

Display processing

The type of display under consideration is the PPI-type device discussed previously, and the system functions will be tracking and the assignment and control of weapons.

The computer generated data to be displayed change frequently, for example, the position of an air track. In addition, the superimposed sensor data usually must fade prior to its next sequential presentation or scan in one to ten or more seconds. This combination of computer and sensor derived data requires, for visual clarity, a display phosphor which rapidly decays and must be frequently regenerated or refreshed.

Typical refresh rates are 10 to 50 times per second if the display is to be "flicker free." In most systems the computer does the refreshing, that is, it retransmits the data to the display at this 10 to 50 times per second rate. Since the computer must continuously have the rapidly changing display data in high speed memory anyway and since at least some output data changes each refresh period, it is less expensive to use output time than to provide memory in each display or on some intermediate storage device such as a magnetic drum.

There must be an effective way for the operator to request information from the system and to enter data. One method of requesting information is through use of a "ball tab" or cursor on the face of the display. The ball tab symbol is sent from the computer and displayed. As the operator moves his track ball or joy stick, coordinate changes (Δx and Δy) are accumulated in a display register. Periodically the Δx and Δy are sent to the computer. The computer adds the Δx and Δy to the previous x and y positions which, when sent to the display, caused the ball tab symbol to be positioned. The next output to the display sends the updated coordinates, $x+\Delta x$ and $y+\Delta y$, which causes the symbol to change its position on the display. This entire process must be done rapidly enough so that the symbol appears to be moving smoothly across the face of the display and can go from one side to the other in a few seconds. This can be done if each refresh of the display has a new value for the ball tab position, which requires the Δx and Δy values to be received by the computer once each refresh period. In most systems, the computer "interrogates" each display once per refresh period, obtaining the Δx *and* Δy values or any other data entry made at the display since the last interrogation.

The display console operator enters data or requests information using such devices as keyboards, quick action buttons, number entry dials, etc. Some of these buttons have the same meaning for all functional uses of the console. Other buttons have different meanings, depending upon the functional use of the console (determined by a switch position). The input to the computer is a discrete code for any button action which has occurred.

The operator action must be detected by the computer, usually through decoding the response to the interrogation described above. Another method of entry uses an interrupt to the computer to indicate some operator action. In any event, the program must decode the request or action and usually perform some processing to provide an output back to the operator. The decoding can be quite complex, for the same digital input can have different meanings depending on console function and on such ancillary factors as range selection, offset, etc. From 200 to 1000 different possibilities are typical. In order to maximize operator performance, the response should be "instanta-

neous" insofar as the operator is concerned, that is, in several hundred milliseconds.

Frequently the program must search data stores because of the operator action. Such searches normally look for values closest and within limits to a reference rather than for equality; e.g., the operator may wish to "hook" a track in order to take some action or obtain some amplifying information. To do this, he moves his ball tab over the track symbol and depresses a button, say HOOK. The program uses the ball tab coordinates as a reference and searches track stores for the track whose coordinates are closest to the ball tab and within some maximum search area. After finding the track, the program prepares the appropriate outputs such as a HOOK symbol and amplifying data.

Normally a group of displays shares a single computer output channel or data bus. Logic is built into each display to permit acceptance of data based on console address, category selected, etc., and to extract needed digital data such as track position. (Analog symbol information is usually generated by a central device.) The information must be organized in a manner understandable to the displays. For reasons of economy, the word formats are usually designed to save display hardware and reduce the number of words transmitted, rather than for the convenience of the computer program. Output data are organized by the program into one or more blocks for buffered data transmission. Once started, the transmission proceeds without attention from the program, usually at a rate determined by the display system.

The organization of data in the output buffers requires considerable logical processing to change it. For example, an output word might contain X, X dot, and console address data. Changing the X or X dot requires masking and perhaps shifting, assuming the basic computation causing the change has been made from a separate track store which is organized for computational convenience. Also, there are problems of packing the buffers, keeping track of data locations, and preventing update at the wrong time.

A typical display system might require 2,000 words of output at a frequency of 20 times per second. If 2μsec. of memory time are required per word, then the total memory time used is $2{,}000\times20\times2=80{,}000\mu$sec. per sec. or 8 percent. If the computer does not have independent I/O, an equivalent amount of processor time also may be required.

Track processing

Let the system involve a radar processor and radar beacon (IFF) processor as described previously. In the first case, assume the system is to have the ability to automatically initiate, update and maintain the tracks, with manual operator inputs also allowed to inhibit portions of the automatic operation and to correct or assist the automatic operation.

The radar processor inputs coordinates of targets, usually in polar coordinates. The IFF processor inputs polar coordinates plus IFF codes. In addition, radar azimuth data must be input. Input buffers must be continuously available to receive both radar and IFF reports. This requires a double buffering scheme for each device. As discussed previously, the inputs may be real or false and when real, are inaccurate.

The problem then is to reject the false reports while using real reports to update previously developed tracks or to initiate "new" tracks. The processing involves extensive searching to determine if inputs are within computed (and variable) "bins" around predicted track positions (where the program thinks an established track should be). The processing algorithms are complex exercises in logic and involve much "housekeeping." False inputs must be retained until proven false. Identity codes must be correlated for friendlies with means of handling inputs which are satisfactory position-wise but have conflicting identity codes. Radar and IFF data on the same track must be correlated. Established tracks must be continued even though no inputs are received. Operators must be allowed to make inputs from displays and inhibit automatic tracking on a selective basis. Crossing, merging and splitting tracks must be handled for aircraft with substantially different speeds.

The basic anomaly in tracking is caused by the inaccurate nature of the input data. Is the inaccuracy just an error in position or is it caused by aircraft turning? This basic problem leads to additional programming, again "logical" in nature, to follow aircraft in turns and may include the establishment of "possible tracks" until later data are available.

Once correlations between existing tracks and inputs have been performed, track position and velocity must be updated to obtain the best estimate for the next radar scan. These updates utilize typical filter theory algorithms which involve numerous but simple arithmetic operations.

A much simpler program is required if all tracks are initiated by the operators and then maintained automatically. More operator actions are required and hence the system capacity is reduced, but so is the size of the program. Also, there are other combinations of initiation/extrapolation which are used in various systems. In all cases, however, the processing is quite similar, differing more in amount and complexity than in type. In addition, track information arriving over digital data links must be correlated with own-site data in order to obtain a clear picture, since more than one site often sees the same aircraft.

Concurrent with the tracking, information must be continuously presented to operators on their displays. Operators must attempt to identify non-IFF tracks through a variety of means, which could include flight plan correlation.

Weapons assignment and control

The picture of the air situation is used by other operators who determine what action, if any, is required. Based

on weapon status data which must be continuously available, the program assists the operators in various ways by making trial engagement computations (can an interceptor make the intercept with existing fuel and weapons?) and threat assessments. If a decision is made to engage a track with an interceptor or missile, the program must prepare orders to control the weapon. These orders are then dispatched over data link (or by voice) to an interceptor or through interface electronics to the missile system. There is substantial variation in the processing required depending on the type of assistance provided the operator, on the weapon being controlled and on the type of control system. In general, complex but infrequent computations are required. In the case of some modern missile systems, the computer must frequently perform the complex guidance and control computations, involving a computation/order/receipt-of-new-data feedback loop between the computer and the missile, repeated many times a second.

Other processing jobs include assessing the results of engagements and maintaining the status of weapons. Finally, information on actions taken must be transmitted to other sites.

PROGRAM RELIABILITY

Most tactical systems have a 24 hour per day operating requirement with a very high probability of operation. The computer program clearly must support this reliable operation, not only by being inherently "error free" but also by coping with problems occurring elsewhere in the system.

High reliability is built into the program in a variety of ways. First, the program must not fail because of improper operator actions—it must be "idiot proof." This requires all program paths used to process operator actions to be closed, with feedback to the operator of illegal actions. Second, the program must reject bad data entering the computer from sensors, data link, etc. This requires reasonableness checks, parity checking, error detection/correction decoding, etc., as well as the logically complex processing required to reject bad sensor derived data. Third, the program must operate even though some system equipment has failed. The system design dictates whether useful system operation is possible with equipment failure by the amount of equipment redundancy, switching capability, etc. Fourth, degraded modes of the program itself must be available when less than the needed complement of computer hardware is available.

The overall control of the program and data processing system is normally accomplished by an operator located at a special display position with remote computer controls. The operator typically may select program modifications, eliminate program functions, restart the program, etc. Also he may order a reconfiguration of the system. Considerable programming is involved to provide the operator with sufficient information so that he can initiate the best action. The major difficulties are in the Executive (the Loader and Memory Allocation scheme) and in saving critical data (i.e., restart with data).

Building not only a reliable program but also a program that supports reliable system operation is the most difficult area in tactical systems programming. Typically 10-30 percent of the code is used for these purposes, even though less has been accomplished in this area than elsewhere in the tactical program. Part of the problem is that features must be incorporated in the computer and other equipment, which in turn requires rather complete system/program design before equipment is designed.

PROGRAM ORGANIZATION

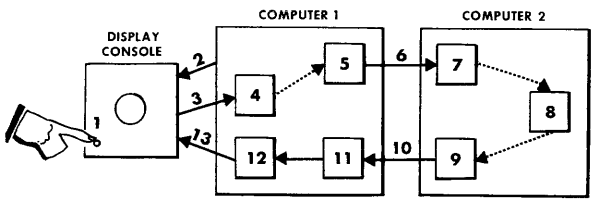
There have been many techniques implemented to organize tactical operational programs. The implementation technique is influenced by several of the factors previously discussed, such as:

- The number of similar but different versions of the program caused by different missions, equipment configurations, and other site variables.
- The projected changes in the data processing equipment during the life of an installation.
- The projected changes in sensors and weapons during the life of an installation.
- The requirements imposed by short response times concurrent with peak input loads occurring in an unpredictable manner.
- The projected problems of producing and testing all of the programs as a result of the above factors, while retaining high program reliability.

The size of tactical programs varies considerably but is typically 30,000 to 150,000 instructions (plus data). Because of the size, clearly the program must be organized into "subprograms" which in turn must be organized into one or more levels of "subroutines." The problem first is to establish the boundaries of each subprogram and the method of communication between subprograms; then to establish similar rules for the communications between subroutines (at all levels) within the subprogram. It is then possible to have one or more persons developing subroutines in parallel and, if the interfaces are adequately defined, to perform reasonable testing of each subprogram prior to integrating the subprograms into the overall program. Also, if the subprograms are properly defined, it is possible to accommodate changes in sensors, weapons, and even displays by modifying only one, or at most only a few, subprograms. Further it is possible to produce similar but different programs using some common subprograms plus new subprograms and/or augmented existing subprograms. Program reliability is theoretically enhanced because the smaller number of subprograms get more operational use and can therefore be modular programming" (one of several uses of this over-used term). A typical tactical program may have from ten to thirty subprograms, one of which is the Executive. The intersubprogram communication technique

involves passing messages (under control of the Executive) between modules. Messages include both control information and data. If a multiple computer system is employed, then messages destined for a subprogram in another computer must go through an intercomputer transfer.

Queues must be established to hold the messages until they can be processed. Some programs have one input and one output queue per subprogram, sometimes augmented by a priority designator to determine the precedence of message processing. Other programs have one input and one output queue per processor with priority designators; still others have one priority and one nonpriority queue for input and for output per processor, with a first-in/first-out approach within a given priority. Clearly the problem of designing queues (including handling of overflow) is a difficult problem, usually resulting



1 – PRESS ACTION BUTTON
2 – COMPUTER INTERROGATES
3 – ACTION TRANSMITTED
4 – DECODE ACTION
5 – PACK INTERCOMPUTER MESSAGE
6 – INTERCOMPUTER TRANSFER
7 – DECODE MESSAGE
8 – PERFORM COMPUTATION
9 – PACK INTERCOMPUTER MESSAGE
10 – INTERCOMPUTER TRANSFER
11 – DECODE MESSAGE
12 – PREPARE DISPLAY BUFFER
13 – TRANSFER TO DISPLAY

Figure 2—Events in processing a display request in a two computer system

in queues large enough to handle peak load conditions at the expense of computer memory.

In a typical tactical program, there are 500 to 2000 intermodule messages processed per second, with 200 to 400 messages waiting to be processed at one time.

The method of organization described above requires a number of actions to be accomplished to complete a system task. As an example, consider a two computer system as shown in Figure 2. If a request for computation is initiated which must be processed by the computer not servicing the display, a sequence of 13 events must occur. In addition to these events, the Executive must be utilized between 7 and 11 times (depending on the design).

In Figures 3 and 4 more specific examples are given showing the events required to enter a NEW TRACK and to compute a TRIAL INTERCEPT. Here the individual subprograms are named and the action of each subprogram (including the Executive) is described.

In all of these examples, the time from initial operator action to receipt of reply is from 150-400 milliseconds (in the specific systems used for these examples).

Initial Conditions:

- Two processors with access to common memory
- Processor 1 handles Display; Processor 2 handles Intercept Computations
- Track symbol has been "HOOKED"
- BALL TAB symbol is over interceptor symbol
- Display Operator depresses TRIAL INTERCEPT button.

Sequence of Events:

1. EXEC 1 - initiates display interrogation and receives reply
2. EXEC 1 - schedules operation of display (DS) to process results of interrogation
3. DS - decodes reply; prepares message for further display processing
4. EXEC 1 - schedules DS
5. DS - determines that Intercept Control Program (IC) is responsible; searches and finds track associated with ball tab; prepares message for IC; prepares message for Exec 1
6. EXEC 1 - notes that EXEC 2 is required; prepares message for EXEC 2
7. EXEC 2 - receives message; determines need for IC
8. IC - decodes message; extracts information on hooked target from stores; starts computation; returns to EXEC (because completion would exceed time away from EXEC criteria).
9. EXEC 2 - schedules IC
10. IC - completes computation; prepares message for DS
11. EXEC 2 - notes that EXEC 1 is required; prepares message for EXEC 1
12. EXEC 1 - receives message; determines need for DS
13. DS - decodes message from IC; prepares output to (several) output buffers
14. EXEC 1 - initiates output buffer(s)

Final Result:

- Results of trial intercept displayed with amplifying information on readouts

Figure 3—Typical events in processing a NEW TRACK INPUT

If the time required in the Figure 4 example is 200 milliseconds and if the program processes 100 messages per processor in this time, then the average time per message is 15 milliseconds and about 7 percent of the message traffic during this time is related to processing this high

Initial Conditions:

- One processor is in operation
- Display operator depresses ENTER NEW TRACK button

Sequence of Events:

1. EXEC - initiates display interrogation and receives input reply
2. EXEC - schedules operation of Display (DS)
3. DS - decodes reply; prepares message for further display processing
4. EXEC - Schedules DS
5. DS - determines that Local Tracking (LT) is responsible for processing; prepares message for LT
6. EXEC - Schedules LT
7. LT - enters position data, zero velocity, unknown category, new track "firmness, etc. into track stores; prepares message for DS
8. EXEC - schedules DS
9. DS - prepares output message for NEW TRACK symbol and adds to output buffer
10. EXEC - initiates output buffer

Final Result:

- New track symbol displayed at console

Figure 4—Typical events in processing TRIAL INTERCEPT request in a two-processor system

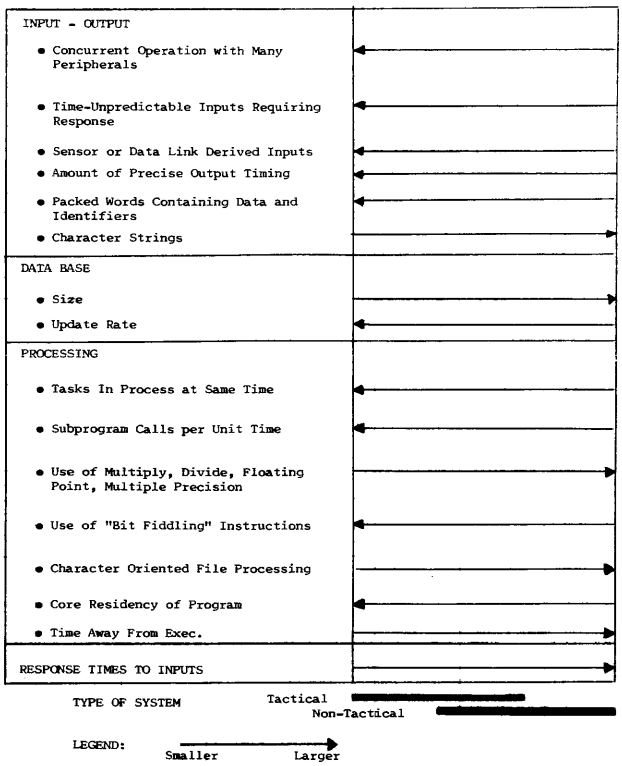


Figure 5—Relative characterists of tactical vs non-tactical data processing systems

priority request (assuming two processors). Since the typical average wait time for processing is 50 milliseconds, there are many low priority messages which can (and do) wait for relatively long periods before being processed. It is this concept, properly implemented so that all tasks are accomplished, which is sometimes referred to as "load smoothing."

CHARACTERISTICS OF PROCESSING AND DATA BASE

This section summarizes the characteristics of the inputs, the data base and the types of processing required to generate required outputs. In Figure 5 a comparison is made of certain of these characteristics for tactical vs. non-tactical systems.

Typical inputs/outputs

- Sensor inputs are derived from analog information which is converted into digital; hence, precision is limited for each item of data; the inverse is true for output data which controls some physical process.
- Many inputs require a response to be output in a short time period; many of these inputs are unpredictable time-wise.
- Concurrent operation with many peripherals is normal; data rates are high.
- Many inputs are "cyclic" in that the data are replaced by new data in a roughly fixed cycle.
- Inputs/outputs are usually "packed" with variable length fields and identifying codes; character strings are uncommon.
- "False" inputs are common from some types of sensors and radio data links.
- Many high data rate outputs require precise timing, for example, display buffering, sensor control and weapons control.
- Real time clock inputs are essential for control.

Typical data base

- Contains mostly variable length fields representing physical quantities, coded representations of status, memory addreses, etc.; character strings are uncommon.
- Size is relatively small because much input data replace existing data. Rapidly changing data are normally in high-speed memory.
- Often tables are packed in that each word contains multiple fields of variable length data items.

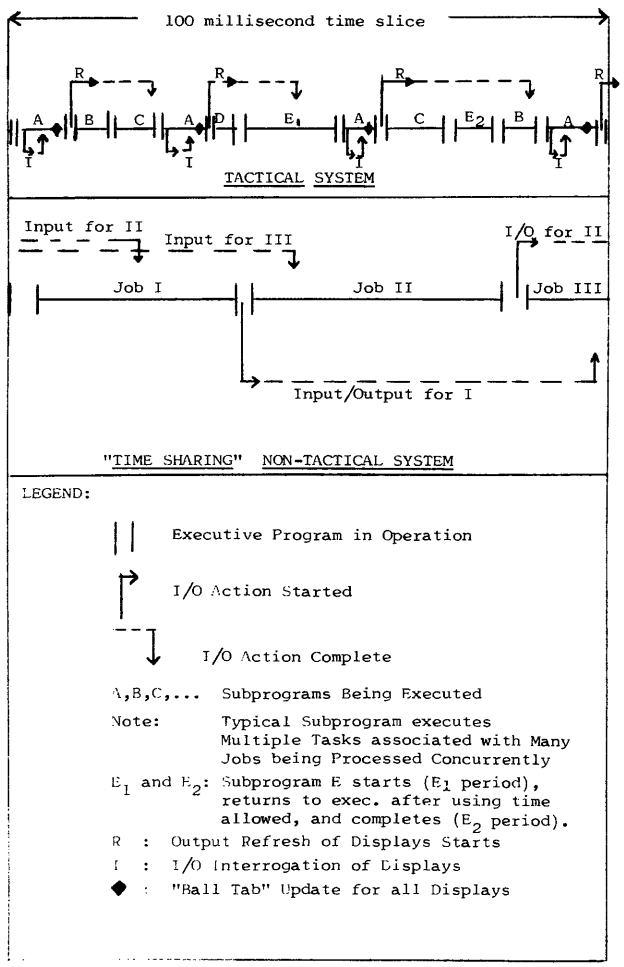


Figure 6—Typical timing sequences for tactical and non-tactical systems

Typical processing

- Usually is highly "parallel" in that many jobs (say 10-50) are being worked on in a short time period (say 100 milliseconds); not serial in sense that a job is worked on from start to finish except for I/O waits. (See Figure 6 for a simple timing chart.)
- Complex method of intra-program communications is required involving many subprogram calls.
- Subprogram is permitted only a short (10-25 milliseconds) operating time away from Executive.
- Logically complex algorithms are required to process inaccurate and false input data; considerable searching is required, usually for "nearest within limits" rather than for equality.
- Data formatting/deformatting is common, making extensive use of "bit fiddling" instructions to mask, selectively set and clear fields, shift, etc.
- Control of data base update is a major problem in multi-computer or multiprocessor systems; responsibility is normally assigned to one subprogram only for each table or file.
- Building reliability into the program requires from 10-30 percent of the code.
- Fields being processed are mostly short in length—one to 18 bits.
- Relatively small use is made of multiplication and divide instructions; little use is made of floating point and multiple precision.
- Table look-up is often used for "housekeeping" and for decode of inputs such as display action requests.

CONCLUSION

This paper has discussed the nature of tactical data processing systems as it affects the computer program. The single most important problem for the program is the requirement for it to respond in very short times (50-500 milliseconds typically) to many inputs which arrive at unpredictable times and which may peak in number at any time. This requires a program organization which combines common processing functions for better computer memory utilization while permitting the many tasks to rapidly occur that are required to process a typical input.

The characteristics of inputs and outputs, the data base and the computer program have been described and compared with non-tactical systems.

What's different about the hardware in tactical military systems

by EDWARD C. SVENDSEN

Consulting Engineer

and

DONALD L. REAM

Naval Ship Engineering Center
Hyattsville, Maryland

INTRODUCTION

A recent special issue of the Proceedings of the IEEE on computer communications[1] provides an excellent state-of-the-art report on the burgeoning field of digital telecommunications and computer networks. Included are papers on terminals, modems, errors and error control, multiplexing, processors and computer communication networks. Many of the system elements and system concepts in Tactical Military data processing systems are similar to those discussed in the above papers but there are also distinct differences. The purpose of this paper is to identify the differences between the hardware required for Tactical Military Data Processing Systems and the better known hardware requirements for fixed commercial and strategic military data processing systems. As system designers will appreciate, it is not possible to address the hardware aspects independently of the overall system design concepts including the software design. Accordingly, some discussion of overall system design concepts is included as background to the discussion of the hardware requirements. Discussion of the software problems, however, will not be covered except as necessary in this context, since other papers will emphasize the unique software requirements of these systems.

An attempt has been made to keep the system concepts as general as possible in recognition of the varying applications among the tactical systems of the different services, i.e., Air Force, Army, Marine Corps, and Navy. Specific examples will tend to emphasize the Navy systems solely because of the greater experience of the authors in the development of the Navy systems.

TYPICAL SYSTEM CONCEPTS

Typical tactical system

Figure 1 is a greatly simplified block diagram of a typical Tactical system. It consists of one or more sensor subsystems, a single command and decision subsystem, and one or more weapons control subsystems. Communication with the outside world, i.e., other tactical systems and higher level command systems, is usually handled by the command and decision subsystem using automatic radio or land line data links.

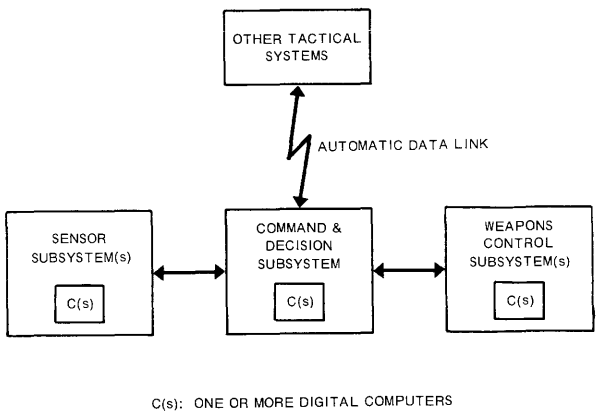


Figure 1—Typical tactical system

Typical hardware found in the sensor subsystems are Radar, Electronic Warfare, Sonar, Optical and Navigation sensors and associated preprocessing and display equipment. Sensor subsystems may vary from relatively simple manual entry systems to very complex automatic entry and signal processing systems using very high-speed, large scale digital processors.

Functions of the command and decision subsystem include (1) the coordination of the data collection from the sensor subsystems and from external sources via the communication data links, (2) correlation of the data to provide a "clear" or filtered display of the tactical situation to the system operators, (3) development of threat evaluation and alternative weapon assignment recommendations for the Commander, and (4) communication of the decisions of the Commander to the weapons control subsystems and to other tactical units. Typical hardware

found in the command and decision subsystem are general purpose digital computers, interactive graphic displays and alpha/numeric readouts, A/D and D/A converters, data link modems, and a very limited number of the more conventional computer peripheral devices. In contrast to the typical large scale commercial data processing systems which employ large numbers of auxiliary storage units such as tapes, drums, or discs and many off line card and tape handling devices, the "pure" tactical system uses only that auxiliary memory required to load and change various operating programs. These include operational, maintenance and training programs which in most cases use no off line hardware except for historical record keeping purposes.

Once the decision to employ a weapon is made, the weapons control subsystem generates the detailed ballistic or vectoring solution and launch orders and the weapon is launched or vectored. Typical weapons are guns, missiles, torpedoes, interceptor and attack aircraft, and active countermeasures. Hardware for these control systems vary widely depending on the type of weapon, but, in general, most of these subsystems now use digital computers and associated interactive displays.

The complexity of the tactical system may vary from a single sensor and a single weapon to the very complex multimission systems employing many different sensors and weapons.

In some cases the sensor or weapons control subsystems are physically separated from the command and decision subsystem. This requires specialized high speed data links using radio or land line communication circuits. The separated systems are typical of some of the transportable ground systems of the tactical Air Force, Army, and the Marine Corps. This variation on the basic tactical system is shown in Figure 2.

On the other hand, the sensor, command and decision, and weapons control subsystems installed on a single mobile platform such as a ship or aircraft having very limited real estate require as much consolidation as possible. The stringent space and weight restraints dictate a much higher degree of integration among the subsystems and raise other very difficult technical problems not encountered in the separated systems. A diagram of this variation is shown in Figure 3.

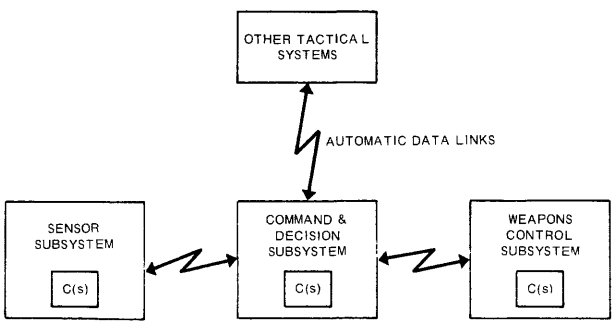


Figure 2—Typical separated system

Tactical support system

Figure 4 illustrates a very different class of tactical systems which has its origin in commercial batch processing and management information systems. This class of systems provides data processing support to the higher level tactical commander in such functions as intelligence, communications, and logistics, or support to the local tactical unit in supply, maintenance and personnel accounting functions. These support systems typically require batch processing, maintenance of large files, and information retrieval programs. An important distinction between these support systems and the other tactical systems is their data bases. In the basic tactical system much of the data base is volatile. Typically, the tactical system starts an operation with almost zero data base which then builds up to a maximum as the operational activity peaks and finally goes back to zero at the end of the operation. On the other hand, the tactical support system usually has a relatively large, stable data base which changes little from day to day.

Typical hardware in these systems include large scale general purpose computers, fewer interactive displays than in the other tactical systems, a large number of auxiliary memory devices such as magnetic tapes and discs, and conventional off-line card and tape handling devices. Functionally, the hardware in these support systems is identical to many commercial and fixed military systems. The major difference is that most of these support systems require mobility or transportability under adverse conditions and hence, the hardware must be designed to meet the same environmental conditions which will be discussed later for the other tactical systems. In some

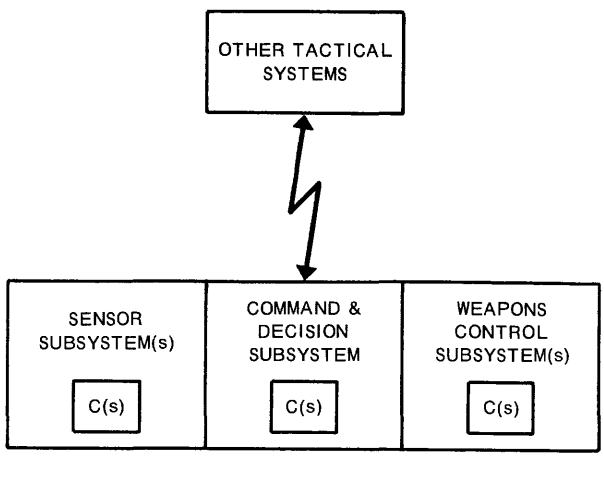


Figure 3—Typical integrated platform system

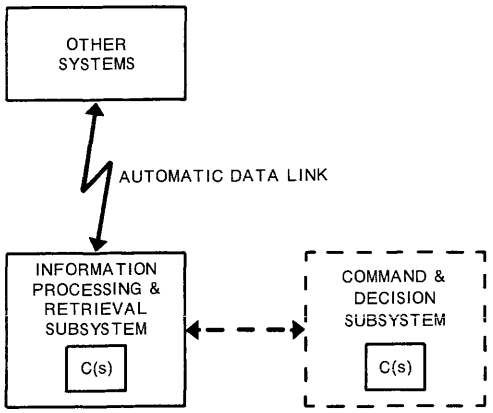


Figure 4—Typical tactical support system

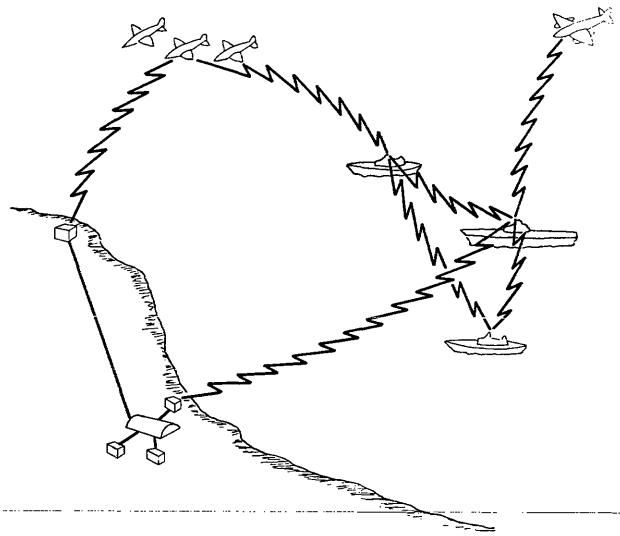


Figure 5—Typical tactical operation

cases, these support systems require colocation or a high degree of integration with the command and decision subsystem of a tactical system as indicated by the dotted lines in Figure 4.

Other variations

There are many other variations to the basic tactical system which involve combinations of the major variations discussed above. In all cases, the driving force behind the system design is the nature of the tactical operations which will be discussed next.

INFLUENCE OF TACTICAL OPERATIONS ON SYSTEM AND HARDWARE DESIGN

The tactical systems described above are required to operate as a network of systems in a variety of tactical warfare operations. Since it is not possible in this unclassified paper to cover in any depth many of the tactical warfare operations, the discussion in this section will be limited to those generalized operational concepts which most significantly influence system and hardware design.

Mobility and flexibility

Perhaps the single, most important operational requirement for the tactical forces is mobility. Figure 5 is a very simplified illustration of a hypothetical tactical situation involving air, sea, and ground forces. The several tactical systems shown are interconnected with a network of automatic tactical data links. Not shown for simplicity are the many other communication links between the tactical units and higher level or rear echelon commands. Ships and aircraft are continuously moving with respect to the area coordinate system. Although the ground systems are usually fixed during periods of operation, they must be capable of rapid repositioning by means of helicopter, aircraft, truck or ship, frequently over rough terrain and during adverse weather conditions. In the absence of an air strip, initial positioning may only be possible by helicopter lift from a ship, or by ship to shore movement in an amphibious landing craft.

The number of tactical units may change from hour to hour, day to day. Additional units may be assigned to the force or the mix of units may be changed as the operation changes. For example, as more ground forces are moved ashore from ships, or by aircraft after air strips are captured or built, the number of ships may be reduced and the number of ground tactical systems increased. Aircraft on station must be relieved several times a day. If the operation continues over weeks or months, the ships must be relieved on station for refueling and supply replenishment. During this continuously changing situation, the readiness of the overall tactical force must be maintained. The mobile, continuously changing tempo of operations create a need for a high degree of flexibility and many technical requirements not usually found in a fixed network of interconnected computer system. Some of these requirements are:

(1) Correlation of the positions of all friendly and enemy units in the area of operation and facilities for the resolution of conflicts caused by the observation of a single target by more than one sensor.
(2) 3D coordinates of sensors and weapons must be converted to the coordinate system used by the local data processing system. This requires accurate alignment and calibration of equipment mounted on a single platform such as a ship or aircraft or accurate positioning and coordinate compensation for any separated sensors or weapons used in a ground tactical system.
(3) Coordinate conversion from the local tactical systems coordinates to a common area coordinate

system. The dynamic positioning of the mobile units requires accurate navigation systems in order to successfully achieve the correlation described in (1) above.

(4) Facilities for communication between the computer equipped systems and the tactical units not equipped with data systems. This usually involves additional processing time and some additional hardware such as a D/A converter and display on the receiving end of a low speed data link.

(5) Provision for rapidly adding or subtracting tactical units from the communications network without interrupting current operations. Each change represents a step function change in the data which must be handled by each of tactical systems in the net.

(6) Provision for the smooth "handover" of the tactical units from one system to another, e.g., shift of control of an interceptor or attack aircraft from a ship to a shore based tactical system.

The hypothetical tactical operation illustrated by Figure 5 is only one of an almost unlimited number of possible combinations of tactical forces. The tactical data systems of each of the services are part of the U.S. general purpose forces which are required to operate globally. In most cases this requires that each system have the flexibility to operate independently or in combination with other tactical systems of its own service and the tactical systems of other services and our allies. In the communication area, this requires a network organization which is designed to meet the essential, common needs of all participating tactical systems rather than to optimize the design for any one system. This also requires a rigid standardization of performance specifications, word formats and operating procedures for all automatic data links used by the tactical systems. The requirement for world wide operation also requires that the hardware for the systems be capable of reliable operation over a wide range of environmental conditions. These environmental requirements which directly result from the mobility requirement represent a major difference between the military tactical hardware and the hardware used in the fixed military and commercial data processing systems.

Continuous on-line, real time operation

There are many so-called real time systems in operation today. Some might consider a payroll to be a real time operation since the objective is to calculate the pay of an employee as of a certain day and deliver the pay check on that same day. The interval between the periodic pay calculations is in terms of a week or weeks and the tolerable delay time is in hours. An airline reservation system is a better example of a real time commercial system. A typical system[2] provides a response time of less than three seconds to the inquiry of a ticket agent. In the tactical data systems there are many events occurring concurrently and asynchronously. Some of the events are created internally by the action of diverse portions of the systems; other events from outside the system, e.g., the motion of a high speed missile or aircraft or automatic communication from another tactical system. The system must handle time critical and time dependent functions and also respond to asynchromous external stimuli. Response time to operator inquiry of a few seconds (as in the airline reservation system) is acceptable for many functions but millisecond response to other operator actions is required. Functions such as target tracking, data base update, automatic communications, and weapons fire control require millisecond response and in some cases with microsecond tolerances.

The real time operations of the tactical systems require that all of the sensor, weapons control, and communication equipment be electrically connected on-line to the system. Delays in switching these equipments in and out of the system or delays in transferring data by means of manual off-line handling to tapes, cards, etc., are not tolerable.

Tactical operations require that the systems be operated around the clock without interruption for long periods of time, sometimes for days or months. This requires very high equipment reliability and redundancy as required to meet some minimum acceptable operational capability at all times. The system must be designed for several alternate modes of operation and "almost real time" system recovery and system reconfiguration when a casualty to any of the system elements is encountered. Ideally, this system recovery and system reconfiguration should occur in "real time," i.e., so that any delays in the operations are imperceptible to the operators required in the alternate mode of operation selected. At present, system recovery and reconfiguration is limited to time of reload of programs from auxiliary storage such as tape or disc and reconstruction of the volatile target tracking data base from own sensors or from target data from other systems by means of automatic data link.

The continuous real time, on-line requirements dictate a multicomputer system with multiprogramming or multiprocessing capability. In fixed commercial or military systems requiring continuous operation, it is possible to provide system reliability by operating redundant computers or completely redundant systems in parallel with the operating system. Examples of this are the airline reservation systems,[2] the SAGE air defense system and some of the NASA ground support systems for the space programs. In most of tactical systems, however, the severe space and weight limitations rule out redundant operation which makes inefficient use of much of the equipment. An alternate design approach has been used successfully in the Naval Tactical Data System for over a decade. In this approach[3] the system on each ship uses two or more identical computers and several identical displays and a minimum of idle or standby equipment. System reliability is achieved by taking advantage of the back up provided by the use of multiple, identical equipments. The system is designed to make use of all the equipment to perform all functions at full capacity, but can be recon-

figured rapidly in the event of casualty to operate at a reduced capacity (e.g., reduced number of tracks) for all functions, or full capacity on the most urgent functions (e.g., full surveillance capability, but reduction in the number of weapons which can be used). During periods of relatively low operational activity this same flexibility permits part of the system to be operated for training or maintenance while the remainder of the system performs the functions required of the operational situation at the time (e.g., surveillance and communication with other ships in the force).

An exception to the above discussion is the airborne tactical system where the requirement for continuous operation is in terms of hours instead of days and weeks as in the case of the other tactical systems. The extreme space and weight constraints on the airborne system legislate against redundant equipment. The emphasis instead is placed on achieving the hardware reliability required to assure the high availability required for the relatively short mission duration.

Man-machine interaction and time-sharing

Although the real time requirements of tactical systems dictate extensive automation of functions, many decisions are made by the human operators and hence the man-machine interfaces are of major importance in system design. Operators must be provided with effective aids to decision making and operator actions must mesh smoothly with automatic machine operations. In most systems the operator interface is provided by one or more interactive CRT displays which time-share the computers in the system. At first glance, these time-sharing systems look very similar to many commercial time-sharing systems which are now in use, but there are important differences. In most commercial applications, the system and program are designed to insure completely independent operations by each operator, i.e., so that each user appears to have private use of the data processing facilities without interference from other users. In the real time tactical application, the system is designed to achieve a controlled, cooperative interdependence among several operators and the system. For example, an action taken by one operator must be immediately made available to other operators in the system to achieve the team effort required for many tactical operations.

Adaptability

Over the life of a tactical system there will be many changes in the nature of the tactical operations. Some of these changes are the result of changes in the expected enemy threat or changes in the tactics used to counter existing threats. New types of weapons, sensors or communication equipment may be substituted for the old. Certain operations (or functions) within the tactical system will require updating, change or replacement by new functions. Within the data processing capacity of the tactical system, this adaptability is generally accomplished by software revision. It follows that the data processing hardware design must be able to accommodate or adapt to such changes in function without hardware redesign or disruption of other ongoing functions in the system. The requirement for adaptability over the life of a system is a strong argument for modular design and hardware expandability. For example, the computer design should allow for the addition of such modules as CPU's, directly addressable memory modules or I/O channels without any changes in the basic system design.

HARDWARE REQUIREMENTS

It is impractical to make an exact comparison of military and commercial data processing equipment because of the wide variation in the characteristics of these equipments and because of the changes that have been made over the years. Since the military systems were the first major users of real-time and time-sharing techniques, characteristics of the hardware used in these systems were decidedly different than commercial hardware. However, over the years with increased use of real time systems by FAA, NASA, etc., using commercially available equipment, there have been many changes which closely resemble the functions found in military real-time systems. Today it is difficult to identify technical features which are clearly and universally unique to one of the other types of hardware. The discussion of the hardware requirements which follow will emphasize those features which generally are not found in commercial equipment.

Requirements applicable to all hardware

Environmental

As was discussed earlier the hardware for tactical military systems must operate reliably for long periods of time under a wide range of environmental conditions. A detailed examination of the environmental specifications of the various services is beyond the scope of this paper. Discussion will be limited to some of the environmental factors which account for the major differences between the tactical military hardware and the commercial or fixed military hardware.

The hardware of all the services must operate over a wide range of temperature and humidity and also withstand severe shock and vibration. As an example of the difference in requirements, a typical military computer must operate over a 100 degree F range as compared to 20 degrees for commercial. In addition, airborne hardware must be capable of operation at high altitudes and at high "G" forces. Ships hardware must operate under conditions of severe roll, pitch and heave and also withstand the corrosive salt water environment. Ground systems hardware must contend with sand, dust, very rough handling over rugged terrain and extreme storage requirements.

When feasible, a controlled environment (e.g., air conditioning) is provided to improve operator efficiency or system reliability, but in most cases the equipment is required to operate satisfactorily with no long term damage in the event of casualty to the environmental control equipment.

On the mobile platform systems such as ships and aircraft, electromagnetic compatability becomes a major technical problem. For example, because of the limited real estate, high power radar and communication transmitters frequently must be installed in close proximity to sensitive receivers. The design option of widely spacing the transmitting and receiving antennas to reduce RF interference is not available as in the case of the separated tactical systems.

The hardware for all of the systems and particularly for the mobile systems must meet severe size, weight and power limitations which require special packaging and cooling techniques not required in the hardware for commercial systems. To illustrate the size difference, large scale military computers including memory, CPU and I/O are typically packaged in less than 10 to 15 cu. ft. This is an order of magnitude smaller than the equivalent commercial computer.

Meeting all of the above requirements is a major factor in achieving the high reliability required in the tactical military system and has a major impact on the cost of military hardware. An important factor is the extensive testing which is necessary to prove that the reliability requirements will be met in the operational environment.

Logistics support

The tactical military systems must be designed so that they can be operated and maintained by military personnel in remote locations without ready access to field engineers or rear echelon repair and supply activities. This requirement for self sufficiency requires an approach to system and hardware design different from commercial systems. For example, maintenance documentation must be more extensive and diagnostics more highly developed for the military technician because of the absence of engineering backup in the field. Rapid repair requires extensive use of replacement modules and the design of modules must match the support philosophy used. On board repair of modules requires on board piece part support. Rear echelon repair of modules or throw away modules require that spare modules be available to the technician.

The maintenance philosophy differs widely for the various tactical systems and is strongly influenced by the nature of the operations. For example, Navy ships are required to operate for several weeks or months away from bases. The Navy maintenance policy is primarily on-board maintenance of all hardware with very limited rear echelon support over the duration of the ships operation. On the other hand, most airborne systems have mission durations of only a few hours and hence are normally designed for no on board repair during the mission but require rear echelon support upon return to base (e.g., to an aircraft carrier or shore air base).

Computers

Military computers are typically required to meet reliability requirements which are an order of magnitude greater than equivalent commercial computers. To meet this higher reliability under the environmental conditions discussed above, circuits must be designed to operate with much wider margins and use special components, e.g., lithium cores for memory to meet the temperature requirements.

Some of the specific functional characteristics which are associated with military computers are:

(1) Large, high speed, directly addressable memories are required since in most instances these systems cannot tolerate the delays of "rolling" information in and out from auxiliary memories. This is governed by the requirement that many of the systems must respond to tasks within specific critical time frames.
(2) Automatic system recovery and reconfiguration requires special hardware features such as small, read only memories to bootstrap the reloading of the operational program.
(3) Since tactical systems are not designed for maximum throughput but rather to respond to job requests within specific time intervals, the use of high resolution clocks and well-organized interrupt capability with many different states and priority levels is required.
(4) Because of the large number and types of inputs and outputs to these real time systems the actual system operation is dependent upon efficient I/O operation. As a result there are normally many bidirectional, buffered I/O channels with separate access to memory. In addition, special I/O functions such as externally specified index, externally specified address and intercomputer operations are features in these systems. It should be noted that when commercial computers have been adapted to shorebased, real time systems usually there have been special I/O cabinets and multiplexers added to the computers to meet the abnormal I/O requirements.
(5) Since data used in these systems is usually bit oriented and byte oriented, word lengths are not necessarily dependent upon some multiple of a byte, but are determined by accuracy of required calculations. Military computers have had word lengths of 14, 16, 18, 21, 24, 30, 32, and 36 bits.

Displays

The interactive CRT displays, which are widely used in the tactical systems, are similar to some of the displays

found in the commercial time-sharing systems. There are, however, some differences in addition to the requirement for an order of magnitude increase in reliability over the commercial displays. For example, some of the displays are required to display raw sensor data (video) concurrently with computer generated spots, symbols, and vectors. This requires very wide bandwidth deflection amplifiers and higher deflection speeds than required for the more conventional interactive displays, which display alpha-numerics or graphics. Typical tactical displays are required to display the tactical "picture" over a wide range of geographical coordinates; from the large area surveillance "picture" to the very short range "picture" of the local tactical situation, or an expanded "picture" of the area of action around[4] a distant remote tactical unit. This requires a "smart terminal" (i.e., small processor in the display) or a combination of some logic in the display and the remainder of the processing in one of the system computers. The computer generated data on CRT displays must be refreshed at a high rate to provide a "flicker free" picture, either by means of a small memory in the display, or from one of the system computers. Trade off between the "smart display" and displays driven from a central processor is influenced by the overall system design. The "smart display" is usually found in systems requiring very few displays. In systems requiring many displays (e.g., 6 to 25) or interaction among many operators, it is usually more cost effective to drive the displays from central processors. The design of these display systems requires adequate consideration for casualty modes of operation. This requires that most of the displays be designed with the flexibility to be used for several functions and the capability for rapid switching in and out of the system and for change in functions without program changes, e.g., change of function by a selector switch on the display which automatically changes labels on operator action buttons and readouts and informs computer of change of function. The design of the display hardware and software must also insure that the system is resistant to inadverdent operator error.

Communications

The real time, on-line tactical systems require automatic computer to computer communications without human intervention, except for network initialization or change in network participation. Extensive error detection and correction is required to achieve reliable communications over a wide range of noise conditions. This is particularly true in the case of the radio links where there are wide variations in error rates caused by changes in propagation paths or by jamming.

There are two types of automatic data links used in tactical systems. One is used to net all the tactical systems together, and the other is used to interconnect two widely separated subsystems, or as a control link to a mobile weapons systems, e.g., a ground to air link to control an aircraft.

The data link used to net all of the systems together must necessarily be designed to rigid technical performance specifications including data word formats. In addition, the design and operating philosophy of each of the tactical systems in the net must be compatible with the design restraints of the common links. For example, the mobility requirements of many of the tactical systems require that the common operating net make use of radio data links. Because of the limited bandwidth available for these radio links, each data source in the net must preprocess the raw data in order not to monopolize the link or saturate the processors at the receiving end.

On the other hand, the point to point subsystem interconnecting and control links can be designed to provide a more optimum match between subsystems. When such links are required to be used by several different tactical systems, however, rigid standardization is required, e.g., the surface to air data links to aircraft which must be controlled by either ship or ground systems.

Functionally, the hardware in the military systems is identical to that used in many commercial digital communication systems. The major differences are in the higher reliability requirements and the facilities required for flexible network control.

Other peripherals

In some of the tactical systems, large numbers of equipment must be connected on line to the system by means of A/D and D/A converters. Functionally, these converters are identical to many converters found in industry, except for the requirement to achieve high accuracy and rapid response times over a wide range of environmental conditions.

Other peripherals such as magnetic tape units, discs, printers, punched card and paper tape devices, when used, are usually adaptations of commercially available hardware to meet the stringent environmental requirements.

SYSTEM STATUS AND TRENDS

Systems

The design, development, production and operational introduction of military tactical data processing systems has been and continues to be a difficult and challenging problem for both the military and industry. An indication of the difficulty of the design of these systems is that very few of the many systems under development since the mid 50's have survived the development cycle and reached operational maturity. There are many reasons for the failures but, in the opinion of the authors, the major factors are (1) premature obsolescence of the systems because of faulty systems design concepts which did not recognize the continuously changing nature of tactical operations, and (2) management of system development by people who did not understand the operational prob-

lem or who had no actual experience with data processing systems or both.

On the positive side, there are several tactical systems now being operated successfully. One such system is the Navy Tactical Data System (NTDS)[3] which was first installed on 3 ships in 1961 and is now on 48 ships and planned for most future combatant ships. For over a decade this system has been operated reliably by Navy operators and technicians for long periods of time (for months in many cases) and with practically no support required from contractors' engineers or rear echelon repair activities. Since 1961 this real time, on-line system has continuously demonstrated the feasibility of:

(1) multiprogramming in multiple computer systems (1-4 computers)
(2) time sharing with many (up to 25) interactive CRT displays.
(3) automatic computer to computer communications between ships using digital radio links, and since the mid 60's between ships, the Navy Airborne Tactical Data System (ATDS) and the Marine Corps Tactical Data System (MTDS).

The major factors in the successful development of the NTDS were:

(1) Continuous, active participation in the development by several key operational officers representing the user.
(2) Staffing of the project management office in the developing agency with several key officers and civilian engineers having some knowledge of the operational problem and/or prior experience in digital data processing.
(3) Establishment, early in the development, of a Navy programming activity staffed with operational officers and civilian programmers. This activity was initially under the control of the system developing agency but later assigned to a user command to support the operational ships.
(4) Excellent performance by several key industrial firms and a Navy Laboratory in the development of systems hardware and software. The high reliability of the hardware from the beginning was a major factor in the rapid solution of early system and software problems.

Development of the early tactical data systems concentrated on the command and control and communication functions, i.e., the Command and Decision Subsystem of the overall tactical system. Most of the sensor and weapons control subsystems at that time were analog systems and the tie-in with the Command and Decision subsystem was made by extensive use of A/D and D/A converters. Today most of the new sensor and weapons control subsystems are being developed using digital computers and digital interactive displays. Much of the current systems development effort is to standardize the hardware and software of the various systems and better integrate the operational functions in order to achieve more effective and less costly overall systems in terms of life cycle costs such as training, software support, supply support, etc. This increased integration and standardization has created a significant management problem because of the greater interaction among the subsystems and the overall tactical system. Progress toward greater integration and standardization is dependent upon the solution of the system management problem, not the technical problems.

In the system reliability area, the requirement for real-time system recovery is being implemented in a specific system and shows promise for widespread use in future systems.

Hardware

Computers and data processors

In 1965 the authors[3] suggested that the multicomputer system using multiprocessing computers appeared to be a competitor to the multiprogrammed, multicomputer system, but that actual experience was required to verify the performance of these systems before they could be seriously considered for use in real-time tactical systems. Enough experience has been accumulated with available multiprocessor computers to warrant their use in tactical systems. Acceptance of their use is slow as indicated by the fact that available multiprocessor computers are in many cases being multiprogrammed. This is a management, not a technical, problem. It is evident that multiprocessing will be more widely accepted and used in both commercial and military systems.

In applying LSI to military computers and processors, the major stumbling block appears to be the problem of testing and proofing the hardware. Because of the high reliability requirements over a wide range of environmental conditions and the relatively low volume production of military systems, it is highly probable that widespread application of LSI to commercial computers will precede its use in tactical military systems.

Preprocessing of sensor data before transmission to the Command and Decision computers is usually done by very specialized high speed processors or combinations of special purpose logic and a general purpose computer. In the area of special purpose logic, it is likely that much of the hard wired logic will be supplanted by associative and /or microprogrammed processors as the reliability and cost problems are overcome, and applications become better understood.

Displays

Although there are many promising developments in display technology (e.g., large screen displays) it is unlikely that the interactive CRT display will be replaced in the foreseeable future as the "workhorse" display in

the tactical systems. There will be evolutionary changes in these displays to increase their utility as a standard display in all subsystems and at the same time preserve the unique features required to match each operator to the system. An evolutionary not revolutionary policy is dictated not solely because the state of the technology but by the requirement to maintain a long term commonality of operator functions in order to maximize the effectiveness of operator training. There will also be an evolutionary trend to automate more functions and reduce the number of operators in the system.

Communications

In commercial computer communication networks the trend is to do more processing at the terminals in order to minimize the communication line costs. In the tactical systems, the trend will be the same but for a different reason, i.e., the limited bandwidths available for radio data links.

There will be a trend toward adaptive communications, i.e., automatic adjustment of the redundancy in the data link in response to changes in the error rates.

It is expected that the tactical systems will make extensive use of satellite communications in the future.

Summary

Because of the rapid increase in the use of multiprogramming, multiprocessing, interactive displays, remote terminals and on-line communications in time-sharing and real time commercial and fixed military systems, it is clear that the functional characteristics of the hardware for these systems and the Tactical Military systems will tend to converge. The high reliability requirements of the tactical military hardware under a wide range of environmental conditions will continue to be the major difference.

REFERENCES

1. Special Issue on Computer Communications, Proceedings of the IEEE, November 1972.
2. J. R. Knight, "A Case Study: Airlines Reservations Systems," *Proceedings of the IEEE*, November 1972, pp 1423-1431.
3. E. C. Svendsen and D. L. Ream, "Design of a Real-Time Data Processing System," *Proceedings of the IFIP Congress 65* Vol I, pp. 291-296, May 1965.
4. L. C. Hobbs, "Terminals," *Proceedings of the IEEE*, November 1972, p. 1282.

What's different about tactical military languages and compilers

by RAYMOND J. RUBEY

Logicon, Inc.
San Pedro, California

BACKGROUND

Until recently, the programming for tactical military computer applications relied on assembly or machine-oriented languages in contrast to the widespread use of higher-order languages in commercial applications. In the past, the tactical military software development process was plagued with many problems, including the relatively high cost on a per-instruction basis, the long development time required, the necessity for highly-trained engineer-programmers, the non-transferability of the resultant programs, and the considerable difficulty of effective program maintenance. The military customer who had to pay the penalties resulting from these problems became convinced that at least part of the reason for the problem was the reliance on MOLs. As a consequence, the military customer provided the impetus and funding to apply HOL concepts and technology to the unique problems of the tactical military software environment. At the same time, NASA, which encountered the same problems in developing real-time software for space applications, took the same approach. The conclusions reached from the resulting HOL studies and developments has led the customer, in many cases, to insist on the use of a HOL unless it could be conclusively demonstrated that such usage was infeasible.

TACTICAL MILITARY HOL ALTERNATIVES

An organization beginning the development of tactical military software has several alternatives as to the HOL it can use. First, an existing commercial HOL, such as FORTRAN, ALGOL, or PL/I could be selected. This alternative has been universally rejected because of the limitations of these languages in tactical military programming. A second alternative is to select and modify a language originally intended for another application to make it suitable for the tactical military software environment. This approach was taken, for example, in the selection of a JOVIAL J3 subset and its modifications for the B-1 avionics software development. A third alternative is to select a more specialized language that was developed in response to requirements of the tactical military software environment or the closely related real-time space software environment. The languages in this category include the Navy's Compiler Monitor System-2 (CMS-2), the Air Force's Space Programming Language (SPL), NASA's Computer Language for Aeronautics and Space Programming (CLASP), and NASA's HAL. Whether the second or third alternative is taken, the language must contain the facilities for performing functions not common in most commerical programming endeavors. Because of the different approaches of the language designers in solving the problems in the tactical military environment, the languages indicated above are significantly different with regard to the way these facilities are provided. The following paragraphs provide generalizations about these facilities.

TACTICAL MILITARY HOL FACILITIES

A tactical military program development is a more varied activity from the software viewpoint than the typical commercial program development. The tactical military programmer begins with an empty computer and must code his own executive program, I/O routine, subroutine library, and diagnostic routines. Thus a tactical military HOL must provide the facilities which, in the commercial environment, are associated with systems programming. Of course many of the facilities in commercial HOLs are essential in tactical military HOLs, including arithmetic, conditional, looping, and transfer-of-control statements.

Besides dealing with the usual numeric data, a tactical military HOL must allow for the definition and manipulation of logical, Boolean, textual, and character data. It also must provide the facility for manipulating portions of data words down to a single bit as well as the usual full data words. All of the conventional arithmetic and Boolean operations on these portions of words should be provided. The programmer uses this facility to operate on the varied inputs and outputs received and transmitted by the typical tactical computer and to create the needed data structures.

Many of the calculations performed in tactical military applications involve the movement of objects in three-

dimensional space. An effective tactical military HOL simplifies the programming of these calculations by allowing for the definition of appropriate arrays and by providing powerful non-scalar operators. Examples of such operators are vector dot product, vector cross product, and matrix multiply.

No matter how rich in facilities a HOL is, there may be some functions that cannot be easily or efficiently performed. Thus many HOLs provide for easy regression to assembly or machine-oriented languages. If this facility is not provided, the object code generated by the compiler must be modified, which is more difficult and more likely to induce errors.

A tactical military computer has specific locations dedicated to a particular purpose through hard wiring. Examples are the dedicated, fixed locations where the computer registers are automatically copied when an interrupt occurs. Through declaration statements, the HOL must provide for the association of programmer-defined symbols with such hardware locations and functions. Similarly, the HOL must allow the programmer to allocate the operational tactical military program and data to specific computer locations according to a declared memory map.

The architecture of many tactical military computers and their application often require the use of fixed-point arithmetic in addition to or in place of the floating-point arithmetic that is universal in other applications. This requires that the HOL have facilities for the declaration of fixed-point data containing both integer and fractional parts, and for the utilization of such data in calculations with a minimum of programmer effort. The scaling operations performed for fixed-point arithmetic calculations become an important part of the HOL semantics. The increasing use of floating-point architectures are likely to solve many of the existing problems before completely effective language and compiler solutions are found.

The real-time nature of tactical military applications requires that suitable HOLs contain facilities for the real-time control of the programs. Facilities are needed to enable and disable interrupts, to correlate specific program actions with particular interrupts, and to indicate the required interrupt levels. Less obvious but equally important is the need for facilities to control the accessing of data by several levels of interrupts. For example, an array computed at one interrupt level should not be referenced at a higher level without checking that the complete array has been computed.

The debugging and validation of a tactical military software system is the most expensive and difficult part of the software development cycle. A tactical military HOL must assist in these debugging and validation efforts. First, the language itself must have a minimum number of error-prone features or syntactic constructions. Second, it must enable the creation of a compiler that can perform a considerable degree of compile-time fault diagnosis. Finally, it must provide for the generation of run-time diagnostics, particularly those produced by simulated execution of the object code obtained without any changes in the object code itself. While such features as these are desirable in a commercial HOL, they are mandatory in a tactical military HOL.

One problem that has not been completely solved in any existing tactical military HOL is the definition of a computer-independent and general input/output facility. This problem is caused by the widely varying input/output devices in tactical military systems, the considerable differences in the input/output modes of tactical military computers, and the stringent constraints on real-time input/output operations. Indeed, some language designs are based on the assumption that input/output functions will continue to be coded in the appropriate MOL and therefore do not provide input/output facilities. Other languages provide basic input/output facilities little different from those provided in commercial HOLs. Regardless of the input/output facilities in the HOL, many compiler implementations contain custom-tailored input/output statements that are both computer and application dependent.

TACTICAL MILITARY COMPILER CHARACTERISTICS

A single commerical computer installation may execute a hundred programs in the course of a week; a single tactical military computer program, on the other hand, may be the only program executed in a hundred tactical military computers. While the savings in programmer labor made possible through HOL usage more than compensates for compiler inefficiencies in the commercial environment, this is not necessarily true in the tactical military environment. If a tactical military compiler generates code that requires 50 percent more memory than the equivalent MOL program, then memories 50 percent larger for all one hundred computers have to be purchased. The efficiency of a tactical military compiler is usually measured by determining the increase in memory and execution time of the object code it generates compared with an expertly-coded MOP program. A high efficiency, usually on the order of 80 percent for both time and space, is crucial to the acceptance of a tactical military compiler. Therefore, the typical tactical military compiler has more extensive optimization features than most commercial compilers. There is usually more emphasis on local optimization than global optimization because this approach appears to offer the biggest payoff at the lowest cost and because the effect of many global optimizations can be obtained by appropriate modifications to the source code.

The task of creating a highly efficient compiler is facilitated because the usual emphasis on compilation speed is absent in the tactical military environment and because the compiler usually executes on a larger general-purpose computer rather than on the tactical military computer itself. This enables the use of optimization algorithms

that take a long time to execute and require a large amount of compiler storage.

Many tactical military HOLs contain features that allow the programmer to control the optimization that the compiler performs. For example, the programmer may specify that space optimization is more important in one portion of the program, and time optimization in another portion. This is desirable because the minor-cycle portion of a tactical military program may be executed 50 times more frequently than the major cycle portion, and unless the appropriate areas are delimited, the compiler does not have sufficient information for effective optimization.

Compilers for commercial applications are supplied by the computer manufacturer; tactical military compilers are often developed by the same organization that develops the operational software or are supplied by the military customer. While a commercial compiler may be used in hundreds of installations, a tactical military compiler may be used by only one or two organizations. The cost of tactical military compiler development can therefore be a significant portion of the total tactical military software cost. Considerable attention has been paid to methods of reducing compiler cost; in particular the meta-compiler approach has been under study. Although the meta-compiler approach has shown considerable promise, most tactical military compilers have followed relatively conventional designs. Usually the code generation module is written so that it may be easily replaced when a compiler for another tactical military computer is needed.

Because it receives much less usage than a commercial compiler, a tactical military compiler would be very similar in reliability to the first release of a commercial compiler unless special testing precautions are taken. Anyone familiar with the lack of reliability in early releases of commercial compilers can appreciate how much effort must be expended in tactical military compiler testing. Even with extensive testing, those responsible for the validation of operational tactical military programs pay considerable attention to the object code.

FUTURE LANGUAGE AND COMPILER TRENDS

The use of HOLs in tactical military software development will continue to grow, largely because of pressure from the military customer to reduce development cost, simplify maintenance, and provide visibility into software behavior. The recent rash of overlapping language definition efforts will come to an end and language standardization will become more important, again largely because of pressure from the military customer. The language or languages that prove to be successful in large, significant tactical military software developments will have the greatest chance of being selected as the industry standard, regardless of any theoretical virtues or faults. This will parallel the Air Force's selection of JOVIAL as its standard command and control language.

The greatest improvement in languages will come in the features that aid development of reliable operational software. Diagnostic directives, required redundant statements, and compile-time limit and validity checks will be added to existing languages and their use will become mandatory in an effort to reduce debugging and validation costs. The current emphasis in language design will thus shift from defining succinct and elegant ways to describe the procedures that should be executed. Instead, the emphasis will be on creating forms that more clearly (and even redundantly) describe those procedures.

The cost and long development times for tactical military compilers will be the chief inhibiting factor in the use of HOLs in tactical military applications. Efficiency considerations will grow less important as more improvements are made in optimization methods and as the cost of computer hardware itself continues to become cheaper.

Finally, the tactical military computers themselves will become more suited to the HOL approach. Initially, this will take the form of computers whose organization and instruction set facilitate the writing of compilers. Ultimately, the tactical military computers will execute an HOL directly, obviating the need for a compiler.

What's different about tactical executive systems

by WILLIAM G. PHILLIPS

Radio Corporation of America
Moorestown, New Jersey

The program for a computerized command-and-control system is generally a combination of critically time-constrained real-time tasks, which directly control the tactical mission environment, and non-real time tasks, which support the system. This computer program structure is the basis for determining the allocation of the total available processing time for a complete mission cycle.

TIMING ALLOCATION

Since tactical command-and-control systems (Figure 1) are triggered by a series of predictable and non-predictable events, the computer-program task allocations must be designed for complete flexibility within the total available processing time period. In the case of predictable event triggers, the design may be simple to the extent of repetitive processing of a single chain of tasks, called a "thread." In the case of unpredictable event triggers, such as special-threat target detections, the design must be more complex to the point of interleaving threads. The process of interleaving threads presents an interesting timing problem within this type of system because of the requirement that a real-time thread must complete its processing in a predefined critical time period. This time period is frequently a function of the design requirements of the interfacing tactical equipment.
equipment.

Figure 2 illustrates a processing sequence where the triggering events are predictably separated and therefore the thread allocations (P_i) and their respective critical time periods (Q_i) are predictably separated. The slots which occur between threads (R_i) are available for processing of non-real time tasks. The real-time tasks (q_{ij}), as individual items, must all satisfy their individual time allocations within their respective critical time thread period Q_i. This timing constraint can be represented by the following inequality:

$$Q_i > \sum t(q_{ij}) \tag{1}$$

where $t(q_{ij})$ is the time allocation associated with task q_{ij}.

If the non-real-time tasks are also time constrained to a fixed completion period (T_p), then the general equation for timing allocation within a complete processing sequence (T_p) is:

$$T_p > \sum_i (P_i + R_i) \tag{2}$$

which describes the inequality to be satisfied by the combination of real-time and non-real time tasks over the total available processing period, T_p. This period represents a complete cycle of tactical events, such as radar-track processing, weapons assignment and firing, and special-threat target processing. The critical time-thread period (Q_i) of Ineq. 1 represents intervals of tactical events, such as radar-target detection, weapons designation, and special-threat target-assignment processing.

This time allocation can also be easily applied to non-tactical systems, which frequently allocate a range of time (Q_i), in equal quantums, to a group of application tasks. Any unused time (R_i) between the actual completion of a quantum period cycle, i.e. all process state tasks have completed their respective quantum period of execution (P_i), and the beginning of period Q_i+1 is allocated to system background processing such as on-line fault analysis or some accounting procedures.

The major difference between the two types of systems is the criticality of satisfying Q_i in inequality (1). Non-tactical systems most frequently are responsible for the scheduling and processing of a group of non-related homogeneous tasks, which are not critically dependent upon when they initiate or complete processing. That is, the tasks will not have failed their intended purpose if they complete processing two or three seconds later than if they had been run in a "batch" environment. This philosophy can also be applied to some real-time systems, such as a communications network which, while a two or three second delay would postpone the completion or initiation of a call, it would not cause the system to fail its intended "mission" of initiating and completing phone calls in sufficient time to be compatible with human reaction speed. Tactical systems, on the other hand, are constrained in time by high speed device interface requirements which frequently must be satisfied within tolerances no greater than a few milliseconds. Any perturbation to tactical task scheduling could cause these tolerances to be violated, thereby possibly causing the

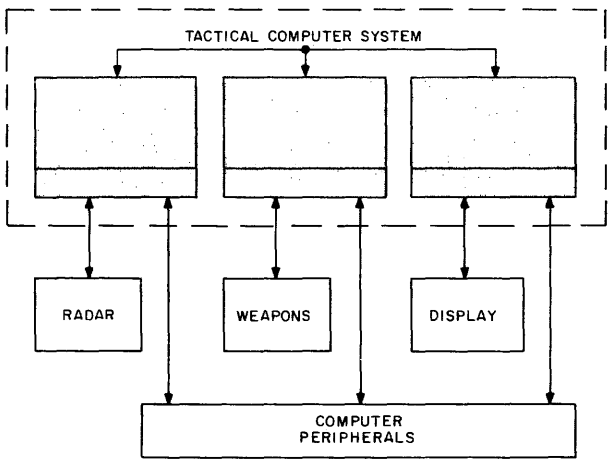


Figure 1—Command and control computer program complex

intended mission to degrade or fail. The degree of impact due to mission failure (criticality) is far greater in a tactical system primarily because of the direct relationship between mission success and human lives.

An added perturbation to the system-timing allocation is the introduction of the executive program tasks that support the scheduling and dispatching of the system tasks. This additional allocation can be represented by expanding Ineq. 2:

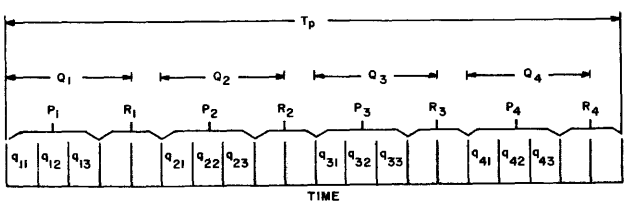
$$T_p > \sum_{ij}[t(q_{ij})+t(e_{ij})]+\sum_{ik}[t(r_{ik})+t(e_{ik})] \quad (3)$$

where $t(e_{ij,ik})$ is the time allocation associated with task e_{if} or e_{ik} and $t(r_{ik})$ is the time allocation associated with task r_{ik}.

This general inequality must be satisfied for timing allocation over the period T_p; it includes the real-time task times, $t(q_{ij})$; the non-real time task times, $t(r_{ik})$; and the executive task times, $t(e_{ij})$ and $t(e_{ik})$. We can also expand Ineq. 1 for the critical-thread periods (Q_i) to include the executive tasks:

$$Q_i \sum_{ij} t(q_{ij})+t(e_{ij}) \quad (4)$$

It is apparent from Inequalities 3 and 4 that there are many unknowns: in addition to ascertaining the process-



T_p = Total available processing time for a mission cycle

Q_i = Allocation of Critical time period within which real-time threat P_i must be completed

P_i = Processing thread i, consisting of a processing sequence of real-time tasks, q_{ij}

R_i = Processing thread i, consisting of a processing sequence of non-real time background tasks, r_{ij}.

Figure 2—Processing time allocation non-interleaved

ing-time allocations for the real-time and non-real-time tasks, we must also determine the time expenditures of the executive tasks. This is further complicated by the fact that executive-task durations may vary because of the varying types of services to be performed (such as I/O scheduling), the type of real-time task scheduling (i.e., immediate with or without messages, time delayed, etc.), and the scheduling queue backlogs. Until all of these unknowns are determined, or at least closely predicted. Inequalities 3 and 4 cannot be credibly satisfied.

A practical solution to this problem is to arbitrarily allocate a budget of a fixed percentage of the total available processing time (T_p) to the executive tasks $\Sigma t(e_{ij})$ and $\Sigma t(e_{ik})$. A more precise procedure is to further allocate a fixed percentage of the available critical-thread period Q_i to the executive tasks $\Sigma t(e_{ij})$. A typical allocation, at the beginning of system development, is 10 to 15 percent for both of these periods. As the development progresses and the task timings become more defined, the terms of the inequalities must be adjusted. This, in fact, is an excellent method of ensuring that the task design is meeting its timing allocations; for if the inequalities fail to be satisfied, the system integrity is compromised, and the system must be redesigned.

Real-time command and control systems differ in complexity. A simple system, with predictably sequenced trigger events, has its real-time-thread critical periods (Q_i) allocated somewhat as in Figure 2, with the required condition that the following inequality be satisfied:

$$T_p > \sum_i Q_i \quad (5)$$

However, some systems are more complex because of unpredictable sequences of trigger events. These types of systems frequently require that real-time threads overlap each other, but that each thread must still complete processing in its allocated critical period, as illustrated in Figure 3. This figure shows the critical real-time periods, Q_i, overlapping the non-real time tasks, R_i, naturally being delayed until the completion of all real-time threads. This allocation permits us to then concentrate on the critical real-time periods, Q_i, and to process the low level, R_i, tasks, in a background mode, if and when time is available during T_p. This overlapping (interleaving) process is described by the following inequalities:

$$\sum_i Q_i > T_p \quad (6)$$

$$Q_i > \sum_j t(q_{ij})+e_i+\sum t(q) \quad (7)$$

where e_t represents the total fixed executive overhead time allocated for the period, Q_i, and $t(q)$ represents the tasks interleaved in Q_i.

This type of complex system requires that the executive program, through a scheduling mechanism, manage the interleaving process to ensure that inequalities 3 and 7 remain satisfied during task processing. To accomplish this, the executive-program scheduling mechanism must be designed to manage a dynamic queue based on task

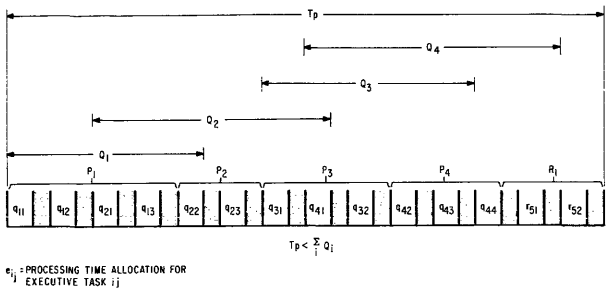


Figure 3—Processing time allocation—interleaved

priorities and to resolve any timing conflicts between competing tasks across threads, as described by inequality 7.

It is not unusual to encounter system requirements that dictate that the first task within a thread, q_{i1}, (Figure 3)—and therefore the thread itself—shall be repetitively triggered at some frequency relative to the occurrence of an event; that is, the thread initiates a processing sequence, P_i, (Figure 3) repeatedly at some frequency after some initial event trigger. This may occur for three general reasons in a command and control system:

(1) Periodic interface requirements with time-pulsed radars or other similar equipment.
(2) Periodic interface requirements with display consoles, which require refreshed data.
(3) Periodic polling of interfacing equipments for input messages.

In the case where a thread is scheduled in constant intervals, relative to a single event (i.e., the triggering event always occurs at the same time within each T_p interval), the thread timing allocations in Figure 3 are identical for each succeeding T_p interval. However, in complex systems, the possibility exists that some periodic threads will be scheduled some constant frequency after, or possibly before, the occurrence of an unpredictable event. This case will then cause the thread critical allocation times, Q_i, to drift from one T_p period to another, as illustrated in Figure 4. This drifting would also occur for those cases where a thread was scheduled with a variable frequency.

It is important to understand, at this point, that tactical tasks are not amenable to a multiprogramming scheduling technique because of their homogeneous functional properties. Unlike a commercial data center environment, where each task in the processing queue is completely heterogeneous and consequently is not dependent on the processing state of any other task; the tactical system tasks, within a thread, are dependent upon their predecessor/s to supply both data and initiation triggers. This dependency is required primarily because tactical tasks frequently interface with equipments which require time tagged data from other equipments. For example, it is unrealistic to execute a task which supplies data to a display console, prior to the completion of a predecessor task whose function was to pre-process the data from a radar buffer.

Let us now summarize the four common types of critical real-time tasks:

(1) The dynamic task that must be scheduled strictly according to a priority sequence.
(2) A periodic task that must be scheduled repetitively at a fixed frequency relative to a predictable event occurrence.
(3) A periodic task that must be scheduled repetitively at a fixed frequency relative to an unpredictable event occurrence.
(4) A periodic task that is scheduled repetitively at a variable frequency relative to a predictable event occurrence.

Since a mixture of these type tasks may be required to complete processing within the same critical-thread period and since each task will perform a unique tactical function, a priority scheduling philosophy must be developed, which will ensure the hierarchy of tasks in relation to one another. This is especially true in the case where a periodic task, and possibly its associated thread, is unpredictably triggered while a lower relative priority task is processing. Inequality 7 represents the total time allocated to a complete mix of real-time tasks over a critical processing period, Q_i, assuming, of course, that random-interrupt processing is included in the appropriate allocation; and therefore is the principal timing requirement to be satisfied by the design of the executive scheduling mechanism.

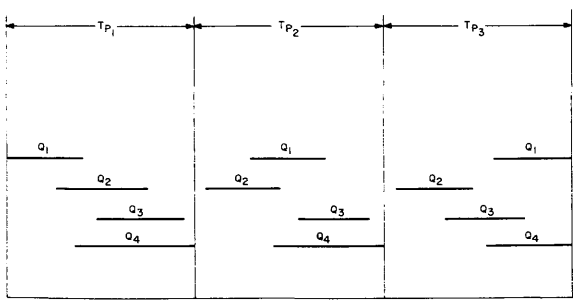


Figure 4—Drifting critical time periods

EXECUTIVE DESIGN APPROACH

The executive program, to satisfy the above timing allocations, must provide efficient mechanisms for performing the following functions:

Scheduling critical real-time tasks according to a dynamically changing priority-sensitive environment.
Interleaving processing threads.
Monitoring the processing of all tasks and threads to ensure critical time periods and total available processing time periods are not violated.

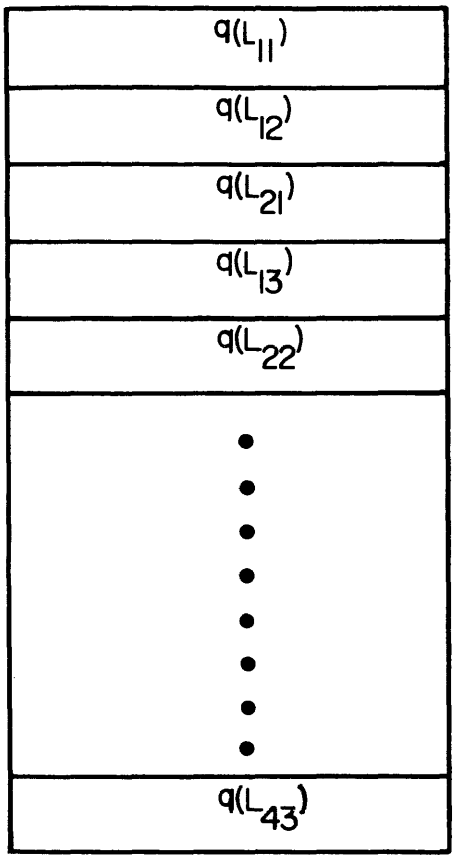


Figure 5—Single-level queuing model

If a unique task program is equivalenced to each timing allocation, q_{ij}, in Figure 3, we can state the following general priority characteristics of threaded tasks in this type of command and control system:

$$q(L_{i1}) \geq q(L_{i2}) \geq \cdots \geq q(L_{in}) \tag{8}$$

where $q(L_{ij})$ is the priority of task q_{ij}, and

$$P(L_i) = q(L_{i1}) \tag{9}$$

where $P(L_i)$ is the priority of thread P_i.

Inequality 8 shows that tasks are always structured within their respective threads in descending priority order, independently dynamic. This priority structure differs considerably from most non-tactical systems, which contain tasks of different priorities within a single thread and the execution of any specific task is a function of both priority and associated I/O states.

The reason for the difference is, again, because of the heterogeneous characteristics of tactical tasks versus the homogeneous characteristics of most non-tactical systems.

As established in Eq. 9, the priority of thread P_i is dictated by the priority of its first task, q_{i1}. These characteristics show that the only dynamically changing priorities in the system are those associated with the "lead" task of each thread; thus, a simple queuing model can be structured to satisfy this scheduling requirement. Figure 5 illustrates a standard queue structure in which the tasks, q_{ij} (Figure 3), are randomly dispersed and are serviced by the executive according to their respective priorities. This queuing model will satisfy the task-scheduling requirements of our system, but will not provide the executive with an adequate mechanism to monitor the thread critical time periods for possible overrun conditions.

This dynamic process of time budget management is probably the greatest single difference between tactical and non-tactical computer systems. The tactical executive design must contain the capability to compensate automatically for as many perturbations to the processing norm as possible, while maintaining each critical thread period, whereas the typical non-tactical executive design logic usually will rely on an external operator to restore system integrity. The process of automatic time budget management contributes greatly to the sophistication of tactical executives, especially in the area of dynamic task queue maintenance.

An approach to provide an executive time-monitoring capability is to structure the basic queue with time bounds which correspond to the *critical thread periods*, Q_i, illustrated in Figure 3. The priorities of these time bounds could then be established according to the priorities of the threads they represent, P_i (Figure 3). All of the tasks associated with a thread, and consequently a thread critical period, would then be contained within the corresponding thread priority level in the queue, as shown in Figure 6. This structuring is possible because of the characteristics described by Eqs. 8 and 9. If we now associate an overrun time parameter with each thread level and with each task, it is possible to predict the probability of

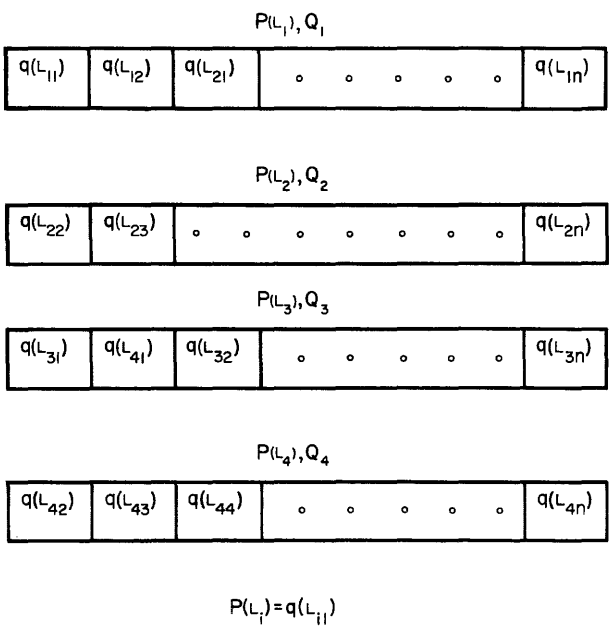


Figure 6—Multiple-level queue structure

achieving the required critical period constraint of Q_i (Eq. 7) and T_p (Eq. 3). If an overrun occurs at the thread level —i.e., Q_i is not satisfied—the only recourse is to transfer to some error-processing state. However, it is simple to predict the varying probability of achieving Q_i by carefully monitoring each intra-thread task for an overrun condition. If a timing problem should arise, the executive program has the capability to temporarily suspend the interleaved Q_i+_j tasks (Figure 6), which are processing within the $P(L_i)$ thread priority level, in favor of keeping the $P(L_i)$ tasks within their time constraint, Q_i. This is a simple process whereby the tasks, which are interleaved, are simply moved to their normal thread priority level, thereby allocating the entire critical period, Q_i, to q_i tasks only.

For example, task q_{41} (Figure 6) would be moved out of the thread critical period, Q_3, and into the thread critical period, Q_4, if task q_3, was in a time overrun condition, which jeopardized the completion of $P(L_3)$ tasks within the thread critical period, Q_3. This methodology is possible because of the common priority structure of these type systems, as described in the following inequality:

$$q(L_i) \geq q(L_i+j) \tag{10}$$

for a thread critical period, Q_i.

Inequality 10 states that interleaved tasks (q_{i+j}) have a priority less than or equal to non-interleaved tasks, (q_i), within the same thread critical time period (Q_i). This is likely to be the case, except in the rare instances where a high priority task may be dynamically interleaved into a thread period, in which case the high priority task would be processed in priority order within the thread and the lower priority non-interleaved tasks could overrun their critical thread period. This requires a tradeoff on the part of the system analyst/designer of the tactical priority structure to determine whether it is more important to satisfy a critical thread period or immediately process a high priority task.

Under certain circumstances, a complete thread of tasks may require immediate processing because of the arrival of some unpredictable high priority event. If the priority of this event is higher than the thread priority level of the currently processing task, the executive program will initiate a special suspension process called "preemption". This preemption process is not unlike a multiprogrammed non-tactical system's interrupt logic, except that tactical system preemption takes place at the thread level (i.e., an entire group of tasks is interrupted), while most non-tactical systems interrupt at the single task level. This thread level preemption evolves from the functional properties associated with a tactical thread. For example, if a currently processing thread's primary function was to load a launcher and fire a missile, and at the instant of load, the computer system, by virtue of some event, decided to suspend the thread, the preempted thread may actually be recalled to support the preempting event by reloading the launcher and firing at another target. Thread level preemption then requires that the tactical executive scheduling logic be capable of suspending and awakening multiple tasks simultaneously. It is intuitively obvious from the previous scheduling queue structure discussions that a preemption could occur as the direct result of (1) the current processing task requesting the scheduling of a higher-thread-level successor, or (2) an external interrupt from a decrementing clock or an input/output operation. The most frequent cause for preemption is the arrival of an external interrupt from another computer subsystem announcing "special-threat" target detections. This event arrival will cause any processing task, and its associated thread, to be suspended during normal executive interrupt processing, and the appropriate higher priority event processing thread will be placed into its appropriate priority position in the scheduling queue. The executive will then examine the scheduling queue in search of the highest priority pending task (which in most cases would be the suspended task). In this hypothetical case, however, the highest priority pending task is the new arrival. This special case causes the executive to preempt the previously interrupted thread/task and save all registers and volatile data-base contents. Processing control is then transferred to the new candidate. The executive then increases the priority of the preempted task to the highest within its predefined thread level. This procedure ensures that the preempted task is "awakened" prior to any other pending candidate selection in its (the preempted tasks) thread priority level.

This scheduling logic and queuing model enables the executive program to manage the critical processing periods, t_p (Eq. 3) and Q_i(Eq. 7): to provide instantaneous response to special high priority events while maintaining the system integrity; and to permit task interleaving between threads.

Let us now examine periodic task scheduling requirements which are represented by either of the following process initiation triggers:

$$t_i = t_1 + \sum_{i=1}^{j} \Delta t_i \tag{11}$$

or

$$t_i = t_l + \Delta t \tag{12}$$

Eq. 11 is the scheduling-time calculation for determining when to begin processing predictable periodic tasks. This is obvious because the term t_1 represents the first time the task was processed and Δt_i represents the fixed frequency; therefore, the next processing time will always be initiated some Δt factor after the first processing time. These types of periodic tasks are scheduled for processing in an identical manner to non-periodic tasks, as earlier described.

Eq. 12 represents the scheduling time required for unpredictable periodic tasks. This is evident because the term t_i is the *last* time the task was processed and Δt

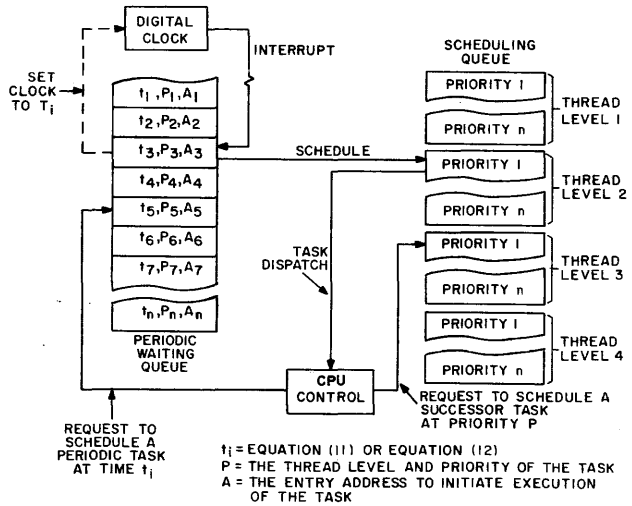


Figure 7—Executive scheduling philosophy

is a constant time increment, therefore in the event the last processing time t_i was triggered by a random event, this type of task would not always be scheduled in exact Δt intervals relative to the first scheduled time. As mentioned earlier, periodic tasks may be more critical than nonperiodic tasks, especially those represented by Eq. 12. This then requires that periodic tasks must begin processing in relation to their relative priority, when compared to all other system tasks. The executive program could satisfy this requirement by inserting all periodic tasks, which are ready for processing, per Eq. 11 or 12, into the scheduling queue, according to their thread level and priority. This technique will avoid having high priority pending tasks delayed because of lower priority periodic tasks instantly being processed upon achieving their respective scheduling times.

We can represent this design concept in Figure 7, where the "periodic waiting queue" is a "holding table" of unordered frequency dependent tasks awaiting their scheduling times, (t_i), as represented by Eqs. 11 or 12. Periodic tasks are selected from the "periodic waiting queue" and inserted into the scheduling queue according to their respective thread level and priority; i.e., they will compete for processing time with the entire set of system tasks.

CONCLUSIONS

The performance criteria for today's sophisticated tactical Command and Control systems imposes unprecedented requirements on the design of both the hardware and the software which control the systems. The most critical aspects include the time tolerances associated with the scheduling/dispatching and processing of tactical tasks, and the dynamic attribute of an ever changing, highly unpredictable tactical environment. The executive program, which is the nucleous of any tactical system, must be designed to operate not only in the classical non-tactical environment, but must additionally continually monitor the critical time periods associated with task group (threads) processing and automatically compensate, if possible, for any overruns. The executive scheduling and dispatching mechanism must also be sufficiently flexible to suspend and subsequently awaken groups of tasks in the event of unpredictable high priority event arrivals. These dynamic attributes of a tactical executive set it apart from the typical non-tactical executive, which operates in a well structured and fairly predictable environment, however it is obvious that many similarities do exist and in fact frequently outweigh the differences. Although most of the techniques discussed here are not new to the computer sciences, the method of implementation to solve the tactical problem is noteworthy. Most of the system characteristics described in this paper are representative of the U.S. Navy's AEGIS program presently being developed by RCA Corporation.

REFERENCES

1. Phillips, W. G., "Executive Program Scheduling for Large Command and Control Systems," RCA Reprint No. RE 18-1-6, *RCA Engineer Magazine*, Vol. 18, No. 1, pp. 55-59, July 1972.

PART II

METHODS AND APPLICATIONS

STRUCTURE - METHODS AND APPLICATIONS PROGRAM

Session	Chairman
Government Section	
Computers in the Congress	Baynard
Computers in the Elective Process	Lundell
Urban Services 1, 2	Blum
The State and Local Scene	
Five-Year Plans for Computers in State Government	Gentile
Computer Operations of State Agencies and Universities	Gentile
Metrication	Andrus
Simulation of International Relations	Draper
Installation Management Section	
The Law of Computers	
Regulation of the Computer/Communications Industry	Bigelow
Legal Protection for Software	Bigelow
Performance Evaluation and Measurement	Boehm
Software Products	
Status and Future of Software Products Worldwide	Goetz
Development of Generalized Software Products	Porter
Inhouse Training I	Tucker
Inhouse Training II	Savides
Inhouse Training III	Buerger
Inhouse Training IV	Tucker
Confidentiality and Security	Gosden
Four Major Reports on Privacy and Computers	Gosden
Security and Privacy, in Specific Computer Systems	Riley
Security and Privacy in Disaster	Steel
Data Integrity	Bryce
Economics and Remote Terminals	Printz
Industry Section	
Automotive	Grimes
Computers in Automotive Design and Manufacturing	Kuschnerus
Off Vehicle Diagnostics	Sweeney
Onboard Computers for Automobiles	Tyson
Automobiles, Computers, and the Consumer	Mainwaring
Manufacturing Automation	Anderson
Graphics Applications for the Garment Industry	Gilbert
Merchandising Section	
Retail Systems	
Point-Of-Sale Systems	Hampson
Data Processing Directions in the Retail Industry	Hampson
Advertising and Marketing	Gulotta
General Section	
Publishing and Knowledge Dissemination	
Publishing	Anzelmo
Knowledge Dissemination	Aines
Reliability for Integration into Human Affairs	Parker
Ordinary Mortals Using Computers Without Pain	
The Impact of Hand-Held Calculators	Williams
Voice Answerback Comes of Age	Fisher
Computers are for People	Groner
Computer Use Around The World	Gilchrist

The Methods and Applications portion of this volume was produced entirely by use of computerized text processing and photocomposition.

COMPUTERS IN THE CONGRESS

Ernest C. Baynard

"It must be remembered that there is nothing more difficult to plan, more doubtful of success, nor more dangerous to manage, than the creation of a new system. For the initiator has the enmity of all who would profit by the preservation of the old institutions and merely lukewarm defenders in those who would gain by the new ones".

--- Niccolò Machiavelli, *The Prince* (1513).

INTRODUCTION

Information is the lifeblood of the legislative process, yet Congress has not exploited computer technology to meet its needs. Present-day computer technology, applied to the legislative process, would have a dramatic impact upon the effectiveness of the Congress. Most importantly, computers applied to the budget and appropriation cycle would provide information concerning budget requests and the nature and effectiveness of Federal expenditures far beyond the capacity of traditional data processing procedures. There are other areas of computer application -- improving retrieval of reference and historical data -- an effective bill status system -- administrative improvements in the offices of Members of Congress. Applications in these areas that interface with the *substantive legislative process* is where Congress will obtain the true advantage of the computer.

Congressional payrolls, equipment inventories, and other "housekeeping" applications of computers have come to Capitol Hill in recent years. There have long been discussions of broad application of computers to the Congress. However, the first formal effort to bring this about began in 1965, when Congress established the Joint Committee on the Organization of the Congress to study means by which its operations could be improved. More effective legislative control over the budget and Federal expenditures was a priority item of the Committee.

Congressman Jack Brooks, Democrat of Texas, the leading congressional proponent of effective and efficient use of computers, was instrumental in adding language to the committee's report that recommended the development of a sophisticated computer system to support the budget cycle in both the Executive and Legislative Branches of the Government. Legislation implementing the Committee's recommendations was approved by the Senate, but the House failed to act and the Committee was dissolved in 1968. In 1969 January, Brooks introduced legislation aimed at getting congressional computer operations off the ground. But by then the House Rules Committee, which must give approval to legislation to reach the Floor for debate, had assumed responsibility for congressional reorganization and considered computers as falling within its jurisdiction. As a result, this legislation, although approved by the House Government Operations Committee, did not get to the Floor of the House for consideration.

Fortunately, however, the Rules Committee bill, which Congress finally adopted as the Reorganization Act of 1970, included the language concerning computers which Brooks had previously recommended. Thus an affirmative mandate was established calling for cooperation between the Executive Branch (represented by the Office of Management and Budget) and the Legislative Branch (represented by the Comptroller General), in the development of a computer system to support the budget and appropriations cycle.

However, progress has been painfully slow and at times difficult, if not impossible, to observe. The decisive steps necessary to exploit the power of computers in the Congress have not been taken. All the necessary ingredients are available. The resources, the expertise in both computers and in the legislative process, are available for the asking. All that seems to be lacking is initiative on the part of key officials in the support units of the Congress, and a comprehensive plan detailing the flows of data that are of value to the Congress and the extent these data are susceptible to modern data processing techniques. There is no plan in existence outlining who in the Congress should do what to make the computer systems Congress needs a reality.

There is an equally important problem -- the need to describe this plan in language that people who are not computer technicians, people like Congressmen and their staffs, will readily understand. This is no altruistic requirement. Congressmen are zealous of their prerogatives. "Big" money is involved. Members will want an explanation of the product before Congress writes the checks.

Accordingly, this paper has two purposes. First, it is an attempt to explain computers and their relationship to Congress in simple language that laymen might understand. Second, it is to provide a rough "discussion draft" that could be altered or refined by computer experts and those involved in the legislative process, or, in fact, by anyone who might wish to make a contribution. Hopefully this might ultimately lead to development of a coordinated and comprehensive approach to congressional computer system development.

THE COMPUTER

Computers -- or, in more precise terms, electronic data processors -- as their name implies, process data. Information is some recallable or communicable tidbit of data that is oriented to the needs of a specific person or persons, for an identifiable purpose, at a specific time and location.

A computer is a very highspeed adding machine, commonly using binary (base 2) numbers instead of decimal (base 10) numbers. 1 is 1, 2 is 10, 3 is 11, and 4 is 100, etc., etc. Using binary numbers makes it easy for computers to add and subtract. To multiply can be to add repeatedly. To divide can be simply to subtract repeatedly.

The potential of the computer flows from this highspeed computational capacity, coupled with the ability to compare numbers according to some predetermined order or sequence.

These two seemingly limited functions, computing and comparing, can theoretically be structured into any nonintellectual thought process of the human mind. A very simple example is the check reconciliation process, known to practically everyone. The computer, by a series of programmed instructions, calls each check up for examination. After feeding both the information on the check stubs and the returned checks into the computer via cards or tapes, the computer compares the number of the check stub with the number on the check. The computer scans the numbers. If the first number is less than the number on the check, the program "loops" back to repeat the comparison on the next number, and so on until the numbers on the stub and the check match. The amount of the check is then, by a simple follow-on computer operation, subtracted from the balance and the next check number is brought up for comparison.

This basic concept is simple. The structuring of "software" -- the body of instructions and codes that direct the computer -- almost always becomes a complex, tedious, time-consuming, and costly operation, with the quality of the result depending upon the capability of the programmer.

Computers can process data into information of a quality and at speeds impossible to obtain from traditional data processing methods. A medium-sized computer reconciles the Government's checkbook on a current basis -- a task previously employing hundreds of clerks who were usually months behind. Larger computer systems, properly designed and coordinated with the sources of data of value to the Congress, can be applied with equal effectiveness to key facets of the legislative process.

Before discussing the use of computers in Congress, certain fundamental policies must be recognized.

POLICIES PROTECTING THE INTEGRITY OF THE LEGISLATIVE SYSTEM MUST GOVERN THE APPLICATION OF COMPUTERS IN CONGRESS

- The legislative process is a direct reflection of our democratic form of government. No computer should be used in any way to alter, or in any substantive sense compromise, the rules of the House or Senate or the basic parliamentary procedures of the Congress. Computer concepts must be moulded to meet the needs of the Congress. The Congress must not be subordinated to computers.

- Computers must be used to improve the quality of information available to Members and enhance their decisionmaking capability. Through improvements in the quality of information available to Members and committees of Congress, higher quality decisions, directly benefiting the nation, can be made by the individual Members. Computer applications that might dilute the decisionmaking power of the individual Member or interfere with his representative capacity cannot be tolerated.

- In the design and development of computer systems for use in the Congress, no arbitrary restrictions or restraints affecting the ultimate availability of data to all Members of Congress should be allowed.

- The committees of the Congress must control the determination of their informational requirements. There are only three acceptable responses to the request of a committee of the Congress for data processing capacity. First, "Yes, the system now in use has the capacity"; Second, "Sorry, the capacity requested is beyond the state of the art"; Third, "Sorry, the system now in use does not have the capacity, but could be improved or extended at the cost of 'X' dollars. If you can obtain this additional funding for us, the data you request will be provided".

  Under this approach, the capacity of computer systems available to congressional committees will depend directly upon the state of the art and the ability of the committee requiring information to obtain the resources needed to provide the capacity.

  It is unrealistic to assume that any committee having policy control over any aspect of computer system development should limit or deny any request of any other congressional committee.

- Computer systems will be costly, but if system development is properly coordinated and carried out, benefits from computer use will be well worth the cost. No computer system should be introduced in Congress without the appropriate Committee of Congress first requiring a full explanation of the computer output in language the average Member can easily understand. Only on the basis of easily understandable explanations of the computer capacity to be provided should the Congress agree to the substantial expenditures inherent to introduction of computers into the legislative process.

- Extensive periods of time are often required to develop computer systems. It is therefore vital to avoid distortion of the effort to get quick results in order to justify the expenditures, when the distortion compromises the ultimate quality of the system.

- It is also fundamental that the Congress, in the procurement of computer systems, should follow the same policy considerations and businesslike management practices that have been laid down for the Executive Branch under Public Law 89-306. Of particular importance is the mandatory constraint that hardware should be procured under competitive conditions.

APPLICATIONS OF COMPUTERS IN THE CONGRESS FALL INTO SIX CATEGORIES

Computers Can Provide the Members and Committees of the Congress with Better Data about the Budget and Appropriations Process

Attempt to obtain data concerning the funds available during this fiscal year for any program in the Executive Branch. A response could take days, or even weeks. Most probably the materials obtained would be very general and limited in amount.

Broaden your request to include the amount in the budget for the coming fiscal year covering this specific program. Or ask about the funds expended in prior years, the number of personnel employed, the results achieved, the statutory authority, and other rudimentary data regarding the program. You will find that weeks can elapse before any material is received.

On a more general basis, make a request for the total sum being expended for some particular category of expenditures, such as education. Request data about specific types of expenditures, such as travel on a governmentwide basis. There will be a similar delay, if, in fact, it is possible to obtain the data at *any* time in *any* form.

Relatively simple requests for data concerning the budget, just as outlined above, are preliminary to any evaluation regarding the need for, or efficiency of, any Federal program.

Through the use of computer techniques, a broad variety of data would be readily available within a period of a few hours concerning any facet of government operations. The day a new budget is submitted to the Congress, the committees and individual Members could obtain data concerning those facets of government operations, as reflected in the budget, of specific interest to them. They would not have to grapple hopelessly with the telephone-book-sized Budget Appendix.

Accompanying the basic fiscal data (what agency is to obtain how much for what purpose) would be the ability to obtain a limited, but important, number of basic evaluations. As an example, a Government Operations subcommittee could obtain, on the day the budget was submitted to the Congress, a printout of the funds allocated to each program in each of the departments and agencies under the subcommittee's jurisdiction. These funds could be broken down under the traditional object classification, indicating the amount to be spent for travel, personnel, etc., etc. Accompanying this breakdown could be a comparison with the funds available for these programs during the prior fiscal year. Immediately available would be data as to what programs involved increased expenditures, the areas of activity in which the increased expenditures would fall, and the percentage differences from prior years.

With this data could be a statement of statutory authority corresponding to the specific programs, a present and projected breakdown of personnel by GS rating, together with the ability to develop input-output indexes to indicate changes in operational efficiency of programs as a whole. The same data, but in differing formats to reflect the specific requirements of the legislative and appropriations committees, could be available within the same time frames.

With data of this nature readily available, it would be possible for Congress to significantly improve its control over Federal Government fiscal operations. On a "follow-on" basis, in the years to come, advanced data processing techniques could be used to develop a sophisticated analytical capacity that would assist Congress in evaluating the justifications for expenditures involving the various programs under way in the Federal Government. In the technical area, such evaluations would be a major responsibility of the Office of Technology Assessment which the Congress has recently established as a part of the Legislative Branch.

In very general terms, evaluations or assessments would be made by using historically-oriented reference or opinion data to evaluate and analyze the budget data. *Example:* Suppose that a NASA budget request for $10 billion were before Congress to send spaceships to Mars to collect rock samples. Among other things, the Congress would want to have expert opinion data reflecting the opinions of geologists as to what might be learned from an examination of the the rocks. The Congress might well wish to evaluate historical and reference data as to the extent that knowledge had been gained when rocks had been collected on the moon.

Through Use of Computers, Congress Can Obtain More Effective Access to the Vast Store of Historical and Reference Data Essential to the Evaluation of Legislative Proposals and the Budgetary Requests of the Executive Branch

Even assuming that you have reliable, current fiscal data concerning an identifiable segment of a Federal program, to assess ongoing and proposed new programs, and the efficiency of the operations that are involved, requires a research capability to obtain reference or historical data on the basis of which judgments can be made.

The contemporary testimony of witnesses before Congressional committees, at least in the form that it has been presented traditionally, is not susceptible to computer techniques. However, the *prior* testimony of witnesses and other experts, and the almost infinite store of knowledge in the Library of Congress and other depositories is "on call". It is needed by the Congress to properly evaluate the budget and other matters flowing through the legislative process.

The extraction of historical and reference data is a difficult, time-consuming process now. Library of Congress researchers, following traditional methods of past decades, examine a series of bibliography cards to locate data pertinent to a request.

In contrast to this inadequate approach, computers can be employed. The Library of Congress, without adequate funds, has been working in this area for years. Modern library science techniques aid the researcher in locating the desired data. Library bibliography cards can be computerized so that key words identifying the data requested will be matched up with synopses of books and periodicals and other materials, allowing for location of the data in minutes, when manual search would have taken hours or days.

Computers do not like unstructured data. Their application to historical or reference material, such as in the Library of Congress, will be a difficult and costly job. But through their use, substantial improvement in access to this type of material can be obtained.

Computers Can Be Used to Develop an Effective Bill Status System for the Congress

At present, a Member's office or a committee must make at least five or six calls to determine authoritatively the status of a legislative proposal. This is a time-consuming, error-provoking process. A sophisticated bill status system, utilizing computers, can provide a status report on all legislation with one telephone call to a centralized bill status operational center. A computer system operator, given either the bill number or the subject matter of the legislation of interest, could query the system and provide a short response. *Example:* "H. R. 1234 is pending before the House Armed Services Committee; no action is scheduled at this time". If the Member or staff member needed additional information, the operator could obtain a "printout" from the computer system of all information relative to that proposal. This would include the text of the bill, its author, action taken, references to the reports of various agencies, and other data concerning the proposal that an individual would need to make at least an initial assessment of the legislation. Requested printouts would be extracted from the system in the computer room and delivered to the Member's office.

After the system had been established, terminals could be installed at various locations on the Hill where printouts could be obtained. Ultimately, should the House or the Senate feel that the cost could be justified, terminals could be installed in each Member's office.

Computers Can Also Be of Some Possible Immediate Help in the Operation of Individual Members' Offices

In discussing computers in the offices of the individual Members of Congress, a clear distinction should be made between -- the overall development of computer systems that will benefit all Members of Congress -- the linkage between these systems to the offices of individual Members for purposes of transmitting data, and -- computer systems within the office to process data exclusively for the individual Member.

It is academic to discuss providing individual Members with information as to the (1) budget and appropriations cycle, (2) historically-oriented research or reference data from the Library of Congress, or (3) data from a bill status system, until these basic systems are developed and are in operation.

Aside from the installation of terminal or computer units in individual Members' offices to satisfy their needs for data from the systems enumerated above, computers don't offer any broad or immediate potential to the individual Member. However, there are some applications that would be helpful.

There is a present possibility that a computerized filing system could act as an adjunct in the development and maintenance of the Member's constituent mailing lists and also form the "base" for ultimately bringing into his office direct computer access capability with the systems referred to above.

Using a computerized filing system, a secretary would type, on an electric typewriter linked with a computer, the name of a constituent writing or contacting the Member. A "write-out" would immediately show whether the individual had previously written or contacted the Member, the subject matter of the letter, and the date of the response.

Incident to the final answering of the constituent's letter or request, in lieu of filing a copy of the response or noting the response on a file card, the secretary, using the same method just described, would add the identifying information as to the letter to the data stored in the computer for future access. The name could then be added to the permanent mailing list under subject matter categories, or could be printed out for whatever purposes the Member might require.

Computers Can Be Used to Improve the Communications Linkage Between Members and Their Constituents

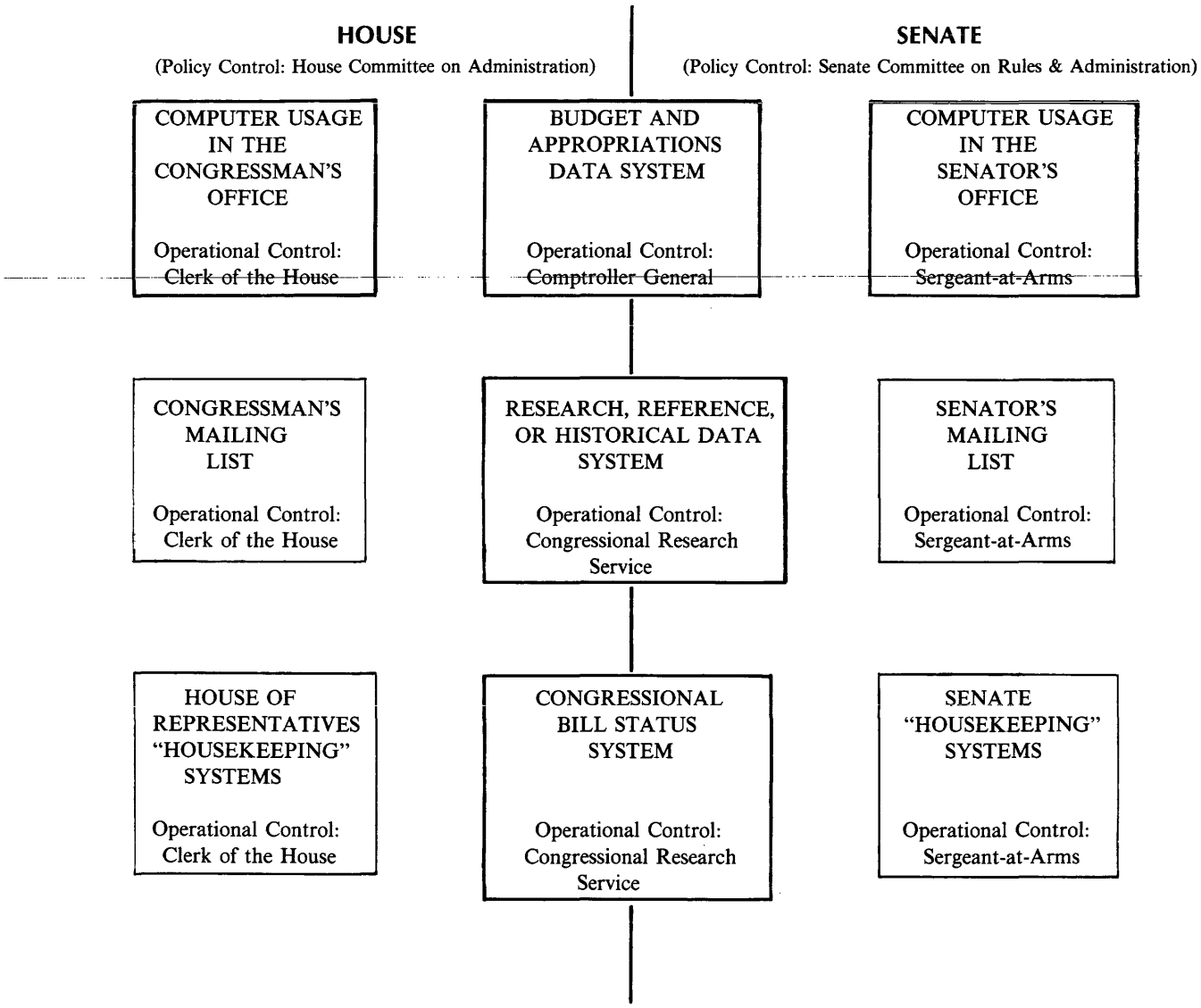
Computers can be used to vastly improve the maintenance of mailing lists and the processing of correspondence with constituents, and this is already under way in both the House and the Senate.

Computers Are Needed to Maintain the Housekeeping Responsibilities of the House and the Senate

As in all other organizations, computers are effective in the development and the maintaining of payrolls, inventory systems, and other "housekeeping" activities. However, application of data processing techniques to these areas of activity is not as relatively important as those applications interfacing with the substantive legislative process, which these do not. Unlike the other applications already listed, there is no need for these systems to be compatible or uniform in any way between the House and the Senate. There is no need for compatibility between House and Senate housekeeping computers. There is no need for them to react. In fact, there may be an advantage in maintaining a lack of compatibility to avoid any possible compromise in the independent status of the House and Senate.

All meaningful computer applications in Congress will fall logically within one of the general categories referred to above, and depicted on the facing chart.

STRUCTURE OF PROPOSED CONGRESSIONAL COMPUTER SYSTEM



Notes:

1. Computer systems depicted in boxes with heavy outlines must be compatible because they support or interface with the substantive legislative process. Data from the Office of Management and Budget (located in the Executive Branch), the General Accounting Office (headed by the Comptroller General), the Library of Congress, the congressional bill status system, and any computer capacity provided individual Members' offices, must be compatible so that these computer systems can "talk to each other" and their data can be blended for evaluation purposes.

2. The jurisdiction of the Joint Committee on Congressional Operations would emphasize the overall development of information systems for the Congress as a whole and, *to the extent necessary,* provide the coordinating linkage for the development and maintenance of the unified systems which are needed by both the House and the Senate, shown in the middle column of the diagram.

THERE ARE SERIOUS LIMITATIONS ON THE TYPES OF DATA SUBJECT TO COMPUTERIZATION, AND THESE LIMITATIONS MUST BE UNDERSTOOD IF OPTIMUM USE OF COMPUTERS IS TO BE ATTAINED IN THE CONGRESS

Computers are like some cats. They are finicky eaters. Computers will not accept just any data that comes along. And even if "just any data" is somehow fed into the memory of a computer, this does not mean that access to that data is possible. And even if the data can be gotten out of the computer, this does not mean it will have any informational value. In fact, computer output processed from poor data, or by inadequate computer programs, can be dangerously misleading. Data that is input to a computer must be structured, identified, and classified in advance, in exact terms of its prospective use, or else the computer simply refuses to work.

The information on a gasoline pump in a filling station as to the price and amount of fuel placed in an automobile tank is a good example of structured data -- data that is identifiable and predictable in terms of form and description. These characteristics of the data remain stable although the data itself may change. On the gasoline pump, which is in essence a very simple computer, experience has shown that information as to the quantity of gasoline and the corresponding price of that quantity will be of value. The data as to these items will be constantly changing and will recur every time gasoline flows through the pump.

In contrast, there is nonrecurring, unstructured data, such as that generally found in testimony before congressional committees and in the vast collection of books, newspapers, and periodicals stored in the Library of Congress. While computers love structured and recurring data, the storage and retrieval of nonrecurring data present formidable problems.

While computer science is making inroads into the use of computers to retrieve data from loosely structured data bases, generally, as the flexibility of data structure increases, the ability to select specific data in the data base decreases.

Generally, data is either structured under some access concept, or is only subject to some library science-type indexing, where a trained researcher must take over from the computer and make the final identification of the data desired. Because of these limitations, only certain flows of data in the Congress are directly subject to computer techniques. They are vital to the legislative process but constitute only a limited part of the total data flow in Congress.

The knowledge of every Member, staff member, committee witness, visitor -- the contents of every newspaper and book that is delivered to the Congress -- every broadcast or telecast that is heard or seen by anyone interfacing with the legislative process -- are all data sources that potentially affect the progress of legislation through Congress. Computerization of this entire flow is impossible. Our interest is in principal data flows, having significant impact on the substantive process, that fall within the perimeter of practical computer application. Note that a principal flow of data into the legislative process, the testimony of witnesses appearing before committees, is not generally subject to computer techniques.

Material submitted in oral testimony or for the record can be similar and related to budgetary and appropriations data received from the Executive Branch. Possibly future witnesses will offer material in computerized form that is compatible with and can be merged and evaluated on congressional computers, but in most instances such data will be predominantly current, unstructured, or reference-type. *Example:* "I am against the SST because .....". Thus the testimony of congressional witnesses, although a principal informational flow into the Congress, is of no immediate concern in applying computers to the legislative process.

To the Congress, (1) budget data from the Executive Branch, (2) historical or reference-oriented data, such as from the Library of Congress, and (3) a bill status system, constitute the principal flows of data essential to the legislative process subject to meaningful improvement through computer techniques. The data flows on a functional basis correspond with the principal computer modules in a congressional computer system as previously described.

THE PRINCIPAL DATA FLOWS IN THE CONGRESS THAT ARE SUBJECT TO COMPUTERIZATION

The Budget, As Prepared in the Executive Branch, Is the Principal and Most Important Data Flow In the Congress

In 1921, Congress delegated responsibility for the preparation of the budget to the Executive Branch. Each year, the Office of Management and Budget submits to the Congress the President's proposed budget, prepared under his direction on the basis of data supplied to the OMB by the 69 departments and agencies in the Executive Branch.

This flow of fiscal data is most important for two reasons. First, money is involved. Money is always important and achieves a top priority in its own right. In practical terms, the appropriation of funds to carry on the functions of government is the most fundamental aspect of the legislative process. Although it may not be the soul, surely appropriating money is the heart of the legislative process.

The second reason that budget and appropriations data is of prime importance to the application of computers in Congress is that the budget items may be the most extensive and most acceptable "common denominator" needed to unify the various computer applications Congress may require. The basic architecture of a computer system must have a backbone. It is doubtful whether a subject matter index (as would be the backbone of a bill status system) would be as satisfactory as the individual seven to eight thousand budget or cost elements in the OMB's proposed budget and appropriations system.

The budget is a prime candidate for a computer application. The structured fiscal data is highly susceptible to computers. The benefits to Congress are formidable. The computer gives Congress a real opportunity to regain fiscal control over the Federal Government after having delegated the budget function to the Executive Branch in 1921. Such a system could be the keystone of the legislative process, and could with logic and reason be ranked as the most important computer system in the nation.

Historical Material, Such as That Found in the Library of Congress, Is Also a Principal Source of Data Relative to the Substantive Legislative Process

Unstructured, historical, or reference data that Congress might need cannot be identified in prospective terms and made available systematically in anticipation of need. Data does not become information, and therefore of value, until related to specific need. And legislative informational needs, as in most endeavors, are for the most part unpredictable. However, Congress must have means to acquire individual items of data concerning practically any facet of human endeavor that might relate to or be pertinent to matters under consideration in the legislative process.

The Library of Congress, the Executive agencies -- the entire outside world -- have vast stores of data of this type. Because the need for most reference material cannot be anticipated, data from historical or reference sources must generally be requested. The sum total of these requests constitutes a second vital flow of data into the legislative process. This data, served up "when needed", is what judgments are based on. Nonrecurring data of past events, opinions, and prior decisions are one of the most important sources of information in the legislative process.

Although access of this data for the most part is not subject to direct computer application, the library science and bibliographic data used to identify and locate the desirable material can be processed by computers. Synopses of certain classes of reference material, such as normally accompanies medical and other scientific treatises, can be placed in computer memory and retrieved on a subject matter or "key word" basis.

Other computer techniques with significant potential in the storage and retrieval of nonrecurring data must be exploited. Note that the structured budget data and the contents of a congressional bill status system become reference material after the fiscal year or a particular Congress come to an end. As time passes, reference or historical data on past budgets and prior legislation should be subject to immediate recall.

The Congressional Research Service must provide Congress the highest level of computer capability, recognizing that this is a difficult area for computer applications.

The Flow of Data As to the Status of Bills and Other Proposals in the Legislative Process Is Also of Vital Concern to the Congress

The nation's vast telephone system would be useless without telephone books. Likewise, the legislative process, to operate most effectively, must provide individual Members of the Congress and the committees with a flow of data as to the status of the various legislative actions under consideration.

The tendency to underestimate the complexity and the cost of this system should be recognized. Many state legislatures have "bill status" systems. OMB has recently developed a relatively simple system to follow legislation affecting the President's program. But these systems have little in common with the system Congress requires. Experts on the legislative process must "think out" every step of the process and identify in detail the ultimate product or requirement the computer is to provide.

Design of a bill status system requires much preliminary work on the part of experts in the legislative process even before computer technicians are involved. A standard and improved subject matter index applicable to the Congressional Record and the U. S. Code must be completed. Fortunately, the Congressional Research Service has done considerable work on a project of this kind.

A number of other exceedingly delicate problems must also be worked out so that the system will be of meaningful value to Congress. There must be an effective way to handle "clean bills", that is, measures that are introduced after committee consideration has been completed, to reflect, in "clean" draft form, the amendments that the committee recommends to the House or Senate. If a Member should query a bill status system concerning "H. R. 1234" and be advised that the bill had been indefinitely postponed, he would be rightfully concerned to learn at a later date that a "clean bill" had been introduced as "H. R. 3456" and had gained the approval of the Congress.

As computers don't like undefined, unstructured data, every bill that goes into the system must be analyzed and given a title or reference in accord with the system's subject matter index. This takes time. Yet the system must be immediately responsive. Think of the logjam at the beginning of a Congress when hundreds of proposals are introduced at one time. There is no requirement, nor is there likely to be one, that Members conform the titles of bills they introduce with some standard subject matter index. To fit each bill into a standard index will require individual evaluation by the Congressional Research Service. This, and other analyses that must be made if the system is to be of real value, will require considerable time for much of the legislation introduced. An apparent solution is a temporary system based on bill number and title. Then, when action is scheduled on the bill, it could be placed in the permanent data system. The delay between introduction and action would give the Congressional Research Service time to analyze the bill and to structure its contents in data forms acceptable to a computer.

There are countless other problems that must be anticipated and researched before a bill status system concept should be allowed near a computer.

THERE ARE A NUMBER OF SYSTEMS DESIGN CONCEPTS AND OTHER IMPORTANT CONSIDERATIONS THAT MUST BE TAKEN INTO ACCOUNT IF THE CONGRESS IS TO UTILIZE COMPUTERS EFFECTIVELY

Computer Compatibility Is Essential to the Efficient, Effective Use of Computers in Congress

Computer usage in Congress will evolve over a long period of time. It isn't practical or even possible to develop in one cycle a comprehensive system covering all possible computer applications. What Congress must do is develop computer systems on a modular basis with the view that systems that *relate to the substantive legislative process* will ultimately interface or even merge. Congressional computer development will be like planting plugs of zoysia grass.

The interests of the Congress demand that these modules be capable of mutual interaction at some future date. Computers supporting the substantive legislative process must be able to "talk to each other" through data exchange, as data systems grow and the data they maintain begins to intermingle with data in other legislative systems. This means that congressional computers in both the House and Senate that support or potentially interface with the *substantive legislative process* must be compatible between themselves and with the basic budget systems maintained by the Office of Management and Budget. Data generated at various points within the government must be merged and evaluated, for understanding by all concerned.

There must be uniformity. Otherwise, computers will not work. If the Senate were to develop a system that stated the date as "3 November 1969", the House were to use "11/3/69", and the Executive branch were to use "November 3, 1969", hopeless confusion would be the computer output rather than valuable information Congress needs.

The development and maintenance of a broad spectrum of classifications, standards, and conventions essential to an acceptable level of compatibility will be a difficult, time-consuming and permanent job. Some people say it is impossible. Others suggest that it is not necessary. But they are wrong. Compatibility is the key to effective use of computers in Congress, and compatibility requires a unified approach *at the operational level* in the development of initial systems Congress urgently requires. A unified approach to the development of computer systems directly relating to the substantive legislative process must either be the cardinal rule, or you can virtually write the future effectiveness of Congress "off the books".

During the Design Stage, the Architecture of Congressional Computer Systems Must Be Tailored To Meet the Needs of the Committees

Design of effective computer systems for the Congress does not allow all 535 Members of the House and Senate a choice in system structure or data format. Computer systems design must reflect the same considerations that led Congress to develop the committee system, under which the organizational structures of the House and Senate allow special attention to be given specific matters of legislative interest.

Variations in system output, such as the format of the data, must be kept within reasonable bounds. The requirements of Members must be placed in logical categories in the same manner that their interests are categorized under the committee system. This does not compromise the interest of the individual Member. The sum total of committee requirements for data is essentially the same as the collective requirements of the Members. Once the systems are in operation, the data would be available to individual Members under whatever procedures and policies the House and Senate may adopt.

The requirements of the committees during the systems design stage are the immediate goal of any congressional computer effort. Responding to this more specific objective will be difficult enough, even considering duplication in requirements among the committees.

The House has 21 so-called "standing" or regular committees, the Senate has 17. Apart from the rules and administration committees, with no extensive interface with the substantive legislative process, the congressional committees fall into these four categories:

- **The Revenue Committees**

  The House Ways and Means and Senate Finance Committees raise revenue. They are interested primarily in a data flow from the Office of Management and Budget as to the fiscal condition of the government and revenue estimates of the funds that will be paid into the Treasury under tax laws now in effect. Most of this is heavily structured, budget-type data.

- **The Appropriations Committees**

  The House and Senate Appropriations Committees are interested in structured budget data from OMB in the traditional object classification format. In addition, these committees are interested in related support data justifying an expenditure for the particular item at the level requested, which is a combination of structured budget data and reference or historical data.

- **The Legislative Committees**

  The various legislative committees, with subject-matter jurisdiction over varying categories of operations such as "Armed Services", "Labor", "Commerce", etc., deal with the authorization of programs. Although requiring structural fiscal data, their primary interest is in contemporary, reference, and historical data.

- **The "Oversight" Committees**

  The House and Senate Government Operations Committees have responsibility to audit and evaluate the economy and efficiency of government, i.e., follow the appropriated dollar to see that it is properly expended in accordance with the legislation authorizing the program. These committees have a primary need for fiscal data, but in a different format than that presented in the budget and used by the Appropriations Committees. The primary need is for Federal expenditures set forth in a program format.

Thus the data requirements of all congressional committees *subject to computer techniques* can be met by three basic systems: (1) a system providing information as to the budget and appropriations cycle, with the capability for formatting or arranging the data in a manner consistent with the different needs of the four types of committees; (2) a system encompassing the latest advances in library science computer techniques to provide the most effective retrieval capability for historical and other unstructured data that might be stored in the Library of Congress or any other depository available to the Congress; and, (3) a bill status system. In other words, the systems previously described.

The design of these three basic systems, with data structures and formats required to meet the needs of these four basic types of congressional committees, is the immediate goal. This approach reduces the congressional computer system design problem to a manageable size.

It Is Important, As Far As the Use of Computers in the Congress is Concerned, to Distinguish Between the Acquisition of Data by Computers and the Follow-On Analysis of This Data Using Computer Techniques

The application of computers in Congress will be a long process that will evolve continually for decades. Use of a modern cost-benefit analysis technique in the Congress is a popular concept -- and quite reasonably a controversial subject. In practical terms, the broad and meaningful use of such techniques in the Congress is a long way off. *In any event, it will depend upon Congress' first acquiring the data in the proper quantity, quality, form and structure to analyze.*

Meanwhile, broad access to fiscal data alone, without any accompanying analytical capability, would be of the greatest value. Just a simple inventory of the money. Just a simple statement of "who" is getting "how much" for "what" would be "hawg heaven" for most Congressional committees, in contrast to what is presently available.

Congress must not overlook the data to be analyzed in grasping for analytical capability.

The Most Effective and Efficient Computer System Design Comes from Blending the Technical Expertise of the Computer Expert with the Knowledge of Those Familiar with the Field of Application and Those Who Will Use the System

While there are always exceptions, computer systems must be tailormade. Few applications programs can be purchased "off the shelf". The system design must reflect the highest level of understanding of the substantive process to which the computer is to be applied. Good system design technique requires a "cycling" or repeated exchange between computer experts and the experts in the field of substantive application. It is essential that prospective users demand a complete explanation and data format description of a proposed system before implementation begins.

This is particularly important if computerization of obsolete systems is to be avoided. Present noncomputer data systems often reflect the limitations of their time, and some of these presently in use in Congress date back to George Washington's administration. These limitations need not be retained after computers are introduced. *Example:* There is no need to limit the requirement for an update of the budget to July 1 each year as provided in the Reorganization Act. Computers can provide for a constantly updated budget, reflecting changes as they are made.

Only through the blending of computer expertise with that relating to the proposed application can such limitations on data flow be identified and discarded. Only by such discussions -- in a "give and take" environment -- can an optimum system be devised. Such discussions are essential before *any* system is slated for implementation.

THERE MUST BE AN AGREED-UPON PLAN DELINEATING RESPONSIBILITY FOR THE ORDERLY AND EFFECTIVE INTRODUCTION AND USE OF COMPUTERS IN THE CONGRESS

Applying Computers to Congress Immediately Becomes a Question of Who Does What

There are a number of potential players in the congressional computer game, each with different roles. It is essential that the fundamental question "who does what?" be settled in advance of computer system development. Otherwise, the success of the overall effort will be endangered from even an additional source.

All parties potentially involved in computer use in Congress can be set forth in three categories. First, the users of the system, that is, the committees and the individual Members. Second, those who will have policy control over system development and use. Third, those who will have operational responsibility for the development, maintenance, and continual improvement of computer systems.

At this point, a traditional philosophy of congressional operations comes into play. Congressional committees do not become involved in the routine operation of the various systems and services that support the legislative process and the Congress in general. *Example:* The staff of the Joint Committee on Printing does not directly manage the Government Printing Office, but rather asserts policy control over those officials with this responsibility.

The advantage of this policy is that it will allow the congressional committees to maintain the necessary independent "oversight" and policy control. If the committee were to become involved in the day-to-day operations through the use of its staff, then the committee would have no "leverage" in maintaining "oversight" and policy control over the operations.

In keeping with this policy, congressional committees should not become involved in actual computer system design and operations, but should remain aloof. Only thus can the proper policy control and the essential "oversight" function be maintained on a viable basis.

The House and Senate Committees with Administrative Jurisdiction Have Primary Policymaking Control Over Computer System Development and Maintenance in Their Respective Houses

In keeping with the policy of avoiding change or distortion of the present legislative or parliamentary system, it is logical that the House Committee on Administration and the Senate Committee on Rules and Administration maintain jurisdiction over computer system development and maintenance in their respective houses. . However, the emphasis on the oversight and policy control these two committees assert should be on the final product the committees, the individual Members, and other units of the Congress require. Both of these administrative committees should determine informational requirements and see that whatever operational unit has responsibility for providing these requirements delivers "the goods".

This approach allows the subordinate operational unit in the Congress with jurisdiction over development and maintenance of a particular system to "merge" the requirements of the House and Senate into one system that will provide for the needs of both. Not only will this allow for compatibility in computers between the House, the Senate, and the Executive Branch, but it will also avoid costly and unnecessary duplication in computer systems between the House and the Senate.

There is ample precedent. All major congressional subordinate support units -- that is, the General Accounting Office, the Library of Congress, and the Government Printing Office -- have always operated under this principle. Both the House and the Senate levy requirements against these units in terms of the product required. Neither the House nor the Senate require some unique auditing approach, library reference system, or printing classification that forces a division or duplication of operations.

The same fundamental policy must apply to computers. With the House and the Senate administrative committees saying what the two Houses of Congress want, rather than how to meet the requirements, the operational units can merge these needs into a unified compatible system.

The Joint Committee on Congressional Operations Should Undertake Long-Range Policy-Oriented Studies of the Informational Needs of Congress As a Whole

A vast improvement in basic data as to the budget and appropriations cycle is within our grasp. With additional funds, limited improvements can be made in the reference and research activities of the Library of Congress. An effective bill status system can be developed within a reasonable time period -- assuming the magnitude of the task is recognized and proper discipline is kept during development.

The advantages of this data acquisition capacity are small compared to those that will flow from the use of advanced computer techniques in the simulation and analysis of legislative proposals and appropriation requests, and in the audit of government operations. Such sophisticated systems cannot be developed overnight. They must evolve from the more rudimentary computer capacity discussed here. Painstaking analysis must be made of the overall informational requirements of Congress as a prelude to the development of more sophisticated computer capacity.

The Joint Committee on Congressional Operations is ideally suited to carry on such studies and analyses. It has no legislative authority, but no other committee of the Congress has comparable jurisdiction regarding congressional operations. The Joint Committee can operate effectively in this area without concern that its work impinges upon the jurisdiction of other congressional committees. Its reports need not and undoubtedly will not always reflect the optimum approach Congress should follow, but they can provide the essential discussion base, for those who might disagree, to offer affirmative and constructive criticism.

In a sense, the overall application of computers to the Congress requires "discussion drafts" of proposed computer system capacity. The Joint Committee offers the most comprehensive jurisdiction for the development of these studies.

The Comptroller General Has the Responsibility to Establish the Essential Classifications and Other Standards Necessary to Provide the Government with a Computer System to Support the Budget and Appropriations Cycle

The Legislative Reorganization Act of 1970 requires the Comptroller General to cooperate with the Secretary of the Treasury and the Director of the Office of Management and Budget to "develop, establish, and maintain standard classifications of programs, activities, receipts, and expenditures ..." necessary for the development of a computer system to support the budget and appropriations cycle. This delegation of authority extends to the Comptroller General the responsibility to develop and maintain a compatibility "linkage" between the various departments and agencies in the Executive Branch and the Office of Management and Budget. While primary responsibility for the development of a computer system to support the budget and appropriations cycle rests with the Office of Management and Budget, the act provides for the Comptroller General's "cooperation" in this task, to see that the various classifications in the overall system that is developed reflect the needs of the Congress.

The Comptroller General should extend the classifications and other standards developed in cooperation with OMB to systems in Congress interfacing with the substantive legislative process. This means that when other units of the Congress develop a computer system, the standards that are necessary to assure compatibility of the new system with the basic system of the Office of Management and Budget will be provided by the Comptroller General. The Comptroller General will not become involved in computer system design and development for other subordinate units of the Congress, but will provide the "rosetta stone" of compatibility that is essential to effective use of computers in Congress.

The Subordinate Units of Congress Should Maintain Responsibility for the Development of Computer Systems under the Overall Policy Control of the House and Senate Committees on Administration

Those units of Congress with present responsibility to provide the data under traditional systems of processing should be directly responsible for systems design, maintenance, and improvement. The Clerks and Sergeants at Arms of the House and the Senate should have responsibility for applying computers to the various housekeeping operations. These officials should also maintain responsibility for the mailing list operations and whatever computer applications the House and Senate may desire regarding the tabulations of votes on the Floors of the two houses.

The other main computer applications referred to earlier are the responsibility of the General Accounting Office and the Library of Congress, with an increasing role by the Government Printing Office as concepts such as "text editing" and computer typesetting prove out and are implemented. To the extent these Congressional support units need computers, they should be primarily responsible for system development, subject to compatibility standards set by the Comptroller General under authority extended in Title II, Legislative Reorganization Act of 1970.

STRUCTURE OF PROPOSED CONGRESSIONAL COMPUTER SYSTEM

The Committees on Rules and Administration

- Both the House and the Senate committees, independently, should oversee the development of computer systems by their subordinate units to ensure that the computer capacity to be provided meets the needs of the House and the Senate as a whole. To a great extent this means developing "laymanlike" statements of proposed computer systems. If subordinate officials of Congress responsible for system development can't fully explain the system output to the complete satisfaction of a layman, then they have not gotten to the point where the system should be implemented. Testimony or reports as to "progress" being made should *not* be accepted as substitutes for actual computer system descriptions.

- The House and Senate committees should order studies of the individual Members' offices to obtain an authoritative determination of potential computer applications (apart from bill status, and budget and appropriations) that might be of assistance now. Of particular interest should be development of a computerized filing system that would form the basis for updating the Member's computerized mailing lists, as well as relieve his staff of the drudgery involved in the maintenance of present filing systems. As these systems may be linked directly to basic congressional computer systems supporting the substantive legislative process, compatibility should be maintained through the Comptroller General by means of the standards adopted under the authority of the Reorganization Act.

  This project should be undertaken on a pilot basis, and the basic system should be "proved out" before any computer peripherals are installed in Members' offices.

- Both the House and the Senate should continue the present active development and improvement of computerized mailing lists to improve the communicative link between individual Members and constituents.

The Congressional Research Service and the Library of Congress

- The first job for the Congressional Research Service is to develop requirements for a comprehensive bill status system. This computer application is difficult, but nevertheless within the present state of the art.

- The Congressional Research Service and Library of Congress generally, in cooperation with the Joint Committee on Congressional Operations, should study and delineate what computers now offer in areas of reference and research. This problem definition is mandatory before any substantial sums of money are expended on Congressional Research Service computer systems.

The Comptroller General

The Comptroller General should modularize his efforts in the computer field in order to maintain the identity of the different programs and objectives that must be carried out on a parallel basis.

- The General Accounting Office should obtain computer capacity corresponding to that which the Office of Management and Budget uses in the preparation and maintenance of the budget. This will allow the Comptroller General's staff and interested Members of the Congress and their staffs to become familiar with the computer operation in the Office of Management and Budget. This familiarization is essential to future budget and appropriations cycle system design and the bringing of this capacity into the Legislative Branch on a broad and effective basis. Members and committees conversant with computer operation would also find this of immediate value. The Congress has data and program tapes for the current fiscal year available, but nothing is being done with them.

- Congress must have a budget and appropriations computer system comparable to and compatible with the system used by the OMB of the Executive Branch to prepare and maintain the budget. Fundamentally, Congress must have the capacity to make independent evaluations of its own data for use by the Congress. An independent Congressional computer system is also necessary to solve the "executive privilege" problem. Obviously, the President would never permit the Congress direct access to Executive Branch computer systems. They contain data about discretionary matters, future policies and expenditures, possible courses of action, etc., etc., that the President logically wants to remain confidential. Yet such data is often intermixed with that which Congress can justly claim and must have for legislative purposes. Therefore the transfer of data tapes and disks from OMB to Congressional computer systems is the only practical way of communication which also protects the informational integrity of both the Congress and the President.

- Plans should be made to bring into the Legislative Branch the new Office of Management and Budget computer system McKinsey and Company has designed to support the budget and appropriations cycle[1]. As with the present system, this will begin with compatible hardware in the Legislative Branch plus the acquisition of Office of Management and Budget computer programs and data base. On the basis of this system, future analytical capacity for the Congress will be developed. Classifications and other standards for this system must be extended to all systems in the Congress that interface with the substantive legislative process.

- A clear, laymanlike description of this new budget and appropriation computer system under development in the Office of Management and Budget should be created and provided to the Congress. If Members and their staffs know exactly what the new system is designed to provide in the form of data, much of the mystery and complexity that presently envelop the concept would disappear.

- There should be developed a hierarchy of categories of classifications, standards, and conventions that are essential to the development and maintenance of a compatibility linkage between the Office of Management and Budget and legislative computer systems.

  The highest order of budget classifications is the budget and appropriations system itself, in that there will be companion systems technically of the same "order" or "level" for use in both the Executive and Legislative Branches of Government!

  The next order of classifications is for the variations in which the basic budget can be formatted to meet the unique requirements of different officials. At present, there are 1200 budget accounts in the budget. But these 1200 accounts cannot be rearranged in any simple way to translate the budget data they contain from, say, the traditional object classification the Appropriations Committees require to the program classification the Government Operations Committees require, or any other classifications that are needed in both the Executive and Legislative Branches.

  The present Office of Management and Budget effort under the so-called "McKinsey Report" is to go down to seven or eight thousand cost elements that can be arranged into any format that might be required.

  Information as to these varying categories or formats must be provided Congress at the earliest possible date. There is at least the outside chance that Congress might require other formats of the budget that do not presently fall within the contemplation of the OMB's proposed system.

  Below these basic budget classifications are tiers of other classification standards and conventions relating to systems design and data that directly affect compatibility. These classifications soon become so technical in nature as to be beyond the interest of anyone other than computer technicians. However, congressional computer technicians must understand them to protect compatibility and the general interests of the Congress.

  The identification of these various categories is an essential first step in the development of effective standards for computer systems.

- The Comptroller General, in cooperation with the Director, Congressional Research Service, should begin an inventory of substantive data bases in and outside the Federal Government and that might be of interest or value to the various congressional committees. Included would be whatever computerized data bases have been developed either in the Executive Branch or can be obtained, that would be of interest to specific committees.

  As an example, data on unemployment and labor trends from the Department of Labor, presently maintained in computer form, could be adapted and made available to the House and Senate Labor Committees. Adequate safeguards, however, would have to be maintained to protect the purity of the basic congressional system from whatever incompatibility may lurk in the data obtained by this means, as well as whatever computer capacity may presently exist in the Congress that interfaces in any way with the substantive legislative process.

  This inventory would be reviewed and assessed incident to the preparation of an individual report concerning the potential benefits of computers to each specific committee. In the preparation of these reports, the Comptroller General would be expected to adhere to the similarity in structure of various committees referred to earlier in this paper so that he could plan to provide the computer capacity for all of the committees and Members by the determination of a net requirement to be obtained from the operation of just the one basic system.

- Discussions should be held with the Director of the Congressional Research Service, the Government Printing Office, and other possible support units of the Congress whose data requirements interface with the substantive legislative process, to work out a system of maintaining standards on a legislative-wide basis.

- The General Accounting Office's requirements for computer capacity should be evaluated, aiming at the development of a modern audit concept based upon computer usage. By this means, the GAO's requirements could be merged into those developed in the Congress so as to avoid unnecessary and costly duplication, as well as possible problems of incompatibility.

There Should Be a Congressional Computer Working Group

For reasons already discussed, there is little likelihood of obtaining rigid centralized policy control over computer system development in Congress. Nor is such an approach necessarily desirable except in the area of computer standards, to assure compatibility, and they are usually of little concern to the average user.

Yet the nature of systems design requires some degree of overall coordination. This coordination could be brought about by a quasi-formal working group. This group would represent the chairmen of the various committees involved in system design and implementation, as well as the Comptroller General, the Director of the Congressional Research Service, and the Public Printer. This group could identify some of the problems that will inevitably plague computer system development in the Congress and report these problems to the appropriate officials so that they might be resolved in a timely manner. Aside from this limited but most important function, such a working group would have no power and would not undertake any concerted action of a formal nature.

CONCLUSION

As the 93rd Congress begins its deliberations, the complex question of limiting Federal expenditures will be a primary subject of concern. It is a fact of life that neither man nor nation can live within available resources without reliable information about needs and expenditures. For this reason it is hoped that among the solutions that are devised will be the granting of top priority to the development of the computer system to support the budget and appropriations cycle. Above all other considerations, this is the most critical need of the Congress. With annual expenditures at the $250 billion level, even a minor improvement in the budget and appropriation system would save billions.

Computers are the only hope that our Congress has to acquire the basic data needed to control expenditures. Without this data there can be no effective Congress and, ultimately, no democratic system.

REFERENCES

1. "Strengthening Program Planning, Budgeting, and Management in the Federal Government", McKinsey & Co., 1970 Dec.

Selected Readings

American Enterprise Institute for Public Policy Research, "Congress: the first branch of government", AEI, Wash., DC, 1966. 515 pp. (see papers by C.R.Dechert, K.Janda, J.A.Robinson).

Brooks, Jack, "Data-processing techniques to aid Congress", Extension of remarks in the House, Congressional record [Daily edition], **115**, 1969 Jan 3, pp. E29-E31.

Califano, J.A., Jr., "Congress has been bypassed in analysis technology", the Washington Post, 1971 Jul 13, p. A-18.

Chartrand, R.L., "Congress, computers, and the cognitive process", *In* Planning and politics: uneasy partnership (T.L.Bevle and G.T. Lathrop, *eds.*). 167-187, the Odyssey Press, New York, 1970.

-----, "Redimensioning congressional information support", Jurimetrics J., **11**, No. 4, 165-178 (1971 Jun).

Chartrand, R.L., K.Janda, and M.Hugo, *eds.* "Information support, program budgeting, and the Congress", Spartan Books, New York, 1968. 231 pp. (see papers by Chartrand, Janda, W.T.Knox, R.McClory, F.Schwengel).

Comptroller General of the US, "Budgetary and fiscal information needs of the Congress", US Genl. Acctg. Off., Wash., DC, 1972 Feb 17. 49pp.

"Congress looks to computers for legislative help", Congressional Quarterly, 1969 Apr 11, 524-525.

Donham, P., and R.J.Fahey, "Congress needs help", Random House, New York, 1966, 203pp.

"Digitizing Congress", Electronics, 1970 Aug 31, pp. 41-42.

Glass, A., CPR report, "Congress moves into computer age but divides on control of new systems", Natl. J., **2**, No. 22, 1150-1157 (1970 May 30).

Hopkins, B.R., "Congressional reform: toward a modern Congress", Notre Dame Lawyer, **47**, No. 3, 442-513 (1972 Feb).

Hunter, K.W., "Toward better information and analysis support services for the Congress", Proc. Assoc. Comput. Mach., 1971, 139-155, ACM, NY.

Information systems: current developments and future expansion. Proceedings of a special seminar held for Congressional Members and staff, 1970 May 20, AFIPS [Amer. Fed. Information Processing Soc.], Montvale, NJ, 88p.

Janda, K., "Information retrieval -- applications to political science", Bobbs-Merrill, New York, 1968, 230pp.

Kravitz, W., "Preparation of committee legislative calendars by computer: description of a working system", Legislative Reference Service, Library of Congress, 1969 Mar 3. 8pp. + appendices.

Little, A.D., Inc., "Management study of the US Congress", commissioned for special NBC news report, "Congress needs help", 1965 Nov 24, 37p.

McClory, R., Congressman McClory suggests computer uses for Congress. Extension of remarks of Honorable Tom Railsback, Congressional record [Daily edition], **114**, E275-E277, 1968 Jan 29.

Maiorana, Charles. A computer system to aid in Congressional decision making. the George Washington Univ., Wash., DC, 1966 May 1, 40 pp.

Moorhead, William S. Computers for the Congress: the challenge and the potential. An address before the 15th Annual EDP conference of the American Management Assoc., New York, 1969 Feb. 15pp.

Schloss, L., "Congress needs computers -- but they're many years away", Government Executive **2**, No. 6, 42-43 (1970 Jun).

Schneier, E., "The intelligence: information and public policy patterns", Annals of political and social science **388**, 14-24 (1970 Mar).

Semling, H.V., Jr., "Congress and the computer", Modern Data, 1969 Sep, 68-71.

Sisk, B.F., "Toward a more effective Congress", Los Angeles Times, 1969 Oct 22, part 2, p. 1.

Truman, D.B., *ed.*, "The Congress and America's future", Prentice-Hall, Englewood Cliffs, NJ, 1965, 185pp.

US Congress. House Committee on Government Operations, "Use of computers in the legislative process", Report together with supplemental views of the committee, 91st Congress, 1st session. Report No. 91-258, US Govt. Print. Off., Wash., DC, 1969. 39p.

----, House Subcommittee on Government Activities. "Effective and efficient use of computers in Congress", Hearing before the subcommittee, 91st Congress, 1st session, 1969 Apr 23, US Govt. Print. Off., Wash., DC, 1969, 53pp.

US Congress. House Committee on Administration, First progress report of the special Subcommittee on Electrical and Mechanical Office Equipment. 91st Congress, 1st session, October 1969, 64 pp. (see also the second progress report, 20 pp., and the special report of the same committee, 17 pp., 91st congress, 2nd session), US Govt. Print. Off., Wash., DC.

----, House Committee on Rules, "Legislative reorganization act of 1970", Report of the Committee on Rules on H. R. 17654. 91st Congress, 2nd session, House Report No. 91-1215. US Govt. Print. Off., Wash. DC., 1970, 182pp.

----, Public Law 91-510 [84 STAT. 1140]. An act to improve the operation of the legislative branch of the Federal Government, and for other purposes. 91st Congress. H. R. 17654, 1970 Oct 26. 65pp.

----, Senate Committee on Rules and Administration, "Initial report of the Subcommittee on Computer Services to the Committee on Rules and Administration", 92nd Congress, 1st session, 1971 Jul 21, US Govt. Print. Off., Wash. DC, 11pp.

----, Secretary of the Senate. Administrative reorganization of the offices of the Secretary of the Senate. Based on administrative survey by O.B. Potter. 92nd Congress, 2nd session, US Govt. Print. Off., Wash., DC, 1971. 11pp.

THE ROLE OF A COMPUTER IN THE PUBLICATION OF A PRIMARY JOURNAL

by ROBERT W. BEMER

Honeywell Information Systems. Inc.
Phoenix, Arizona, US

INTRODUCTION

The Honeywell Computer Journal has had some acclaim for social responsibility in the computer milieu and for the extensive and pervasive use of a computer in the publishing function. The basic elements of the latter are described here. The Journal is published simultaneously in hardcopy, microfiche, and magnetic tape with embedded text control. Its mixed-media character is accented by the fact that not all articles in the microfiche and tape editions appear in the hardcopy edition.

Specifically, the copy that you are now reading has been produced by the identical methods of the Honeywell Computer Journal, as are all of the papers in the Methods and Applications Section of these Proceedings. Thus many of the features can be self-descriptive. The only differences are:

- Video Times Roman font is used here (instead of Optima).
- Column width is 242 points (instead of 228).
- Column height is 57 lines maximum (instead of 60).

To reset this paper for the alternate conditions would cost $3.50 per page!

MAJOR COMPONENTS OF COMPUTER USE

The computer plays a major role in:

- Subscription fulfillment.
- Entry of text, tables, and figures.
- Production of photocomposed copy, with justification and hyphenation.
- Control of page layout.
- Proofing of copy.
- Control of readability and style.
- Indexing.

All except the first and last functions are covered in this paper. The first is omitted because it is common, and we have made no innovations; the last because we make little use of this admittedly powerful feature for the Journal per se.

Furthermore, we do not use the automatic pagination features that are available to us, because computers can never be more than dull and pedestrian in this role. It may be suitable for a contract specification, or legal documents, but not for a publication that must be artistic, attractive, and readable. Automatic pagination also chews up expensive store and time to keep the total text in core to work with.

SCHEMATIC OF THE SYSTEM

The HIS 6000 system is used for text entry, editing, storage, and running concordances. It is not normally used for the "run-off" function (producing formatted copy on the entry terminal). Even though this feature is available, it is tedious, expensive in line cost, and has little value for final copy.

Formatted copy is produced only by photocomposition. When this is desired, a special postprocessor program converts the text stream and embeds macros for the Page 2 System. This produces a magnetic tape which is (now) transported physically to the facilities of Datagraphics, in Phoenix, and input to a Univac (nee RCA) 2 driving a III Videocomp 830. The resulting copy is laid up in desired page form, and a cycle of editing and further photocomposition begins.

Final copy is waxed on templates in the traditional manner. Special heads are added (in fonts not available to the computer system, and chosen to symbolize article content, where possible), and it's off to the printers.

Basically, we have adjoined two free-standing systems, and in so doing removed from the middle the expensive and non-graphic-quality output of the first, and the somewhat tedious and inflexible input of the second. Jury-rigged as it is, it is nevertheless superior to any method formerly available to us, and points the way to integrated systems for the future. We can live for now with our 2-hour turnaround.

TEXT ENTRY

Text entry is accomplished in the timesharing mode with the standard HIS 6000 Text Editor System,[1] an embedded format system based upon M.I.T. work and similar to the IBM Script. It is *not* a numbered line system like ATS, and eight years of experience has proved *this* wisdom. Searching and alteration are done primarily in the string mode. "Cut and Paste" is limited to operate by number of lines moved, but they are not numbered.

All control actions are signified by embedded "dot" commands. This input convention states that a CR (Carriage Return) character followed by a full stop character (period, dot) signifies a control statement, thus:

.begin	.center	.indent n	.subpara n	.TAB
.space (n)	.adjust	.undent n	.para	.break

These are but a subset of the standard Text Editor, and can be learned by an unskilled person in an hour or so. The editing commands will be explained in the running text of this paper.

ENTRY FOR UNSKILLED PERSONNEL

The postprocessor program that converts for the Page 2 System is vital for simple text entry. The standard entry methods for the Page 2 System are certainly *not* simple, and require some training and a crib sheet constantly on display to the enterer. Remember that graphic quality output requires a separate font generation for each unique character. It is not sufficient to overprint an umlaut (in its fixed position) for both the upper and lower case "u", for example. The postprocessor does extensive string analysis, much of it based upon backspace and overstrike for entry, which makes it simple for personnel. Examples:

- Characters with diacritical marks - accent acute, accent grave, tilde, umlaut, etc. - are produced by backspace on the terminal and overstrike with the proper character (double quote is used for umlaut).
- Double and single quotes are used as they are for entry. The postprocessor determines whether they are opening or closing quotes. A double quote is two single quotes in photocomposition, and this is called automatically.
- For minor occurrences in text, boldface may be indicated by overstriking single characters three times. This is visible on the terminal when the line is verified. For a longer string of bold characters, the font is altered by a **.bold** command, and turned off by a **.bold end** command. These commands do not force a new line.
- For minor occurrences in text, italics may be indicated by backspacing the length of the word and underlining. This is visible on the terminal when the line is verified. For a longer string of italics, the font is altered by a **.ital** command, and turned off by a **.ital end** command. These commands do not force a new line.
- The bulleting seen here is accomplished by a **.indent 3** followed by a **.undent 3** (which is operative only for the next line), a lower case "oh", 2 blanks, and then the text. The uniqueness of this string permits the convention.
- To the regular Text Editor convention of using the "at" symbol to delete the previous character (guess why our articles never contain this character!), and CAN to delete the entire line of entry, we have added the caret to indicate the "en" space, which is incompressible to the justification process. Thus a new paragraph is caused by a **.break** and an initial line with two carets for indention.
- Normal font sizes for the Journal are:

  9 point - text
  8 point - references, some displays as necessary
  7 point - sub- and superscripts, figure captions

  Point size may be changed at any point in the text by inserting the ESCape sequence:

  ESC g (7-pt), ESC h (8-pt), ESC i (9-pt)

  These override the original settings, and are used for formulas, etc.

CONTROL OF PAGE LAYOUT

It has been a remarkable discovery to us that reader attraction and satisfaction is increased significantly by tight control of page layout. Only in the most exceptional cases will a column start in the middle of a sentence, and then only on the second column of the same page. Usually a column will start with at least a paragraph (not just an arbitrary paragraph, but one that makes sense), and very often with a heading. The appearance of a figure or table will never precede its first mention in text, nor will it often be on a page that is not visible when that mention is made. "Widows" never occur.

Under traditional methods, the editor loses control of page layout after the galley stage; all of the niceties must be left to a composer who has little understanding of the subject matter, and is often less interested in reader satisfaction. With the low cost of text processing taken in conjunction with photocomposition, we do not mind expending many runs to get just what we want.

A quick reading of the first galley copy gives an estimate of the author's redundancy or flowery speech factor, and other ways that compression can be achieved if necessary. Accordingly, the actual film is cut to lay out an approximation of the article. As the last page is always full, we work backward. Whatever is left for the first page we leave for artistic treatment and the "From the Editor" commentary. Great attention is paid to aspects of future readability, left or right page assignment, pleasing placement of tables, figures and photos. Virtually no attention is paid to typos and other mistakes that exist in the copy. Accordingly, the single columns are taped on with more lines than our standard, trusting to editing to cut back to the right number (60).

The beauty of this system is that many things can be changed simultaneously to create correctness, harmony, and interest - point size for certain paragraphs or tables, tab settings, subparagraphing, font style, and text changes and corrections. Imagine a situation where the column copy has to be reduced by two lines, and yet previous editing has taken advantage of all short lines at the end of paragraphs, filler words have been removed, and big words replaced by commoner smaller words with equivalent or clearer meaning. Now you have to get into the guts of the author's meaning and say it shorter and clearer, without altering the flavor or meaning in any way! Being forced to do this by our aesthetic standards for page layout yields a big dividend in increased readability.

Depending upon the content, we may photocompose the text from 2 to 5 times. Do the authors complain about the alterations? Never, in our experience. When it reads well, they just assume that they wrote it that way, never checking their original copy. We have also experimented in putting the author's work in to typeset even when it is only rough draft; results seem to indicate that the visualization of final copy permits him to improve it more that he could by editing from a typed draft.

Obviously, taking this much work for readability means high acceptance standards, and we insist that this is a good thing. Dung coated with 53 layers of Chinese lacquer is still dung, and we do not intend contributing to information pollution.

Hopefully, it is now clear why we do not use the computer for automatic pagination.

PROOFING OF COPY

An optional feature, or byproduct, is the concordance run, usually exercised on what is expected to be the next-to-last photocomposition run. This produces two listings on the high-speed (upper case only) printer. The first listing is a Key Word Out of Context (KWOC) listing; each numeral and word (except for the very small common ones) is listed on the left in collating sequence order, with its entire entry line on the right. The lines *are* numbered here, for cross-reference to the second listing, which is the consecutive text.

The concordance is now scanned visually, primarily to detect input errors ("typos"). See Figure 1 for some examples. It is our experience that these fairly jump out at one in scanning a concordance, whereas they remain stubbornly glossed over by the eye and mind in traditional proofreading. However, we do read the text - for style and making sense, not for typos. In fact, knowing that you are freed from the typo-hunting task creates a different frame of mind for doing *real* editorial work.

```
AUTOMOHILF          DISPOSITION         MASSACHUSETS
AUTOMOBILE          DISRRUPTION         MASSACHUSETTS
AUTOMOHILIES        DISRUPT             MASSACHUSETTS
AVAILABILITY        DISRUPTIONS         MASSES

CERTAINLY           INSTEAD             SOLVED
CERTAINTY           INSTEAD             SOLVED
CERTIFICATION       INSTITUE            SOLVEDTO
CERTIFICATE         INSTITUTE           SOLVED,
CERTIFICATION       INSTITUTE           SOLVE,

CHARGES             KURT                STEREOSCOPIC
CHARGES             KY,                 STEREOTYPED
CHARIMAN            LD                  STEREOTYPED
CHARITY             LA                  STEROGRAPHIC
CHARLATANS          LAR                 STEROTYPE
                                        STEWARDS
COLUMBIA            LOCATIONS
COLUMBIA            LOCATION,           SUCCESS
COMAPNY             LOCATION,           SUCCESS
COMBINATION         LOCATIOXS           SUCCUMBING
COMBINATION         LODGED              SUCESSOR
                                        SUCH
```

Figure 1. Typos Exposed by Concordance

CONTROL OF READABILITY AND STYLE

The concordance produces a histogram of word size distribution as a byproduct, and the average word length may be calculated. We target 5.0 characters per word, and are very suspicious of readability when the author gets above 5.5.

One aspect of style, or rather one of our rules, is that an acronym shall always be given the spelled-out version in parentheses the *first* time it is encountered in text. One has only to spot the first occurrence in the concordance, and look to the corresponding line on the right to see if this has been done. If not, edit.

The Journal has other style rules. Most important is adherence to ISO Standard 1000, or the International System of Units (SI). Check the concordance for inches, feet, yards, miles, pounds, etc. if they occur, and are for measurement, they had better be in parentheses following a metric value. Other examples: $2 million - not 2 million dollars; 0.5 s - not .5 sec; focused - not focussed.

ECONOMIC CONSIDERATIONS IN WORKING METHODS

As there is no way to predict the pagination of printed copy when entering text, one could enter it all under a single file name. However, the 6000 Text Editor keeps the entire file in the main store for faster processing (and it is *really* fast), and these facilities must be paid for. Thus original input is made in judiciously separated and named files, breaking at headed sections, for example. These are then adjoined for the photocomposition run.

After page layout is determined, they are adjoined again and resplit by page into files with new names, and the old ones purged. This permits single columns to be reworked into final form. The present rate is $1.75 per column. Thus a page costs from $6 to $10 to compose, comparing rather favorably with the $70 per page we were paying for linotype setting to our standards before our system was operable. The 6000 cost is not included, as we have been unable to get real figures because we work on an inhouse "exposure" system used for checking out new software releases. We do, however, feel that this cost is compensated by the system doing automatically what we would have to do ourselves otherwise (like proofreading), and the added quality. We do need to modify to set double column on the last run.

Economy dictates that we should process as much text as possible on each photocomposition run. This means linking several files and saving them as a single file. But this increases the risk that something going wrong early will spoil the balance. Care must be taken to separate and insulate each file from any other. Convention starts each file with **.begin** (for a new galley), **.indent 0** (in case the file ahead of it lacked a command to restore indention to 0), and **.adju** (in case the preceding file had been using tabulation and was not restored to the justification mode).

The power of the Text Editor is of great assistance in checking for correctness of the adjoined file, particularly for closure. Type:

fs:/.bold/;* (meaning "find all occurrences of that string")

and you will almost instantly get a message like:

end of file - request executed 122 times

Hit "b" and CR (for backup to the file beginning, and type:

fs:/.bold end/;*

If the message doesn't say 61 times - trouble! A 60 would mean that bold did not get turned off somewhere, and the copy following will be in useless boldface. Do the same for italics, subparagraphs, point size changes, etc.

The files must always be correct for the magnetic tape edition, and identical to the printed copy. Yet it is often wasteful to rerun the entire file for simple patches. A copy is made, and the correct parts wiped out by string replacement, leaving only the changed copy to be reset as a patch (with due consideration to leaving enough text so that paragraphing, etc., is unchanged). These patches are saved under a different name; a number of them are adjoined and run at one time

INCIDENTAL ADVANTAGES

A number of dividends have shown up that we amateurs did not really foresee:

- Doing our own typesetting permits laying up mechanicals for articles as soon as they are ready, without waiting to group an entire issue for the typesetter to schedule in some time slot. Exclusive of conditions of extreme timeliness, this permits better selection for issue makeup and content.
- Having the feel of the final product, by mockup during the editing and changing stages, affects everyone - author, editor, and reviewer. For the latter, particularly, it gives psychological impetus to hurry up - lest what he dislikes might be in the finished product. All can work simultaneously to correct and improve the copy and make it more readable.
- The Page 2 System hyphenates to English rules and/or custom. Normally we run our French, German, Italian, and Spanish sections in "fill mode" (stretching the spacing between words to fill the line without hyphenation). But if glaring gaps exist we remove them easily by doing a dummy hyphenation, splitting the first word of the next line into two components:

```
rs:/whippersnapper/
ENTER
*whipper- snapper
*
READY
```

  This technique can also be used in our English text when Page 2 fails to hyphenate opportunely or (rarely) incorrectly.

  On one occasion the entire article was side-by-side in both German and English. Here we could proceed more elaborately, removing Page 2 hyphenation that was incorrect for German, forcing correct hyphenation paragraph by paragraph.
- Page 2 also has the flaw of assuming that a change in font style permits a break for a new line just as hyphenation or a space does:

```
..... Protection A
      gency ...
```

  Text Editor can force a correction by replacing sufficient spaces between words by incompressible en spaces.

- We don't have to worry about losing corrected galleys in the mails, as the Journal of the ACM did in 1971 October. We also know that the corrections have actually been made in the printer's copy, without waiting for a blue to be returned and show that they were *not* made. This often shortens the production cycle, and certainly cuts costs.
- Secretaries can make very creditable copy inhouse by cutting and pasting galley segments with Scotch Tape, and then using a reproduction method such as Multilith. Interoffice memos are becoming artistic, easier and pleasanter to read, and certainly use less paper.

OUR WISH LIST

A major purpose of the First National Computer Conference and Exposition was to have the end users tell the suppliers the nature of their applications and what they would like to accomplish those applications better, cheaper, and faster. I must follow my own principles. We would like:

- A larger portion of terminals to be equipped with cassettes. Entering text in the timesharing mode is not efficient in line cost.
- Cassettes attachable to office typewriters. If this means new office typewriters, then let them have standard keyboards! By this I mean not only the placement of the printing symbols, but also the placement of the controls, either as separate keys, or in the control position on the regular keys. For example, Control-X is the usual position for CANcel (deletes the line just typed). Some keyboard designers have not realized that this makes Control-Z a poor place for EOT, because a slip of one position turns off transmission, with resultant loss of all one's work to that point!

  With an increased portion of input being generated offline, it would appear that the introduction of the computer at the proper point in the copy production cycle permits entry by *less skilled* people, possibly to the point where the original creator of the text and the enterer are one and the same person. One can imagine an author out in the woods typing his rough copy and getting a cassette record. He would mark up the pages as needed, and send both pages and the cassette to an editing service, which would enter the cassette contents and make online corrections to the author's copy according to his indications.
- Alternatively we would take a CRT display if it corrects certain faults of existing systems in line runaround, etc.
- And perhaps a pointer system that could indicate both the beginning and end of a string to be identified for a working purpose.
- A registry of available digitized symbols, so that one would know where to buy their representations in a transferable form.
- More than any hardware imaginable, we would like to see the development of a common composition language, and its elements, that is, universally-agreed encodings for printed symbols - their graphemes, their placement, and their style. Elements of a proposal follow:

FEASIBILITY OF A COMMON COMPOSITION LANGUAGE

Production of graphic copy from encoded data is an important component for present and future information retrieval systems. Dot matrix characters on a CRT screen will just not be satisfactory for some purposes. Production of graphic hard copy from an information bank may in the future be cheaper than ordering an existing printed reproduction to be invoiced, found, packaged, mailed, and delivered.

Because future information retrieval will consider many more symbols than those of the present ISO Code, existing and future graphic devices must be connectable to the retrieval system.

Equipments that produce hard (or film) copy may be viewed in the same way that we view computer central processors utilizing different instruction sets and object code, and as we view various numerically-controlled machines. There are single programming languages that are common to many central processors. In N/C, the APT language is processed to produce the CL Tape, which is also common to many processing machines. In both cases the common language is processed by computer to produce instruction for specific and multiple equipments. In both cases the translation capability to specific equipment is usually the responsibility of the manufacturer of that equipment. That this is not so in the composition industry is due to the lack of a standard composition language and metarepresentation of text (with associated characteristics of alphabet or other symbol class, font, size, style, weight, and 2-dimensional positioning). If this existed, it would be a high-level language for copy production which is translated, by computer, to instructions for the various hard-copy equipments. The industry suffers from this lack.

To be feasible, the basic functions of copy production must be similar, even if not carried out in the same way. This appears to be so; it has been proved for the Honeywell Computer Journal, which can also be printed from entry terminals. Indention, font change, size change, etc., seem to operate as primitives.

To construct a general text-processing language, of which the composition language is one part, we need to enumerate the functions and then assign standard encodings to them. The provisions to do so exist in the ISO Code and the associated expansion and extension techniques. The most general mechanism is ESCape, although SO and SI exist. Some 2-character ESCape sequences are now virtually standard in the 7-bit code, and will likely be single characters in the 8-bit expanded code. Examples are Half Line Reverse Feed, Cursor Up.

Utilizing code extension procedures, provisions are made to be able to select unambiguously a group of symbols, a font, weight, size, etc. We then use a key device or pressure display panel with single function buttons. The operator would perhaps press "Cyrillic" (to get the GOST Standard encoding), "8" point on "10", "bold". Each key would generate an ESCape sequence in series, inline in the text. He then uses either a special typewriter keyboard, a standard keyboard with a chart of correspondences, or some other device, to enter the Russian text. One can imagine the total set of symbols paged on a microfiche for back projection on a screen.

Computer programs (postprocessors) are created to translate from this standard language into the actual commands and character inputs for the copy device, which could be 6-level Teletypesetter, Monotype, Photon, RCA Page One and Videocomp, Datel typewriter terminals, IBM Selectric Composer, etc.

Until new entry equipment is made available to conform, similar preprocessors could be written to convert from the various entry conventions to the metarepresentation. This would reduce the translations from N! to 2N. If all entry equipment would eventually conform, a further reduction to N occurs, where N = the number of different composition equipments.

It is expected that this would free the photocomposition industry for expansion in the same way that FORTRAN, COBOL, and ALGOL did so for computational usage. It would provide international standards for alphabet representation, to aid the UNISIST project.

CLASSIFICATION AND GROUPING OF SYMBOLS INTO PAGES

ISO TC46 (International Standards Organization Technical Committee 46), Documentation, has a Subcommittee 4 on Automation in Documentation. This body has responsibility for collecting and/or developing the pages of encoded symbols. Examples of such pages are:

- Characters to form natural languages (alphabets)

ISO [DIS 646]	Kata Kana [JISCII]
National/accented	Kanji
Cyrillic [GOST 13052-67]	Braille
Greek	Phonetic
Hebrew	Dactyology [hand signs]
Arabic	....
Sanskrit	Other punctuation [character augments, bullets, rules, bars, leaders, etc.

- Symbols of various fields

Aeronautics	Medicine
Astronomy [Astrology]	Meteorology
Biology, Botany	Money
Business [Commerce]	Music
Chemistry	Philately
Ecclesiastic, Fraternal	Pictorial, Ornaments
Electricity, Magnetism	Transportation
Flowcharts	Typography
Games	Welding
Heraldry [flags, insignia, arms]	....
Logic diagrams	Other Scientific
Mathematics, Geometry, Physics	

- Controls - for changing point size, weight, slope, font, position relative to the base line, horizontal compression, etc.

An ESCape sequence and prefix character should be proposed for each page of symbols, for registry with ISO TC 97, Computers and Information Processing, which body maintains this registration authority for extension and expansion of the ISO Code.

REFERENCE

1. "TEXT EDITOR Quick Reference Manual, Series 600/6000", Honeywell Information Systems Inc., DB42, 1972 June.

A CONCEPTUAL FRAMEWORK FOR MAN-MACHINE EVERYTHING

by THEODOR H. NELSON

University of Illinois
Chicago, IL

This paper is not about everything between man and machine, but about *man-machine everything,* that is, the desirable future condition where most of our information and tasks are attractively and comprehensibly united through man-mechanisms. The breadth of possibilities is mind-boggling, but it does not seem to be clear to people yet that they are possibilities for the choosing, rather than eventualities to be engineered. The myth of technical determinism seems to hold captive both the public and the computer priesthood. Indeed, the myth is believed both by people who love, and by people who hate, computers. This myth, never questioned because never stated, holds that whatever is to come in the computer field is somehow preordained by technical necessity or some form of scientific correctness. This is cybercrud.*

Computers do what people want them to do, *at best.* Figuring out what we should want, in full contemplation of the outspread possibilities, is a task that needs us all, laymen no less. There is something right about the public backlash against computers: things don't have to be this way, with our bank balances unavailable from computers, the immense serial numbers of our drivers' licenses generated by computers, the unstaunchable rivers of junk mail sent to us by computers. And it is the duty of the computerman to help demythologize, to help the intelligent layman understand the specifics of systems he must deal with, and to help the public explore the question, *what do we want?*

Various "professional" approaches to our online future have confused us and left us stumbling. I refer particularly to (a) the field of "computer-assisted instruction", where a computer is often programmed to act like a crabby schoolmarm, coercively leading students around by the nose and chiding them personally; and (b) the field of "information retrieval", where a computer is often programmed to act like a wind-up librarian, sorting new questions into obsolete categories.

The possibilities are much, much wider, and not to be restricted by the parochialism of "professional" approaches. This paper approaches the question: "If computers can give us rich services - and they will - what do we want?" And it supposes that the answer is specific. And that a lot of new terms are needed to cope with the variety of what's coming.

* *"Putting things over on people using computers".*[1]

THE HOME SYSTEM

Computer fans agree that the home computer is on the way. Soon a minicomputer can be put on a few integrated circuits, and the price will be right - perhaps a thousand dollars retail before discount pricing. But the question of how we will use it, and thus how it will be marketed, stalls such an enterprise. There are perhaps four models for the Personal or Home Computer. We might want it as a Calculator; Genie; Toy; or Crystal Ball. The question is, what will catch on? We have always had a strange inability to realize what will Catch On Next, though by now we have the vivid hindsight precedents of gramophone, Kodak, telephone, movie, TV, tape recorder, Instamatic, videotape, audio cassette, stereo LP, pocket calculator, and so on. But now what?

People are not likely to pay a thousand dollars for a calculator. As to the genie - something that will open garage doors, manipulate the hi-fi and the model train - the interface costs are prohibitive. The supertoy idea is swell, but too expensive for most of us. Inescapably the home computer that catches on is going to be a Crystal Ball: that is, its principal function will be as a general-purpose viewplate into realms of digital text and graphics. It may cost more, of course, but in this form the Home System may provide a sufficient and viable basis for a whole new market.

Various combinations have been suggested for media of the future, from branching video cassettes to (almost) holograms with dial-up audio. But when things get sorted out there will be resolution to fewer things. Just as movie-makers usually do not mix-and-match different forms of output, but stick with sound-on-film 35mm, certain combinations in the grand computer-audio-visual realm will surely predominate. The question is where to cut and combine, what not to bother with, and pre-eminently, what will Catch On. I think when the smoke clears our main new medium of the future will be the branching, performing, digital text-and-picture package. Coming over the phone (or other) line to the home system, in pieces summoned by a chain of user choices, it will be almanac, encyclopedia, novel, comic book, playground, travelog, and time machine. The Home System will thus be both a Fun System and a Work System.

It is conjectured that a Universal Console, a text-and-picture demand console, will evolve - standard in its performance and interface specifications, permitting the free interchange of materials. Such a general unit must include graphics refreshment, keyboard input, a selection device, and many service provisions and conventions. There are of course two major ways to do this: as a satellite "terminal" to big computers, or stand-alone.

The usual supposition is that graphics systems need support from a big computer. Indeed, for that case we now have a stunning demonstration that mass computer graphics are practicable, the PLATO system.[2]

The opposite approach is exemplified by XANADU,* designed over the last decade under private auspices. XANADU is presently under development as a program for a popular minicomputer. However, possible reduction to microprocessors - specialized to the functions of retrieval and display - is foreseen.

XANADU's basic design criteria were to compress the communication, retrieval and revision of big and fancy files, along with interactive animation, into a small stand-alone machine. The software design, completed in 1972, may best be characterized as a retrieval-and-animation complex handling very large and versatile files. These files, stored on a mix of disk and tape, may be recursively coupled (annotations on annotations on annotations), with couplings surviving revisions; they may have numerous separate data-type breakouts or "enfilades"; they may be large, being currently defined over a addressability-space of 15 x 2^{20} elements. XANADU files are subject to extremely rapid revision due to the storage structure, retrieval and edit algorithms; these will, for instance, swap two halves of a large book in a few consecutive disk fetches and writes. Finally, the system has an unusual display language, DINGO (Display lINGO), permitting interleaved retrieval and animation, and maintaining picture stability while data and picture parts change. Finally, the design permits incremental roving in *n* dimensions of uniform data web, *n* not related to the number of enfilades.

None of these things seems difficult by itself, or with "enough core" on a big machine; but setting up to operate on a mini (16K or less) *without stopping for breath* has been the problem that we believe has been solved in this proprietary design.

XANADU is intended as the programming substrate for a variety of simple-user front-ends involving complex animation, retrieval and data entry. Designed as a stand-alone system with communications facilities, it is expected to function as a network machine *simpliciter,* nothing else being necessary to communicate text, pictures, or interactive animations between XANADU sets. We intend to create a standard XANADU file transmission protocol for all varieties of text, pictures, etc. This involves a complex range of mating conventions among files and programs and data, including preambles with faceted data classifications, and default conventions of program and data, so that, e.g., 2-dimensional graphic picture lists can be piped through 3-dimensional display programs with the missing features assigned the proper arbitrary values. Finally, our approach to security involves systems of criss-cross integrity checks to prevent falsification and counterfeiting, a problem that perhaps has not received enough attention for library systems of the future.

One hope is to promulgate this system with sufficient force - e.g., widespread licensing and PR - to create a *de facto* standard for extremely intricate files. The main problem is, of course, how to enforce standardization in a field where intentional destandardization is a universal lowdown trick.

In any case, since the undemonstrated XANADU system has met much incredulity, we will merely assume here that if this one doesn't work there will be another one of comparable breadth, and talk about what that ought to be.

** "XANADU" and "Parallel Textface" are claimed as trademarks for computer systems offered by the Nelson Organization, Inc.*

INTERPENETRATING SCREENWORLDS

We all agree that, one way or another, a heyday of computer graphics is coming, and for uninitiated users (let's call them simple, as they may not necessarily be naive). But it seems to be supposed that the simple user of graphic systems will still have the same psychological environment of today's computer user: he will "call programs" and employ "terminal languages", or at best make selections from uniform-looking columnar menus. In other words, there will still be explicit user-invoked transactions and transitions among data and programs. A little thought may reveal that this is neither desirable nor necessary. We want to be able to roam across boundaries, to call things from one place into the windows of another. Thus tomorrow's sensible graphic systems should permit *merged graphic composites* - 2-dimensional tapestries or 3-dimensional scenes that may be selected and blended from among available graphical and program structures, and roamed over freely by the user.

This suggests that a preliterate child, for instance, could guide his display screen down a carnival midway with a joystick, turn to watch a cartoon "juggler" do tricks with numbers, and then, if interested, guide his screen through an entrance into a "circus tent" where the number tricks continue. An adult, roving on his screen through explorable views of Stonehenge, may branch from twinkling screen-markers to the many theories about it, and thence to the books and articles expressing these theories - all the while he still explores, and searches out relevant angles in, the 3-dimensional model of Stonehenge still on part of the screen.

Such screenworlds can be created for the wholly computer-naive. The sophisticated user should be allowed to move with freedom through graphic tapestries opening not only into performing graphics and text, but other services and structures as desired.

THE VENDING MACHINE OF IDEAS

The data conventions of a Universal Console system will allow interpenetration of contents; e.g., juxtaposition and interframing of graphics, windowing between graphics files, and selection mechanisms among them which include the showing-through of jump markers and other advertising for materials available.

If we call a graphic environment and its rules a "screenworld" - whether a tapestry of drawn data or a set of simulation programs - then this many-ported visual (and calling) access between them creates *interpenetrating* screenworlds. The advantages of such explorable graphic mosaics should be obvious: roaming over them will be like perusing the Sunday comics (or Ray Bradbury's Illustrated Man), without getting lost, remaining always in a vividly comprehensible setting. Editorially we will be laying out such tapestries and scenes like magazine spreads.

The question is, what does the human mind want? Given the possibilities of digital exploration, what systems will be best for scholarship, learning, creativity and fun (all closely related)? What are the cleverest and best unifications? We have yet to find out these answers. But to suppose the desirable systems resemble "instruction" or library searches is hugely premature.

We will probably want a variety of things that may be grouped loosely under headings of "responding resources" and "hypermedia". By responding resources I refer to the kinds of things computer people usually think of as "useful programs" - JOSSes and simulators, timetables and typesetters and so on; services and facilities and programs. Hypermedia ("Hyper-" here meaning "extended, generalized, and multidimensional" - roughly the mathematical sense) are essentially prearranged presentations without fixed sequence: animated, branching word-and-picture bundles. These include branching and performing graphics, and branching or performing text, or hypertext.

A few simple examples should indicate the potential power and usefulness of hypertext. Consider the simple case of quoted material in writing. Seeing an interesting quotation in text, it would be nice if we could ask to see it in its original setting, and have the present surroundings fade into the original surroundings of the quotation. We could read in the original to satisfaction, and then return to the setting in which we saw the quotation. This quoteback feature may be thought of as links, of quotations to their sources, that we may jump along.

In another application of simple hypertext, many of us long to be able to follow news stories over enough time and detail to transcend the plainly misleading headlines - but can't, given the existing structure of news media. Hypertext could make it possible. When authors and editors are given the ability to create such discrete jump-links, the character of writing should change dramatically. The potential strength of such new forms of writing can only be surmised at this point, but it should be considerable.

All this was seen by Vannevar Bush in a classic article,[3] but what he really said has been largely ignored[4] and the ramifications of this approach - hypergraphics, hypercomics and so on - have scarcely been touched.[5] As with the movies when they were first introduced, most people are having difficulty visualizing the possibilities.[6] We may summarize some interesting conjectures on hypertext, or branching text structures.[7]

Conjecture 1: we've been speaking hypertext all our lives, and never knew it. Tinkertoy structures of thought, inherently parallel, must be conveyed on the linear conveyor belt of speech. Cross-citing connections by intonation, self-interruption, pushes, pops and cross-reference, has always been a daily problem.

Conjecture 2: there has been pressure toward hypertext since the written word began. (Consider the footnote; hypertext is immanent in any attempt to put text into man-machine systems, and is certain to emerge no matter where we begin.)

Conjecture 3: the interconnective structure of hypertexts will gravitate toward the real structure of the thoughts expressed.

Conjecture 4: understanding of complex relations will come to the hypertext reader via traversal in different directions - like learning the way around a complex piece of architecture.

Conjecture 5: hypertext will be easier to write. This is because rather than *deciding among* expository and transitional structures, the writer may use them all.

It is hard to guess what the forms will be. I suspect that hypertext need not be generally sequential or hierarchical, or general-to-specific in structure, or have obligatory catenas or sequences for the reader to traverse. This remains to be seen.

PRESTIDIGITATIVE PUBLISHING

To make these things possible, the Universal Console must be complemented by a range of meshed services: by central feeder machines (large or small), forwarding message and graphic complexes between consoles, and serving up prepared materials. We may call this latter "prestidigitative publishing", involving as it does both the rapid motion of digital data, and the supplying to screens of material that may be controlled "like magic".

The general-purpose text-and-graphic console may thus plug into libraries, explanatory and teaching complexes, literary and entertainment clusters. But we may also expect the basic console to be merged into complex control systems, with a variety of sensors and effectors. (Therefore the data conventions will have to cover a much wider base, expandable to all possible input and output modalities just as ASCII is expandable to all possible alphabets.) Many modalities may therefore in principle contribute - text, diagram, video, feelies and smellavision and whatnot; but most basic will be text and pictures, for these will be able to come or go among standard systems.

PYSCHIC ARCHITECTURE

I can now state what I believe to be the central problem of screenworld design, and indeed of design of man-machine *anything* - that is, psychic architecture.*

By the psychic architecture of a system, I mean the mental conceptions and space structures among which the user moves; their arrangements and their qualities, especially clarity, integration and meshing, power, utility and lack of clutter.**

It should be noted that these notions are much like those by which we judge regular architecture, and indeed the relationship would seem very close. An architectural grand design - say, of a capitol building - embraces the fundamental concepts a user will have to know to get around: main places, corridor arrangement (visualization and symmetries), access structure. These concepts are the very same in a screenworld or other complex man-made virtual structure: main places, corridors or transition rules (and their visualization and symmetries), access structure. It is a virtual space much like a building (though not confined to three "normally" connected dimensions), and susceptible to the same modes of spatial understanding, kinds of possible movement within, and potential appreciation and criticism.

** "Psychic" is used here for the dynamics of feelings and ideas, as distinct from "psychology", whatever it is psychologists study; - "mental, as distinguished from physical and physiological" - Funk & Wagnalls Standard Dictionary, 1960 ed.*

*** Similar criteria are also considered in (8).*

The orientation problem in both cases - real building and screenworld - is immensely important. Because there is no "natural" structure is to fantic space (definition to follow), as there is in our 3D world, great care must be given to maintaining the user's clarity of mind. Especially for this reason, the fantic space should have a Grand Design - an overall shape easy to remember and visualize in some way. I think that there is art to it, that it is not all "human factors" and reinforcement schedules.

I propose the term "fantics" for this newly-structured but very old realm, the art and science of presentation, especially to the mind, sometimes to the hand. I derive the term, like most English "fant-" words, from Greek *phainein,* show, and its derivatives *phantazein,* render visible or present to the mind, and *phantasia,* appearance or imagination. The related English terms *fantasy, phantom,* and *phantasmagoria* (succession of zooming images) also contributed to forming this word. The word "phantom" is used in the graphic arts for diagrams which show opaque things transparently and also in medicine, for a victim's feeling that a lost part of his body is still attached.[9]

These usages suggest visualizations and physical sensations that come and go, clustering matters I think belong together: showing and presenting things, visualization and kinaesthesis, and alternate ways to structure them in information systems.

Thus *fantics.* Computer graphics will be its principal mechanism, but not its center. Its center is the communication of ideas and thoughts, whether they be facts, poems, or body gestures translated electronically to complex happenings on another planet. Inventing the best presentational media from among the remarkable options now requires our close attention.

FANTIC UNIFICATION, CONSTRUCTS, AND FIELDS

By "fantic unification" I mean tying things together in a central presentational or control structure which unites them conceptually. Example: several wing-flaps of an airplane are united in its control yoke, a crescent on a rod which may be both turned and moved forward and back. The airplane's flaps do not individually correspond to a desired effect, nor do the combined movements of the control, necessarily; yet this integration provides a convenient unified "feel" to the pilot.

It is in much the same way that we unify things in all presentational modalities - in writing, in diagrams, in movies, or whatever - creating structures, organizing principles or unifications which have an integrating conceptual character. Often they may have a fictive or not-quite-real component, yet this fiction may contribute some kind of clarity or simplification, allowing the mind more neatly or conveniently to manage information.

We may define a fantic *construct* as a virtual reference structure used to help imagine or handle ideas and things. It may be added to subject matter or somehow put into a presentational or manipulative system. This concept of fantic construct, then, extends from sequential organizations and headings of text to grossly artificial mnemonics. A fantic *structure,* however, is the structure of a presentation or presentational system, whether experienced by the user, intended by the designer, or discovered later on by somebody else, or an abstraction never suspected.

A fantic *field* is the fantic structure of a complete and closed presentational, manipulative and/or conceptual field system, within which complicated things may be shown or handled. Thus communication media such as books and radio are fantic fields, but then so are complex interactive screenworlds and work-systems such as Sutherland's.

Fantic *controls* are any controls whose correspondence to the realm affected is restructured or mediated by fantic structure, whether by fictive fantic constructs, integrated transpositions, or some other form of conceptual combination or rearrangement. Thus wands and puppet-yoke controls, and "virtual gloves" with which we feel inside a display-space, are fantic controls, but brakes and gearshifts are not. The suppositions are these:

- We now pass to an era where the structure of objects themselves is less important than formerly. Not just using or hooking into objects, but structuring the perceptual and conceptual field interestingly and usefully, is the problem.
- This is *the same as* the general problem of creating presentations in written, audio and other media. Thus we unite with writing, theater, movies and plastic and graphic arts: organizing for presentation to the mind. The problem is aesthetic as well as cognitive and functional. The aesthetics are important and, if not inseparable, should not be separated.
- The principles of psychologically reorganizing receptors and effectors in complex man-machine systems are the same as those of organizing thoughts and other intellectual materials for presentation to the mind.

The most basic principles are *making things look good, feel right,* and *come across clearly.* Perhaps there are special-case principles, like those offered by learning theory and "human factors", but clarifying their correct range of applicability is essential.

It should be realized that it is not only screenworlds to which these criteria apply, but any media and arbitrarily-structured entities to be presented to people. (For example, if someone were to develop a "hypermusic", with alternatives among which a user could move, it would presumably be subject to these criteria.)

PERFORMANCE VALUES AND VIRTUAL SPACES

We need a general terminology for the performance features and special effects that we are going to see in the coming years. Unfortunately, because of the variety of devices, modalities, subjects-matter and professional specialties touched, such a common vocabulary emerges only with difficulty. If we concentrate on aspects which are independent of particular areas and subjects, we obtain some generality, of at the price of occasional vagueness. The following terms have come about from rather detailed considerations of possible screen performance techniques, but wider generality is intended. Hopefully the following language applies regardless of what we are showing or controlling, or how.

A number of performance features or special effects are desirable; we may call these "performance values" (cf. "production values" in films). Many performance values may be turned inside out, and described as if they were places and events.

Fantic space. Spacelike structure, of accessible text, pictures, animations, etc.*

A space may be 1-dimensional (plaintext), 2-dimensional (a tapestry), or 3-dimensional (a scene or object) or even higher. If it is not regular, but consists of two or more dimensional zones attached by discrete connections, I suggest the term *funny-space* (which can be suffixed with the dimensionalities of the zones). Note that the fantic space of the *contents* joins with the fantic space of the *viewing system and controls;* the result we should perhaps call a *grand fantic space* - the overall space the user perceives, thinks about and moves in.

(Note an inversion here: the concept of fantic "space" makes a user's viewplate or other sensorium a moving vehicle, rather than a stationary place to which data are brought. This is as it should be, taking the wider view.)

Fantic contents: The contents of fantic space, as a system of arranged materials and potential performances and events. As in modern cosmology, the space is defined by its contents. *Fantic tissue:* the connective structure of fantic contents, particularly of their interconnections and transition arrangements. *Data web:* the data structure which underlies fantic contents, not necessarily homologous or proportional to them.

Direction: transition gradient in fantic space or tissue; may be presentational or psychological or content-based; may be mapped in visual analogs. *Roving:* moving through a fantic space. *Jump:* discrete step between discontiguous places in fantic space. *Choice point:* place where a choice may be made; e.g., one displaying a menu of jumps. *Juncture:* joining place of two continua of fantic space, contents or tissue. If presentable, it may be a choice point involving continuous alternatives, e.g., paragraph beginnings which continue off the screen.

Rover: a movable place-marker that denotes a currently accessible location in fantic space. *Jump-set:* a set of things to which one may jump at a given moment. *Fast track:* a set of rovers constituting a jump-set; that is, a bunch of markers you can move individually and keep homing to. *Jumpstack:* a stack holding addresses, in series, of jumps to be undone by a RETURN function. *Jumptrack:* a currently active system of recorded jumps, all remaining accessible under some scheme, and constituting a fast track.

Border: notable division between spaces, places, media, works, services or facilities, fantic fields, etc. Note that they may connect only at a few points. *Opening:* local access to another linked place.

Portal: opening permitting full movement of a rover. *Window:* opening permitting access but not unrestricted rover movement. *Border station:* official portal (crossing- or entry-point) on a border. *Tunnel:* quiet portal between spaces, places, etc.; border is not seen. *Customs:* markers, crossing protocol and/or restrictions at a border. *Crossing zone:* area of parallel or other multiple access across a border. *Seamless tissue or web:* tissue or web having no break or sharp discontinuity in performance at a border, or simply no borders.

** Cf. "filmic space", the virtual space created by intercutting different shots in the movies.*

Link: connection between two points, as between rovers and placemarkers, which may serve as a jump from one to the other; connector between ends of an opening. *Coupler:* facility permitting a link to be made, or data structure resulting. *Prehensible coupler:* one coupling into a file that has not been forewarned. *Multicoupler:* facility permitting multiple links to be set between parts of two entities, thus establishing a sort of corresppondence between these parts; the resulting data structure (also called "zipper list").[11] *Collateral structures:* entities linked by a multicoupler.

Mooting system: system permitting complex alternatives to be studied indecisively, e.g., by use of collateral structures. *Creativity system:* mooting system with design (or text, etc.) facilities for creating complex entities indecisively.

VISUAL ORIENTATION DEVICES

Setting aside some of this intended generality, let us consider the visual modality, and screen tricks to keep the user oriented during complex screen transitions.

Orienters: pictographs that show a user where he can go spatially. *Rose* or *compass-rose:* pictograph showing possible "directions" in the space. *(Map:* diagram showing the structure of the space, and perhaps the user's current position.)

Ticklers: pictographs or messages that tell a user what he may do next. (Not sharply distinct from orienters.) A *maplet,* for instance, is a fractional map that tells immediate alternative moves, and possibly which way to some sort of "home"; a *blurb* is a writeup or title of something that may be gone to; a *jump-marker* is an element you point to to jump. *Function box* or *function rose:* pictograph indicating alternative functions which may be chosen. A *menu,* of course, is a textual listing of alternatives. (Note: for fast-roving performance on slow-filling screens it is desirable to put the ticklers up first in each new screenful, permitting the user to jump or move at once.)

By the careful crafting of media and screenworlds, using these devices with as much consistency and attractiveness as possible, we may encourage well-oriented mobility in our fantic systems. This is the point.

A FANTIC AGE

A variety of media await us, and which ones make the "best cuts" remain to be seen. (Conjecture: they will resolve to a very few.) The major mixable options I wish to point out here are: "hyper-" (nonsequential); text; pictures (2D or 3D); animation; overlays; complex roving; complex coupling of structures; interpenetration of presentational tissues; linkage of video and movies; linkage of special control yokes; linkage of special forms of viewing (3D, smellavision, etc.).

Except for the more expensive viewing situations - film, smellies and the like - all these may be presented in the Universal Console, and those which cannot may still be comprised within a generalized data and transmission structure built around such a Universal Console.

HUMAN INTERCOURSE IN A FANTIC ENVIRONMENT

With the spread of computer graphics and prospect of a Universal Console, computers may furnish backgrounds to human intercourse as varied as the blackboard and drive-in movie.

Till now, graphic systems have been built with an organization-based frame of mind. They have involved sitting in office-type chairs at desk-like furniture having screens and keyboards perched or attached. Home models, however, may well be built into lounging chairs, conversation pits, children's furniture, or even into bedlike environments. I am particularly interested in a backpack design for portable wear, one which would mirror the CRT image into a concave transparent faceplate or visor.

The active use of such systems in conversation should make possible whole new depths of communication, both factual and ideological. Engelbart's recent work has shown the power of text CRT systems to enable people to work together over the same materials. Pehaps this ability can be extended to miniature portable systems delivering graphics as well, *e.g., where people can exchange graphics or text through a quick umbilical connection between backpacks.* Suppose we can carry our favorite animated diagrams and reference works around, and exchange them and talk about them and manipulate them together. Does it matter how turtle-like such portable Houses of Intellect might seem? May we not actually come to understand each other better?

SHOULD SYSTEMS TALK?

From Weizenbaum's ELIZA program to HAL-9000 of Kubrick and Clarke's *2001,* there is a constant sense that "of course" we want talking computers. And while the problem of how to get them to talk back is investigated every which way, the question of *whether* computers should talk back seems never to have been examined, let alone posed. As there are far simpler methods of commanding system activity - e.g., light-pen thrust - conversation is by no means necessary. Moreover, it is not obvious that people will enjoy mechanized conversation; rather, it may be offensive, alarming, and a tribulation. Mischievous programmers get a kick out of writing programs that pretend to speak and understand, wise-guy programs that identify themselves as "I, the computer", insist on being talked to by typed input, are full of snappy replies, but don't really do much with the input. The capacity of such systems to offend and annoy has not been sufficiently recognized.

There obviously must be a method of sending new information to the user, and so sentence generation is unavoidable. *But that does not mean a system has to be a smart aleck,* or, indeed, to disguise the exact method of its sentence generation procedures. What I am getting at is: we should have standard ways to introduce systems, to know what kind of an entity you're communicating with. Who wants to be the goat of the Turing-test? The user is entitled to know if he is typing into a real sentence-parser or just a keyword trickster program. It's infuriating to have a program pretend it can understand you and then fail to parse eight consecutive input sentences. There ought to be a law against wasting people's time with this sort of silly program.

More generally, the spread of consumerism means people want to know who they're dealing with. As we program online systems that will involve innocent people, we had better think hard about ways to get the system's cards on the table (or heart on its sleeve) - and play neither coy nor god.

VICTORIAN REMARKS

I think that the Grand Corpus of our written heritage, chaotic and individualized as it is, is a precious substratum of our world. The new age of hypertext and hypergraphics should build on this tradition, rather than mush us into committee authorship and indifference to the past. In forging toward the Screen Future, and the creation of screenworlds we will love to live in, let us remember and esteem the traditions, scholarly mechanisms and arts that have worked so far, and build on them. And we must begin to worry about the problems of privacy, access for everybody, "what gets kept?", and dangers to the corpus once it is online.

CONCLUSION

The foreseen extension and unification of words, pictures, and control will be the apotheosis of "computer graphics"; but the sooner we resolve that field to a set of techniques at the service of explicit presentational goals, the better.

The ambitious terminology was presented to nail down crucial aspects and distinctions of an entirely new realm of human endeavor and experience, not cluttered with mumbo-jumbo from other creeds. The psychic engineering of fantic fields - adult's hyperspaces of word and picture, child's gardens of verses - is our new frontier. We must look not to Asimovian robotics and the automated schoolmarm and librarian, but to the penny arcade and the bicycle, the clever diagram and the movie effect, to furnish this new realm.

REFERENCES

1. Theodor H. Nelson, "Computopia and Cybercrud". In Levien (ed.), *Computers in Instruction,* the Rand Corporation, 1971.
2. D. Alpert and D.L. Bitzer, "Advances in Computer-based Education", *Science,* 167, 1582-1590 (1970 Mar).
3. Vannevar Bush, "As We May Think". *The Atlantic Monthly,* **176.1** 1945 July), 101-108.
4. Theodor H. Nelson, "As We Will Think". Proc. Online 72 Conf., Uxbridge, England, 1972.
5. -, "No More Teachers' Dirty Looks", *Comput. Decis.,* 1970 Sep.
6. -, "Getting It Out Of Our System". In Schechter (ed.), *Critique of Information Retrieval,* Thompson Books, 1967.
7. -, "Hypertext notes", unpublished, 1966-7.
8. -, "Simplicity Versus Power in User Systems: The Text Case", unpublished, 1972.
9. Paul Schilder, *The Image and Appearance of the Human Body.* Science Editions, paper, 1964, 63-70.
10. Sutherland, Ivan, *Sketchpad: A Man-Machine Graphical Communication System.* Lincoln Laboratory Tech. Report 296, 1963.
11. Theodor H. Nelson, "A File Structure for the Complex, the Changing and the Indeterminate". *Proc. 20th Nat. Conf, ACM 1965.*

A CONSERVATIVE VIEW OF THE COMMUNITY INFORMATION UTILITY

by LAURENCE I. PRESS

University of Southern California
Los Angeles, CA

The computer may be compared with the clock as an artifact capable of altering man's life and manner of viewing and experiencing the world. The growing literature on the community information utility (CIU) calls an analogy to mind. I am reminded of the clock in the cathedral of Lund, Sweden, where I once lived. It is a typical medieval clock that tells the time, date, courses of the sun and moon, etc. Furthermore, at noon (1 P.M. on Sundays) and 3 P.M. mechanical knights clash on top of the clock and three wise men appear in the wings to pay homage to Mary and Jesus while an organ plays "In dulci jubilo". Clearly the 14th century builders of astronomical clocks were interested in pushing their new technology to the hilt[1] and quickly putting as many applications as feasible on the town clock.

Today such clocks serve only as tourist attractions. We have developed alternative means for some of their functions, and are getting along without others, like clashing knights at 3 P.M.

Throughout the CIU literature one encounters appreciation for subject complexity and potential costs as well as benefits, yet there seems to be considerable consensus that the CIU should and will eventually be with us. I am uncertain of this conclusion, and uneasy with the haste often urged for its realization.

A common proposal is to establish a prototype CIU as a research vehicle. In the report of an excellent, comprehensive conference - Planning Computer Information Utilities (PCIU)[2] - we find "consensus" that "a well-designed, scientifically-evaluated prototype CIU would greatly reduce the long-term social risk".

Rather than building a prototype CIU as quickly as possible, I would advocate a *moratorium* on CIU construction until the year 2000. At a time of pressing need for our national resources, I am skeptical of building a system which will be obsoleted by changing technology, may provide unwanted or unnecessary services, or might alter man's way of living and experiencing the world in unpredictable ways. Deferring CIU development will give us a chance to gather new information (without a prototype) on technology, applications, and man's values.

TECHNOLOGY

Around the year 1500, Peter Hele of Nuremburg is said to have hit upon the idea of using a spring to furnish power for a clock and he created the first clocks without towers. By the end of the 16th century the domestic clock was introduced in Holland and England. Certainly by the year 2000 there will also be significant changes in information processing technology.

Without explicit impetus of a CIU project, research and development are proceeding in virtually all of the underlying techniques. We see efforts toward understanding the nature of programming projects and languages; alternative storage and communication technologies are being explored, processors are becoming faster, cheaper and more reliable; new forms of processor organization are under investigation, etc.

This all portends cheaper and more reliable systems in the year 2000,[3] with a higher probability of ever working (we have many examples of abandoned projects). A prototype implemented in the technology of 2000 would be organized differently than one of 1973; stories of using 1401/10 emulation mode on IBM/360s to run IBM 650 programs under 1410/650 simulator attest to the problems of getting locked into early design conventions.

The year 2000 will present not only improved technology for CIUs, but also alternatives to the CIU. E.g., many potential CIU applications in education and entertainment might be done via videotape cassettes, purchased or checked out from the public library, if costs of creating, duplicating, and playing videotapes were to fall drastically.

APPLICATIONS

Clocks have been built which predict eclipses and tides; show the movement of the earth, moon, sun and other planets; show the date, day of the week, zodiac sign, and season; depict a wide range of religious and other events; entertain and tell time.

Today many of these applications seem humorous. Obviously they were generally more important to 14th century man than to us, yet even then there were doubtless citizens who didn't care when the next eclipse would occur, or could predict seasons accurately enough for their purposes by alternative means. But since such predictions were of some value and *could* be made, they *were*. Alternative techniques (printing) have also appeared for many of these applications.

Like 14th century clockmakers, CIU designers are in danger of implementing applications that have insufficient justification, or which will become unjustifiable due to changing public needs or development of non-CIU means of performing them.

The major advantage to a moratorium in applications is that by 2000 the public will probably be better informed and familiar with the nature and functional capabilities of information systems, enabling partial decentralization of design responsibility for the CIU. A computer-literate public could be expected to write programs, specify individual applications, and make more reasoned political judgments on a CIU (including the possibility of forgetting the whole thing). If the CIU *is* a good idea, a computer-literate public would also save us the effort which Gilchrist (in PCIU) predicts would be necessary to sell the idea.

The importance of (right to) computer literacy cannot be overstressed. If the CIU becomes as important in our world as some feel (and others fear) it will, then will skill with and understanding of computers become proportionately important? Understanding will be necessary in order to fully participate in and utilize the CIU, while a lack of it would result in alienation. It would be one more facet of society upon which a person is dependent and yet ignorant - one more area in which to defer to the expert and the repairman.

Of course we would not remain passive during the next 27 years. A year's exposure to a terminal in an elementary classroom or math lab, with access to a library of programs for school work, teaching, and entertainment, might suffice for a child to gain understanding of what a computer can do - and also learn to take it for granted. Then he might be ready to learn a simple, specially designed (*not* BASIC, etc.) programming language - to help him internalize the nature of algorithms and the structure of a computer system. We can seek out and work with noncomputerized organizations and individuals in our present communities to see what applications, if any, they can use, and develop techniques for training them.

When and if a CIU finally comes into existence, users would be able to specify and program their own applications. Through incentives to share programs and charging for personal storage space, many user-developed programs would eventually end up in the public domain, probably in generalized form. Many of our current general-purpose systems, from report generators to translator-writing systems, have evolved in this "bottom up" fashion.

As an alternative, it may be argued that a prototype CIU could serve as a vehicle for testing the viability of applications. There are two major problems with this approach. The potential user is passive and is only in a position to accept or reject what is offered him, which (as Parker points out in PCIU, in a different context) puts him in a relatively powerless position.

The second drawback, and this point is more important since it transcends the area of applications, or even CIU's, has to do with scientific organization and method. By virtue of the cost and status of a prototype CIU project it is improbable that we would conduct more than one, and it would certainly be a major influence on all future community information processing. I feel that the topic is too important to be left to scientists, period, much less a single group associated with a prototype project. To quote Paul Goodman, this sort of big science risks "favoring a limited number of scientific attitudes and preconceptions with incestuous staffing". To put it another way, it is difficult to imagine men like Goodman having a place in the prototype CIU project.[4] Instead the moratorium should be used for decentralized (shoe string) science with studies by diverse investigators in diverse communities (the "example" city recurring in PCIU is Santa Monica, home of RAND and SDC. I would be surprised if citizens of adjacent Venice shouldn't desire different applications from a CIU, not to mention, e.g. Pittsburgh, KA. Of the 17 chapters in PCIU, 14 are by RAND employees or Californians).

Note that many investigations bearing upon CIU design and applications (such as those mentioned) may be carried out at relatively small marginal cost using today's timesharing systems to simulate a facet of a potential CIU.

A final danger in the applications area overlaps with the third topic, the effect of the CIU on man and his world. This concerns the tendency to define a need or application in terms of what is amenable to our technology. Though not meaning to criticize here the particular articles in PCIU, the education application serves as an example. The danger is that "education" becomes redefined in terms of what is implemented on the CIU. The question shifts from what constitutes a "good" or "relevant" education to what is feasible to program and deliver via a CIU.

A package of educational applications such as those outlined in PCIU would strongly alter our view of what constitutes education and would of course lead us to divert resources from alternative educational activities. If, as a society, we are to radically alter our concept of education and our educational system, we must consider a wide range of alternatives, not just the CIU. For instance, our moratorium period studies should be concerned with free schools, urban storefront schools, the British and Chilean school experiments, University Extension programs, etc., as well as with computer-assisted instruction.

MAN AND HIS VALUES

To return to the clock, let us refer to Lewis Mumford[5] who states that "The clock, not the steam engine, is the key machine of the modern industrial age". He goes on:

> "the orderly punctual life that first took shape in the monasteries is not native to mankind, although by now Western peoples are so thoroughly regimented by the clock that it is 'second nature', and they look upon its observance as a fact of nature. Many Eastern civilizations have flourished on a loose basis of time: the Hindus have in fact been so indifferent to time that they lack even an authentic chronology of the years. Only yesterday, in the midst of the industrializations of Russia, did a society come into existence to further the carrying of watches there and to propagandize the benefits of punctuality".

Mumford points out that the popularization of timekeeping in America is as recent as the middle of the 19th century. In other words, we have only recently learned that "time is money", and we are different people and live in a different world than if we had never learned this lesson.

Would a CIU have a major effect on the nature of man and his life? If so, are the changes desirable or undesirable? Let us consider three views. First, Thomas Watson of IBM,[6] who sees technology (not restricted to the CIU) as having a major, positive impact on man's life. He recommends three uses or goals for our technology: the improvement of men's lives, bolstering our economy to meet the challenge of international communism, and helping people in underdeveloped nations to improve their lives. While much has happened since 1960, when Watson wrote this, to shake our confidence (or better "faith") in the power of technology, no doubt similar views are held by many today.

As to the question of changes in what man is, Watson says "human adjustment to (technological change) should not be forced, rushed, or humiliating, but must be carefully considered and carried out". He obviously feels that there is no question but that the readjusted human, even if different, is better off.

A more conservative assessment is made by Herbert Simon,[7] who feels that it is fashionable today to overstress potential change from the computer or communication "revolutions", and who therefore tries "to show in what important respects tomorrow's megalopolis will resemble today's metropolis, and indeed, yesterday's Athenian polis".

Simon feels that, assuming man's physiological needs are satisfied, his floating aspiration level will keep him in hedonic equilibrium regardless of computers and communications. As to the nature of man, he states that:

"It is the truth, if not the whole truth, that the focal events and the climactic emotions in my life and my neighbor's are almost identical with those in the lives of Pericles, his neighbors and his forebears. Homer would find all the materials for his third epic in the morning newspaper: wars, floods, murders, shipwrecks, negotiations, births, deaths and marriages. Love, hate, curiosity, friendship, ambition, fun, pain were and are the substance of the human condition. Man is the significant part of man's environment; the nonhuman environment, whether the forest or the sea designed by nature, or the farm or city by man, largely defines the rules of a particular game within which meaningful human interaction takes place".

The third position is that technology has and/or will have a profound but devastating effect on man and his world. This viewpoint has been offered by D.H. Lawrence, Aldous Huxley, Jacques Ellul, George Orwell, Kurt Vonnegut, Samuel Butler, Lewis Mumford, and many other authors. I know of no story more directly relevant to the CIU than E.M. Forster's "The Machine Stops",[8] science fiction depicting the "ultimate" CIU.

The world has moved underground, where each individual inhabits his own room. All communication is electronic and all goods, services, and information are delivered to the rooms so one seldom leaves his room or sees the light of day (eventually visiting the surface of the earth is outlawed).

The inhabitants of Forster's world are physically changed. A woman is described as "a swaddled lump of flesh about five feet high with a face white as fungus". People are barely able to walk short distances, hold objects, and can no longer breath air. They are deluged with input and therefore fanatic about saving time and often "irritable". Direct experience of any sort repels them and they value only one thing: "having ideas". The inhabitants of the machine are well adapted. Artificial grapes with no bouquet, and images without nuance or expression, are "good enough". When the machine begins to fail, they readily adapt to putrid food and stinking baths, and when it finally fails, the sudden silence causes many heart attacks and great pain.

As foreshadowed by the title, the people are totally dependent upon their CIU and no one understands it. When it begins to deteriorate, they begin to deify and worship it, and when it eventually fails totally, mankind perishes.

Good, bad or indifferent? The answer is clearly not known, but Forster et al raise serious questions concerning the potentially negative effects of a CIU. What is the psychological cost of depending upon a perhaps poorly understood CIU? What of the psychological effects of decreased personal contact? Will we tend to further neglect our physical selves in favor of more (active or passive) information processing? What of the social development and "babysitting" function of conventional schools? Will we be transformed by information overload? Will we process ever more abstract information ever more superficially (one thinks of Thoreau[9] who admonishes us to read sparingly only "the best in literature...read as deliberately and reservedly as they were written")? Will the "rat race" force us to complete ever more, often irrelevant, instruction to compete with the Joneses?

The answers are not known, nor is it even possible to complete the list of questions. This high level of uncertainty and risk argues for prudence in the implementation of our technology.

Prudence is also suggested by an even more fundamental charactistic of our time. Our values as individuals and a society are in a state of violent flux. Even if we knew the likely effects of a CIU with certainty, we have no commonly accepted metric for judging it. In a discussion of the CIU concept my students were able to state several reasons why a CIU should promote family units and several reasons why it might tend to weaken the institution of the family. A straw poll showed them almost evenly divided as to whether the family *should* be strengthened.

CONCLUSION

CIU development should be prudent, ecological and decentralized. I suggest a moratorium on any large prototype CIU experiment until the year 2000, when we may re-evaluate our position. We may expect CIU and CIU-alternative technology to mature in the interim, and the general level of understanding of computers to rise, particularly if we work actively for that goal. Possibly our current value upheaval is temporary, a transition to a new, stable state, and society will be more sure of itself then.

I am not advocating that we sit back and passively watch profiteers take over and create a "vast wasteland" of the CIU, but an active interim period for the gathering of more information for the major decisions which we are not yet ready to make. During this period we should conduct diverse, decentralized investigations using current timesharing systems to simulate the CIU when necessary, as well as actively monitor and control such commercial ventures as do arise in the CIU realm.

A CIU by 1980 or by 2080 makes little cosmic difference (it will just be implemented by different people). We have often been imprudent and irresponsible with our technologies in the past, a lesson understood by the general public, if not by ourselves.

REFERENCES AND NOTES

1. The origin of the mechanical clock is uncertain; however, it is known that they existed by the 13th century.
2. H. Sackman and B.W. Boehm, *Planning Computer Information Utilities,* AFIPS Press, Montvale, NJ, 1972.
3. In PCIU, Nielson estimates a minimum cost of $100 per console per month using today's technology. Rough estimates for the cost of a prototype CIU for a city of 90,000 are 500 million to 1 billion dollars and 7 - 10 years time.
4. Goodman's article "Can Technology be Humane", NY Review of Books, 1969 Nov 20. This as well as his other work is most relevant to the topic of the CIU.
5. Lewis Mumford, *Technics and Civilization,* Harcourt, Brace and World, Inc., 1934.
6. T.J. Watson, Jr., "Technological Change", in *Goals for Americans,* Report, President's Comm. on Nat. Goals, Prentice Hall, 1960.
7. H.A. Simon, *A Computer for Everyman,* Amer. Scholar, 1966, 258-264.
8. In *The Eternal Moment and Other Stories,* Harcourt, Brace and World, NY, 1928. This story, as well as writings cited in this paper by Mumford and Watson, is reprinted in Arthur O. Lewis, *Of Men and Machines,* E.P. Dutton & Co., New York, 1963. The entire collection is recommended to those interested in the CIU.
9. H.D. Thoreau, *Walden,* The New American Library, New York, 1960. To attempt to view the CIU through the eyes of a man such as Thoreau (or Thomas Watson) is an instructive exercise.

SOME EXPLORITORY EXPERIENCE WITH EASY COMPUTER SYSTEMS

by Harold Sackman

The Rand Corporation
Santa Monica, CA, US

Like the alchemists of the Middle Ages seeking the magic key to transform base metals into gold, we have a new race of computer alchemists seeking the Universal Computer for Everyman. The magic key for the modern alchemist is the "easy" computer system. The belief is based on the seemingly plausible premise that an easy language, with an easy input keyboard, feeding into easy programs, producing easy outputs and simple displays, will make it easy for anyone to use--so easy as to be virtually idiot-proof, the ultimate goal.

The halo effect of the easy computer system extends its warm glow over the entire system development cycle. Planning, design, production, implementation, and operational use of easy computer systems should also be easy, and the payoff should be huge because of the vast economies of scale for easy systems when the potential market includes everyone.

The objective of this paper is to determine whether there are proven, easy ways to develop easy computer systems. The technique is to cite some examples of relative successes and relative failures, and some inbetween. Six brief examples are selected to illustrate the remarkable scope and diversity of easy computer systems. The last example raises a host of new problems in a new approach to public computer services, particularly for the underprivileged.

Dunlop[1] describes an "easy" management information system developed for top executives in a large corporation. This interactive system came complete with a human-engineered typewriter keyboard, easy language, special training sessions for the busy executives, with hardcopy output and online video displays. Most executives, after some initial fiddling with the system for various minor queries into the management data bank, had the entire terminal wheeled out and placed with the secretary outside the executive office. The executives weren't interested. Why?

Dunlop describes the executives' negative reaction as a kind of status shock. The executive sees himself in a high-status role. He is "people oriented" rather than "device oriented". Sitting and typing at a console terminal is his concept of what subordinates should do, not what the boss should do. The executive felt much more comfortable calling up the information specialist at the computer center, giving the specialist the information request over the phone, and getting answers back verbally or on hard copy. This easy system failed because designers did not anticipate culture shock for the prime user.

Let us turn to a long-term success story. Blackwell and Robertson (1973) have recently completed a survey of JOSS system users at Rand which provides us with a profile of these users. JOSS is one of the pioneering timesharing systems which first went into operation in 1963. The system was designed to solve modest computational problems involving arithmetic, algebra, trigonometry and logic. The user interacts with JOSS in conversational commands through a mobile typewriter terminal connected to the computer with a telephone line.

Blackwell and Robertson found that users liked the fast response time, simple language, convenient program storage, immediate hard copy, extensive math functions, low cost, and around-the-clock availability. The most-used applications for the Rand clientele included scientific calculations, statistics, simulation, accounting, gaming, scheduling, and demonstrations. Only one-third of the programs were written by users, and those ranged from 10 to 50 statements. Users were apparently hooked on JOSS, since most reported that a 1-week shutdown would hurt their work, and that they do not have readily available substitutes for JOSS. An intriguing finding was that personal trial-and-error was ranked first in learning JOSS and getting to use the system. The moral behind the success of this easy system is that it was developed as a limited, special-purpose system for relatively specialized users, not as a Universal Computer System.

The next example is taken from a study conducted by the author (1970) at the Air Force Academy on a sample of 415 cadets. The primary objective of this investigation was a comparison of timesharing versus batch processing in teaching introductory computer science. A byproduct of this study was some useful positive fallout on the attitudes of these cadets towards computers and computer programming. These attitudes are summarized in Table 1.

Positive Attitudes

- Understanding program concepts.
- Understanding the speed and power of computers.
- Insight into the structure of syntax.
- Learning how to debug and test programs.
- Appreciation and respect for computers and computer science.
- Programming is logical.
- Learning how a computer works.
- Appreciation of computer language.
- Batch and/or timesharing is easy.
- Successful communication with computers.
- Sense of triumph over computers.
- Understanding flow charts.

Negative Attitudes

- Inadequate access to computer facilities.
- Unreliability of hardware.
- Programming is too complicated.
- Poor support services.
- Computers are wasteful for many types of easy problems.
- Too much work, too time-consuming.
- Batch and/or timesharing are poor ways to use computers.
- Painstaking care and attention.
- Computer is an antagonist.
- Batch and/or timesharing are unreliable.
- Disappointment in not solving a well-understood problem after great effort.
- Antagonism toward computer language.

Table 1. Positive and Negative Student Attitudes Toward Computers and Computer Programming

A distressing finding in this study is that 26 percent of the cadets reported unfavorable attitudes toward computers as a result of their introductory course. This poses a major challenge for our educational institutions. Initial attitudes are difficult to change; introductory computer courses should not only teach technical content, they should also be oriented toward winning friends and influencing people for more effective lifelong use of computers.

Barmack and Sinaiko[2] reported user experience at the TRW Space Technology Laboratories in connection with their review of human factors problems with interactive graphic displays. The Culler-Fried system was introduced at TRW to permit engineers and scientists to work with interactive graphic portrayals of a wide variety of mathmatical functions. Over 400 of the technical staff received an indoctrination program. Records of system use indicated that less than 100 used the equipment for project work for ten or more hours over the ensuing half-year period. Why?

The major constraint was an underestimation of the mathematical skill required. The Culler-Fried system was not easy as it was supposed to be. The instruction manual was often difficult to understand. There was a lack of self-tutoring features for users at different skill levels and interests. Dissemination of system changes was inadequate. This example highlights the vital requirement to systematically try out the proposed system on representative users with representative problems before introducing it to a large and diversified user community.

The next example is from a pilot study conducted by the author[4, 5] comparing problem solving with and without computers for real-world problems. In this study, 19 subjects reported their experiences solving a computer and a noncomputer problem to test for similarities and differences in problem solving. The problems were significant projects or assignments perceived as being important for the subject's job. The data were collected through self-administered problem questionnaires.

The results showed the usual gripes over computer system performance: poor reliability, slow response time, poor documentation, inadequate software, ineffective training, poor diagnostic and error-correcting features, etc. However, in spite of these complaints, practically all respondents indicated a more favorable attitude toward computers as a result of their problem-solving experience. In essence, the computer helped them to get their work done faster, cheaper and better, typically for problems that could only be conceived and tackled with computers. The key to the attitude change was that computers were helpful where it really counted -- contributing to successful problem solving on the job. The moral for computer system designers is to develop computer services that people can effectively apply to important problems in their working environment, as opposed to expecting users to force-fit their diverse problems into the vendor's procrustean vision of universal software.

The last example concerns a new approach to a resistant problem -- getting computer service to the ghetto. The Tie Line Corporation in Los Angeles is a nonprofit foundation dedicated to facilitate the flow of information from citizens to community institutions, and from citizen to citizen via newsletter and computerized data banks. Tie Line is staffed by young volunteers (including some computer-trained personnel) who receive practically no pay. Their monthly news service lists extensive information on community agencies where individuals can get free help in seeking a job, medical and dental assistance, mental health support, drug rehabilitation programs, vocational training, senior citizen discounts, legal aid, etc. Distribution of this news service purportedly reaches some 10,000 subscribers, mostly nonpaying.

The computerized data bank is under development and is not yet operational. Tie Line plans to demonstrate prototype operations at the California Museum of Science and Industry at Los Angeles to "spread the word". Local universities donate computer time. The data bank consists of messages from individuals in the community asking for or offering almost any kind of "legitimate" goods or services. The Tie Line system is designed to store this information so that an individual with a particular need can be matched with the individual who has the resources to meet the need. The system is called PIE - Public Information Exchange.[6,7] A kind of realtime want-ad matching service.

The user initiates the cycle by communicating by phone with the information specialist at the data center. If the user offers a good or a service, it is tagged and stored in memory. If he has a request, the specialist searches the data bank and tells the user what, if anything, is available to meet it. If nothing is available, the request is stored in memory for subsequent checking against updated files. The service is mostly based on simple barter - a piano for a typewriter, a carpenter and auto mechanic doing work in kind for each other, baby-sitting in exchange for books. In deference to the prevailing economic culture, cash is also acceptable in exchange for goods or services. The idea is to help people help each other via interactive want ads.

Tie Line is too new to be evaluated. It is not economically self-sustaining at this point. They have not published their work in the literature. There are countercultural elements in this movement to "give computers to the community", and this movement is not restricted just to Los Angeles, but has counterparts in other major urban centers.

Tie Line is novel and interesting in its approach to users. The telephone is effectively the user's "console", not a tricky terminal. Recall, from Dunlop's example cited earlier, that top executives preferred the telephone/information specialist method of input over the keyboard terminal for queries to their management information system. We use this approach widely today with airline reservations. There is much to be said for leaving keyboards more complex than telephone receivers to trained and certified information specialists.

If the main objective is to hook the "ordinary mortal" user, as Tie Line is trying to do, why put a monkey on his back with a forbidding terminal and an arcane computer language when he can talk naturally to another person over the phone and get his information verbally or soon afterwards on hard copy if desired? The user has to be convinced he really wants and needs the "user-oriented" information system, whether he is an executive or a ghetto dweller, before he will even consider investing significant time and effort into finding out what it is all about. In the absence of knowledgeable and firm guidance from management, computer professionals, left to their own devices, have tended to use the worst possible and least representative examples of users - themselves - in designing "easy" systems.

The information specialist in Tie Line is the only one who interacts with the computer. This raises the problem of the concentration of information power with the information gatekeeper. Service is theoretically open to everyone. Information requests represent personal needs and personal problems. Barter is preferred over cash. The constituency (or market) is actively solicited and serviced with a monthly newsletter which keeps all up-to-date. The modus operandi of Tie Line is to hang loose and proselytize actively to build up the subscriber base. With this approach, the central data bank could potentially evolve into a realtime community information center with many ramifications in economic, political, and social spheres. (See Fig. 1 for ultimate scope of community services for PIE.)

It must be granted that the Tie Line concept is radically different from conventional approaches in trying to link computer services to mass personal use. If the concept works, it could conceivably become a kind of Computer Confessor in collecting, analyzing, and mediating personal problems in the community.

The Computer Confessor notion is fact, not fancy. Tie Line personnel have indicated that intermediaries such as ministers and social workers have placed requests for personal assistance, such as particular types of psychotherapy, for their "clients". PIE acts as a clearinghouse for matching such requests. If a potential match is found, the minister or the social worker screens the response (e.g., group therapy for alcoholics) and advises the individual whether he should follow it up with a personal call or visit. Tie-Liners believe this procedure maintains the privacy of the individual via his personal intermediary, and reduces the computer system requirement for data privacy.

The trick is not in the computerization, which is basically off-the-shelf technology, but in getting people to be willing to open up in return for useful leads to other people. The ghetto dweller can understand, and perhaps may trust this kind of information service, particularly if other avenues are blocked. However, the Computer Confessor, as with its "manual" precursors throughout history, is open to at least as much social abuse as it is to social melioration. Perhaps the moral of the Tie Line effort is that it is based on a moral rather than an economic approach to computer services for the community.

This anecdotal tour through six examples of "easy" computer systems has succeeded if it has demonstrated one point - easy systems for Everyman may be the hardest problem the computer world has ever faced.

REFERENCES

1. R.A.Dunlop, *Some empirical observations of the man-machine interface question,* Proc. Conf. on Management Information Systems, Carnegie-Mellon Univ., Pittsburgh, PA, 1968.
2. Joseph E. Barmack and H. Wallace Sinaiko, *Human factors problems in computer-generated graphic displays,* Inst. for Defense Analyses, Arlington, VA, Study S-234, 1966.
3. Harold Sackman, *Man-computer problem solving,* Auerbach, Philadelphia, PA, 1970.
4. Harold Sackman, *Preliminary investigation of real-world problem solving with and without computers,* Volume 1 summary, The Rand Corp., R-1205-NSF, Santa Monica, CA, 1973.
5. Harold Sackman, *Preliminary investigation of real-world problem solving with and without computers,* Volume 2, complete results.
6. Bob Polakov (Ed.), *Public information exchange: an introduction,* Tie Line Corporation, Santa Monica, CA, 1972.
7. Douglas Smith, *"Computer's use seen as free tool of public",* Los Angeles Times, 1972 Oct 22.

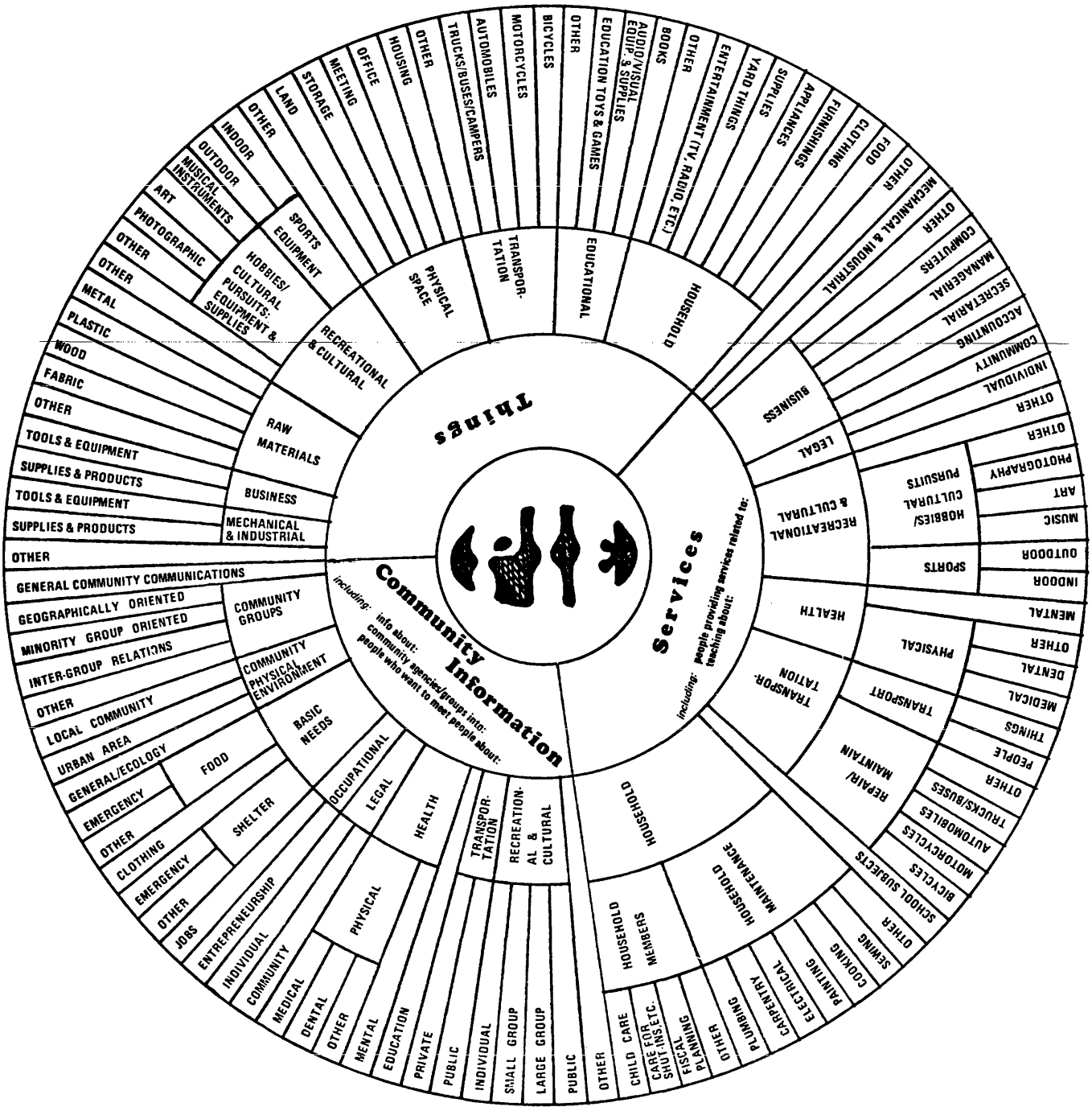


*Figure 1. Scope of Long-Range Tie Line Services

THE DESIGN OF 'IDIOT-PROOF' INTERACTIVE PROGRAMS

by ANTHONY I. WASSERMAN

University of California San Francisco Medical Center
San Francisco, California

DEFINITION AND INTRODUCTION

Over the past few years, the use of interactive systems has begun to shift from program development to a wide range of applications. Simultaneously, the users of interactive systems have begun to shift from programmers and others familiar with computer systems to those without a technical understanding of computer systems and programs. These trends seem likely to continue with the increased availability of multiprogrammed systems, the reduced cost of computer hardware, and the development of new application programs for retail sales, medical systems, law enforcement, financial analysis, and other tasks. This steadily increasing number of conversational applications programs will produce a growing number of computer users who are well-trained in their application areas without knowing how the programs that they use actually work. These technically "unsophisticated" users view the computer simply as a tool and are willing to use this tool only if it is easy for them to do so.

A considerable number of devices have been developed to take it easier for programmers to use interactive systems. These tools range from text editors and incremental compilers to interactive debugging aids and even "automated programmering" aids like Teitelman's DWIM (Do What I Mean).[1] For example, Mitchell has observed[2] that "efficient 'use' of the human is achieved by flexibility in the system" and has studied control mechanisms, data structures, and program dynamics to determine how to best create a smooth and functional human/computer interface.

Unfortunately, very few such methods have been designed for applications users. In fact, development of these methods has received very little attention in the open literature. Furthermore, most of the published material treats the entire subject from the standpoint of the efficiencies of system design and virtually ignores the end-user.

Yourdon accurately observes, though, that "(a)pplications programmers almost never consider the consequences of a system failure in the middle of processing in their program".[3] However, he fails to add that applications programmers rarely consider anything out of the ordinary happening during the processing of their program. As a result of this neglect, an unusual action by a user often results in abnormal termination of the program, loss of data, loss of time, and loss of confidence in the computer system instead of some form of corrective measure which could have prevented more serious consequences. Hansen, in discussing what he terms "error engineering", notes that:

> "A system must protect itself from all such errors and, as far as possible, protect the user from any serious consequences. The system should be engineered to make catastrophic errors difficult and to permit recovery from as many errors as possible ... the system must detect errors and let the user act on them, rather than simply ... terminating the run".[4]

A program which contains a thorough complement of these error-preventing and corrective measures may be said to be "idiot-proof",* i.e., it is designed to anticipate any possible action by its users and to respond in such a manner as to minimize the chances of program or system failure while shielding the user from the effects of such a failure. An idiot-proof program will continue to perform "intelligently" no matter what its users do. As a result, it can easily be used by the unsophisticated user and can assist him in using the program and the computer.

For example, a program checking input from a remote terminal must be prepared not only for correct or meaningful input, but also for a number of other possibilities, including:

(1) meaningless input, such as a string of special symbols;
(2) no input at all, possibly due to a broken connection, a broken terminal, or the user leaving the terminal without terminating the program;
(3) the user hitting the break key, the escape key, or another control character on his keyboard;
(4) a transmission error, which results in illegal characters being received.

A computer program which purports to be idiot-proof, then, must handle properly all of these eventualities.

As this example shows, the ability to handle all possible inputs and all possible system problems is quite difficult and cannot always be handled with the present set of programming tools alone. Many high-level programming languages presently in wide usage do not have the capabilities required to make all of these checks properly. Others, such as PL/I[5] and SNOBOL4[6], offer most, if not all, of the needed language facilities. Accordingly, some programming languages are badly suited for the design of idiot-proof interactive programs.

Likewise, certain computer systems lack the necessary characteristics for designing idiot-proof interactive programs. Because there is likely to be a large volume of input/output in relation to the amount of actual processing, very high speed data movement is essential. System architectures which severely limit memory access are generally inadequate and most certainly too expensive to allow for the necessary tests and still provide a quick response. Because of the possibility of communications errors, another useful hardware feature is a front-end communications processor which does error checking and correcting.

The complete design of an idiot-proof interactive program, then, requires not only a well-designed application program but also requires certain features to be present in the programming language and in the system software and hardware. Although the remainder of this paper will be primarily concerned with the design of applications programs and not with hardware and software selection, it should be remembered that no applications programs will be sufficiently idiot-proofed unless these other requirements are met, making the entire computer/programming environment well adapted for interactive usage.

*The word "idiot" is recognized to have a precise meaning in the field of psychometrics different from its meaning here.

BASIC DESIGN PRINCIPLES

There is one key point to bear in mind as the overriding factor throughout the development of idiot-proof programs. This point can be easily seen through Murphy's Law, which states that:

ANYTHING THAT CAN GO WRONG WILL GO WRONG.

We may slightly restate this law, for our purposes:

ANY ERROR THAT CAN BE MADE WILL BE MADE.

Stated as a rule for the programmer or system designer to follow:

BE PREPARED FOR ANYTHING THAT THE USER OR THE SYSTEM MIGHT DO.

This rule incorporates the two components required for complete idiot-proofing:

(1) checking for all user errors;
(2) making the system "crash-proof" and shielding the user from any system failures.

Before proceeding, it is necessary to make a distinction between 'idiot-proof' programs and 'user-oriented' programs. It is possible to create idiot-proof programs which severely restrict the format of legal input or force the user to change his normal way of thinking, failing to consider the user's needs. On the other hand, it is equally possible (and indeed common) to design a highly user-oriented program with application-directed mnemonics and flexible formats which assumes that the user is sophisticated in the use of computers, thus failing to idiot-proof. Idiot-proofing and user-orientation are then independent characteristics. It should be noted, however, that a truly user-oriented program should be idiot-proofed as an integral part of its design.

Although the specific tests and corrective measures needed to thoroughly idiot-proof a program are highly application-oriented, as well as being dependent upon the programming language features and the hardware configuration, it is possible to present some general principles. It is expected that the means of implementing these principles will vary greatly among programmers; therefore, they should simply be viewed as a set of guidelines for the creation of idiot-proof interactive programs.

Principle I - Provide a program action for every possible type of user input.

In practical terms, there should not exist any program statement which can cause an abnormal termination unless there is also a provision for trapping any errors which may result. As mentioned earlier, it is necessary to provide for meaningless and improper responses, or no response, as well as to handle proper input. In this way, appropriate action can be taken. Otherwise, the program would either terminate at the point of incorrect input or would fail as a result of the input or an internal error. These undesirable actions can be avoided by specifying default actions in the event that the user is unable to give the proper form of input after a reasonable number of tries.

As an example, assume that the user has typed in a line of input and that this line is properly transmitted to the computer. The program must now first determine whether or not the input was meaningful (as opposed to rational). Consider the case in which the user is typing in a hand of playing cards, say for the game of bridge[7,8]. There are a very large number of mistakes that a user can make: he can type in too few cards or too many, type in the same card more than once, or type in meaningless symbols. In all of these cases, corrective action must be taken, and the user must be given a number of tries to input the hand correctly. Whenever an improper response is made, a message should be produced indicating the nature of the problem, e.g. "THE JACK OF DIAMONDS APPEARS MORE THAN ONCE IN THIS HAND." In the event that the user is unable to input the hand properly in the designated number of tries, the program could deal out a hand to be bid, rather than trying to obtain the user's intended input. In this way, the program may avoid waiting indefinitely for the user, who may or may not eventually respond correctly. An example of such a dialogue is shown in Figure 1.

Principle II - Minimize the need for the user to learn about the computer system.

From the standpoint of the nonprogrammer, the computer is simply a problem-solving tool which can be used to reduce the amount of manual processing required in its absence. There is no need, and usually very little desire, on the part of this nonprogrammer to learn about the complexities of the computer or the myriad problems which could complicate his effective utilization of the tool. It is the job of the applications programmer to shield this user from these details. Ideally, there are only four items that the user should have to know in order to perform his work:

(1) how to log into the system;
(2) how to log off the system;
(3) how to request the program(s) to be used;
(4) what character or signal is required to transmit a response to the program.

Most interactive programs require much more from the user and thereby create situations which lead to systems errors and user frustration. For example, many interactive programs require the user, whether he knows it or not, to work with one or more files stored on a peripheral device such as a disk. It is essential that the user not be allowed to clobber the contents of such files. Thus, a user should not be allowed to update, delete, or even rename a disk file until a series of checks have been made to ascertain the legality of the instruction and the user's right to access records of that file. Ideally, the unsophisticated user will not even have to know that such disk files exist and certainly should not have to learn anything about the file structure of the computer system. For many applications, especially when file size is relatively small, it is desirable to keep at least two copies of the file online. In this way, a file that gets inadvertently destroyed can be automatically recreated from the duplicate file so that there are always two valid copies available. It is important to remember that all of these operations must be kept invisible to the user.

Along the same lines, the program must provide for automatic bookkeeping and regular cleanup of the user's working area. This process is necessary because the unsophisticated user regards the computer as a "black box" and is totally unaware of the problems of disk file maintenance and other similar, yet necessary, functions.

PLEASE TYPE IN THE EAST HAND

PLEASE TYPE IN THE SPADES... NINE 2 4 QUEEN EIGHT 10 THR 5-

HEARTS... GLITCH-

1) I DO NOT RECOGNIZE THE SYMBOL G
PLEASE TYPE IN THE HEARTS AGAIN... KIN J-

DIAMONDS... VOID-

CLUBS... SICKS J K FOUR-

2) I HAVE FOURTEEN CARDS.
WHICH ONE SHOULD BE REMOVED?
JACK OF HEARTS-

PLEASE TYPE IN THE WEST HAND.

PLEASE TYPE IN THE SPADES... NINE 2-

HEARTS... K X X-

DIAMONDS... J 10-

CLUBS... Q 8 6 5-

3) I ONLY HAVE 11 CARDS.
WE WILL HAVE TO BEGIN AGAIN.

PLEASE TYPE IN THE SPADES... 9 2-

HEARTS... EIGHT 4 3 KING-

DIAMONDS... J 10-

CLUBS... Q 8 6 5-

4) I ONLY HAVE TWELVE CARDS. PLEASE TYPE IN MY LAST CARD.
JACK OF DIAMONDS-

THEJACK OF DIAMONDS APPEARS MORE THAN ONCE IN THIS HAND. WE WILL HAVE TO BEGIN AGAIN.

PLEASE TYPE IN THE SPADES... 2 NINE-

HEARTS... K 8 4 3-

DIAMONDS... J 10 4-

CLUBS... EIGHT Q 6 5-

Figure 1. An idiot-proof dialog for input of a bridge hand.
8Source - reference 7, 221-225)

Principle III - Provide a large number of explicit diagnostics, along with extensive online user assistance.

The need for meaningful diagnostics cannot be overemphasized. A clear reply to the user's error will often prevent the user from making the same mistake again. Since the typical user is not a computer expert, messages must be expressed in some form that he can easily understand. Thus, a message such as "ERROR 23 -- ABORT" has no meaning for the user; neither does "DIV BY ZERO AT 32774". On the other hand, the user can easily understand "ERROR - ILLEGAL COMMAND" or "ERROR - UNRECOGNIZABLE NUMBER - PLEASE TYPE A NUMBER BETWEEN 1 and 10". Referring to Figure 1, it can be seen that specific diagnostics (numbered at the left side of the figure) are generated to handle the single problem of inputting 13 cards.

An even better means of minimizing user error is allowing the user to type "HELP" or "TEACH" at any point during the program. Properly implemented, this mechanism can be used to instruct the user as to the proper form of input at any given point during the running of the program. In this way, a user can learn about new or rarely used features of a program without making many errors first and without having to log out and leave the terminal to refer to a manual. Error prevention and automatic instruction, while not actually falling into the category of idiot-proofing, are important for making unsophisticated users feel at ease with a conversational program.

HANDLING THE KNOWLEDGEABLE USER

Although interactive programs must be protected against inexperienced users, they cannot assume that all users are incompetent, else users will become dissatisfied with the program as they become more educated in its use; e.g., the explanations of program usage that typically are given to the first-time user become boring to a continuous user. Yet those explanations are necessary, as indicated by Principle III. The need to provide for a broad range of users' abilities and experience leads to other principles which take recognition of this fact.

Principle IV - Provide program short-cuts for knowledgeable users.

In an ideal situation, it would be nice to have the program develop a profile of the user's intelligence, based on the number of mistakes he makes while using the program or the number of times he has worked with the program. In this way, shorter messages could be used automatically for the intelligent user.

More realistically, one should provide at least two modes of operation, which can be termed "QUICK" and "NORMAL". "QUICK" mode allows for successful development of a large interactive system and extensive testing by the designer. In addition, a user familiar with an interactive program no longer wishes to see lengthy messages, such as expanded diagnostics, when the mere knowledge that an error has occurred is sufficient. Such users can request the "QUICK" option at the beginning of their program run (alternatively at any time during the run).

Principle V - Allow the user to express the same message in more than one way.

The unsophisticated user cannot understand seemingly unnatural restrictions imposed upon him by computer programs. Many conversational programs operate on the assumption that input must exactly match some internal quantity. Thus, a quiz on state capitals will accept only the proper spelling "TALLAHASSEE", when many of those who know the answer are unable to spell it correctly. Unnecessary delay is created by failure to accept a reasonable facsimile of the correct answer. For many applications, the first couple of letters of a word are sufficient to identify a user response, particularly in a game playing environment. By scanning the first distinguishing letters of the input and determining the result accordingly, the effect of misspellings can be minimized. This approach is especially successful when the program expects a short response in a fairly rigid format; it would be of limited help in programs where the range of meaningful inputs is less rigidly constrained.

A good example of this type of flexibility is interpretation of a number which a user types in. All of the following are mathematically equivalent and should be so treated by the program:

2E0 2.00000 200.0E-2 2

The GET LIST statement of PL/I will treat each of these inputs in the desired manner. If there is a nonnumeric character in the input stream, the CONVERSION condition will be raised and can be trapped. In most programming languages, however, the only way to achieve the effect of the GET LIST statement is to scan the input string sequentially.

There is also little reason to repeat yes-or-no questions when some answer is given. There exist programs so rigid that they will not accept the input " YES" (with leading blank) as distinguished from "YES" and will repeat the question, requiring the proper input to be in the proper columns. It is better to assume that any input beginning with the letter "N" signifies no, without regard to preceding blanks the characters following the "N". Thus, "NO", "NYET", and "NOPE" are treated identically; any input not beginning with "N" is treated as signifying "YES" including, for example, "DA", "SURE", and "CERTAINLY". (Unfortunately, "NATURALLY" is a counterexample).

SOFTWARE AND HARDWARE REQUIREMENTS

While these five principles are important, complete idiot-proofing requires certain features to be present in the programming language and in the computer system. Unexpected program errors or hardware failures can occur at any time and there must be adequate protection against these contingencies. If the programming language has no provision for checking and trapping arithmetic overflows or invalid array subscripts, then it is possible for the user to cause the program to fail unexpectedly, to lose all or part of his work, or to receive a meaningless message. Likewise, the programming language should have provision for checking a real-time clock, so that a user failing to respond can be prompted and eventually logged out.

Unfortunately, many high-level programming languages lack these necessary features for complete idiot-proofing, yet it is generally impractical to develop such programs in assembly language, even though most contain the necessary features. AN Standard FORTRAN[9], for example, does not contain any of the necessary features, nor does Revised ALGOL 60[10]. They do exist, though, in extensions to these languages designed to run on computer systems with strong interactive facilities[11,12]. These languages contain statements which allow program control to be transferred to any specified location when an error condition exists. In addition, their file handling statements include the ability to test for failure to open files, trying to read beyond the end of a file, and inability to read and write on files.

Two languages which have most of these desirable features are PL/I and SNOBOL4. PL/I, for example, has over twenty ON-conditions, which can be set to check for computational conditions, input/output conditions, and other similar conditions, including a programmer-defined condition, all of which allow for control of program flow following the interruption of the program by the occurrence of one of these conditions. SNOBOL4 allows the programmer to associate a transfer of control with almost every statement based upon the success or failure of the action specified by the statement. Thus errors may be trapped by inclusion of a failure branch specification with every SNOBOL4 statement which might cause an error. Although these two languages are quite different in structure, range of applications, and means of handling undesirable program conditions, they illustrate that it is indeed possible to achieve a high degree of success in building idiot-proof programs.

As noted earlier, implementation of these features often require certain characteristics of the computer hardware. Certain computer systems are simply not designed for interactive use, even a sophisticated operating system can not overcome bad architectural design. For example, the operating system overhead for a system with only a single port to memory is so great that it is generally uneconomical to use it for interactive applications, especially if an attempt is made to provide rapid response as well. For many interactive applications, the computer system will be I/O-bound rather than compute-bound, so that improved memory access results in improved system performance. For this reason, the rate of information flow is the most important measure of a system's ability to handle a large number of conversational users. File updating, information and retrieval, and nonnumeric applications in general require less "number-crunching" than information transfer. This fact becomes even more important when one is working with displays, rather than teletype terminals, because of the high speed needed to fill the screen.

It should be clear that there are significant trade-offs involved in designing idiot-proof interactive programs. A good deal of programming effort, a sizeable amount of core storage, and considerable processing time is required to idiot-proof a program thoroughly. The extra cost of development, storage, and computing must be measured against intangible quantities of ease of program usage and user satisfaction. With the steady decrease in cost of computer time and memory, and the advent of the relatively low cost computer mainframes and peripherals, it would appear that the balance is swinging in favor of the user.

CONCLUSION

There is a serious need for improved facilities for the design of idiot-proof interactive programs. With a growing number of non-programmers using computers, development of comfortable man-machine interfaces will outweigh many traditional considerations in the overall creation of interactive programs.

The need to pay more attention to user needs will also affect the roles of computer professionals. Systems analysts will have to study applications more from the user's point of view than from the programmer's point of view, which will undoubtedly result in increased work for the programmer. Data processing management will have to include additional factors and modify traditional considerations in selecting computing equipment.

This new factor in applications program design will require the modification of existing programming languages and eventually will lead to the development of new languages designed to make it easier for programmers to write conversational programs. Many of the kinds of routines needed for idiot-proofing, such as scanning lines of input, interpreting typed numbers, handling user-generated interrupts, and preserving the integrity of user files, should be written and placed in program libraries.

Once these facilities exist generally, it will be far easier to design and program for unsophisticated users. No longer will half (or more) of the code of interactive programs be devoted to handling explicitly the possible problems that regularly occur with conversational programs. It is only at this point that much of the user resistence to computers can be overcome and the computer can be made an effective tool for all.

REFERENCES

1. Teitelman, Warren, "Automated Programming - the Programmer's Assistant", *Proc. 1972 Fall Joint Comput. Conf.*, 917-921.
2. Mitchell, J.G., "Design and Construction of Flexible and Efficient Interactive Programming Systems", Ph.D. Thesis, Carnegie-Mellon University, 1970.
3. Yourdon, Edward, *Design of On-Line Computer Systems,* Englewood Cliffs: Prentice-Hall, 1972.
4. Hansen, Wilfred J., "User Engineering Principles for Interactive Systems", *Proc. 1971 Fall Joint Comput. Conf.*, 523-532.
5. *IBM System/360 Operating System PL/I (F) Language Reference Manual,* Form GC28-8201-3. White Plains: IBM Data Processing Division, 1971.
6. Griswold, R.E., Poage, J.F., and Polonsky, I.P., *The SNOBOL4 Programming Language.* Engelwood Cliffs: Prentice-Hall, 1971.
7. Wasserman, Anthony I., "Achievement of Skill and Generality in an Artificial Intelligence Program", Ph.D. Thesis, University of Wisconsin, 1970.
8. Wasserman, Anthony I., "Realization of a Skillful Bridge-Bidding Program", *Proc. 1970 Fall Joint Comput. Conf.*, 433-444.
9. *USA Standard FORTRAN,* Amer. Nat. Standards Inst., ANSI X3.9-1966, New York, 1966.
10. Naur,P. (ed.), "Revised Report on the Algorithmic Language ALGOL 60", *Comm. ACM,* **6**, No. 1, 1-17 (1963 Jan).
11. *XTRAN Reference Manual,* Ann Arbor: Com-Share, Inc., 1970.
12. *Burroughs B6500 Information Processing System Extended Algol Reference Manual,* Detroit: Burroughs Corp., 1970.

DISPLAY TERMINALS CAN HELP PEOPLE TO USE COMPUTERS

by GABRIEL F. GRONER

The Rand Corporation
Santa Monica, CA

EXPECTATIONS OF USERS

A user likes to think of a display-terminal screen as a working surface upon which he can create objects and with which he can view and easily manipulate these objects. He is interested in such objects as text, line drawings, data clusters, and natural scenes. In reality, such objects may be black-and-white or in full color, and may be 2-, 3-, or higher-dimensional.

A user likes to take natural actions, i.e., actions that do not require much learning and that are appropriate for the object of interest and the function to be performed. For example, a user might like to create text by writing or typing, to create drawings by sketching, and to "create" a natural scene by taking a photograph or by pointing a television camera.

A user likes to take direct actions. He usually prefers pointing at an object rather than typing its name, and he prefers to move an object directly rather than to type a command that says "Move object A to position x,y".

A user expects his working surface to be responsive. When he attempts to move an object, he expects it to follow closely. In general, he likes things to happen immediately, but he usually realizes that the time required to fulfill a request should be roughly proportional to the complexity of the request.

A user likes his display-terminal's working surface to be attractive, large yet mobile, legible, reliable and - perhaps most of all - as accessible as a telephone, a typewriter, or a desk calculator. Finally, most users do not want to be bothered with extraneous details like complicated identification procedures, programming languages, computers, and terminal/computer interfaces.

We will examine a few general application areas to clarify user expectations and requirements, and the performance characteristics of current display terminals.

TEXT-ORIENTED APPLICATIONS

Alphanumeric display terminals, i.e., those that can display only characters, are useful in a variety of text-oriented applications in which both the information displayed and the user's interaction vary over wide ranges. During the preparation of documents such as reports and computer programs, the user enters a large amount of text and, during initial entry and afterward, manipulates it by adding, deleting, changing, and moving individual characters and blocks of text. In data-entry applications, the user, who may be prompted by computer-generated questions or a displayed form, enters data values which he verifies visually and which the computer may validate by applying sets of tests. In this case, as when editing a document, the user can work most efficiently if he can move easily in two dimensions to fill in or change items on a form.

In still other applications, such as computer-aided instruction, retrieving stored files, and monitoring behavior of a computer program or a realtime process, the computer may display and quickly change natural scenes and large amounts of text on the display screen, but the user's actions may be limited and rather simple.

Readability

Applications and operations such as these imply a number of requirements, the most fundamental of which is that displayed information should be readable. The screen should be bright enough that the characters can be seen in normal light. The contrast between the characters and their background should be sufficiently high for the smallest characters to be resolved at the user's normal working distance from the screen.

Readability also depends on the character font. Characters are most legible if they are constructed from a number of line segments or curves and if the height of a character is between 7 and 10 line widths. In the more usual case where a character is formed as a dot matrix, 3 dots wide by 5 high is barely legible, 5 by 7 (which is typical) is good, and as the matrix gets finer, a character's legibility and appearance improve.

Character Sets

Just as when using a typewriter or keypunch, the required character set depends on the application. Upper-case letter and numbers are usually adequate for data entry; document preparation may require lower-case letters in addition; still other applications may require special mathematical symbols or even italics. In practice, a 64-character set comprising upper-case letters and numbers together with some mathematical symbols and punctuation marks is standard; in some cases a 96-character set, which includes lower-case letters and some additional symbols, in addition to the basic 64, is available.

The Number of Displayed Characters

The total number of characters that can be displayed on the screen is also an important consideration. A number of lines of text are necessary to provide a working context, particularly when preparing programs and reports, but also when entering data on a form. 10 to 30 (with about 20 typical) lines may usually be displayed, but some terminals provide for as many as 40. Screens that are 72 to 80 characters wide are usually adequate for programming applications. Report preparation requires screens that are 60 to 80 characters wide. Screens only 20 to 40 characters wide are often adequate for data entry and other applications.

An additional way to provide the required context is to "scroll" or move the text up and down the screen in such a way that when text is deleted at the top (or bottom) new text is displayed at the bottom (or top). Some terminals not only provide for scrolling, but also enhance this by including memories sufficiently large to store more text than can be displayed on the screen without communicating with the computer.

Speed of Terminals

The time it takes a terminal to display a given amount of text is important in several ways. Most, but not all, current display terminals use a CRT (cathode-ray tube) display screen that is coated with a low-persistence phosphor. An image is changed by drawing a new one while the old one quickly fades. A consequence of this is that a fixed image must be "refreshed" (i.e., redrawn) 30 to 40 times a second simply to maintain it. This works well if the terminal is fast enough to redraw all displayed information at this rate. However, if it is not fast enough, refreshing does not occur often enough to prevent the image from fading, and the result is a very annoying flickering effect.

The terminal's speed must also satisfy users' requirements for responsiveness. When a user causes a small amount of text to be displayed, he expects it to appear instantly (i.e., in one refresh cycle). Even when a user causes an entire new screen of information to be displayed, he will not tolerate more than a few seconds delay if the information is already available to the terminal. For scrolling to be acceptable, text should be redisplayed at a new (vertically upward or downward) position at one-second or shorter intervals.

Establishing a Text-Entry Position

There are several techniques for establishing a 2-dimensional typing position on the display screen when the user creates and edits text or enters data on a form. All involve moving a marker imaged on the screen, called a cursor, which indicates the position of the next typed character. A cursor usually looks like an underline and, on some terminals, blinks.

A hardware-based automatic cursor-positioning technique that aids in filling out forms moves the cursor from one data-entry field to another as the user completes the form. Another common cursor-positioning technique, which is independent of any displayed text, uses a set of keyboard keys for moving the cursor left, right, up, down. When the directional keys have a repeat (continuous operation when held down) mode, they can be used to position the cursor quickly and easily. Other devices such as a trackball (a mounted, rotatable ball) and a joystick (a stick-like device which has one end held fixed) cause a cursor to be moved when they are manipulated. Finally, pointing devices such as a light pen (a stylus for pointing at the screen), a tablet stylus (a stylus used in conjunction with a horizontal electronic tablet), or a touch screen (a screen at which one points with his finger) can be used for positioning the cursor with a single, direct action.

Text Editing

Once the user has positioned the cursor, he may edit the text by typing a series of special editing-control keys and character keys. The simplest editing feature provides for typing over (and thereby changing) existing text, but some terminals also provide for inserting, deleting, and erasing text.

When inserting, the new text is entered in place and the text to the right of the cursor position is shifted to the right of the new text. Some terminals limit insertion to one character at a time and simply make the line of interest longer (possibly "wrapping it around" onto an automatically inserted, otherwise blank, line); some provide only for character and line insertion. Other terminals have no such limit, but rather maintain the line margins by automatically moving text from one line to another in all following lines as the user types.

sions), in the layout of forms, in the examination of stick-and-ball chemical molecules from any aspect, and in the exploration of such figures as antenna radiation patterns. This type of application requires convenient techniques for describing the desired manipulation as well as quick response to a manipulation request. Other applications that require a quick change in the displayed information are those that involve observing an event as it occurs in actual or simulated "real time". These include watching the behavior of a simulated mathematical model, participating in an interactive simulation such as driving a simulated car on a simulated highway, using a simulated air-traffic-control or other radar-like screen, or monitoring performance of a real system.

The most demanding applications for graphic terminals are those that involve both complex diagrams and a high degree of interaction. Computer-aided design, for example, may require the user to point, to drag figures from place to place, to sketch, to type, and to perform various other functions on a variety of diagrams. Other applications, such as abstracting a line drawing from a natural scene by sketching, may involve similar user actions, but with reference to a photograph of terrain or of a human face, an X-ray, or an electronmicrograph.

Comparison of Alphanumeric and Graphic Terminals

Before describing the requirements for graphic terminals it is useful to clarify the difference between alphanumeric and graphic terminals. Some graphic terminals are simply alphanumeric terminals with a graphic feature added on, but usually even the *alphanumeric* aspects of the two types of terminals are different.

In the case of the alphanumeric terminal there is a fixed set of (say 20 by 40) positions where characters may be displayed. Thus, the terminal need only store the type of character (including blank) to be located at each of the (say 800) positions and store a description (e.g., a dot matrix) for each character in its character set. The fixed format is also compatible with a fairly simple display generator and straightforward cursor control and editing features.

A graphic terminal, on the other hand, has a large number (e.g., 480 by 640 or 1024 by 1024) of addressable points. Each point can be the center of one or more characters, the start of a line segment, or the end of a line segment. Because of this, images (even if they are only of characters) for graphic terminals are stored as a series of descriptors. This complexity results in larger storage requirements, more computer-to-terminal data transmission (and transmission time), more complicated display generation, faster display generation required to assure responsiveness and lack of flicker, and consequently a higher cost for a graphic terminal than for an alphanumeric terminal. Furthermore, because characters can be located at arbitrary positions on a graphic display screen, hardware-based cursor control and editing features may be impractical.

The alphanumeric terminal requirements described above (with regard to brightness, contrast, character, readability, possible gray tones, and color) apply to graphic terminals as well. The lack-of-flicker and display-speed requirements also apply, but are more stringent in the graphics case because here the images can be both more dense and more complex.

Deletion is the opposite of insertion; i.e., text is moved to the left and upward when characters are deleted. For any given terminal, deletion usually has the same restrictions as insertion. Erasure, unlike deletion, replaces characters with blanks; it has a starting position (the cursor position) and a specified extent such as a single character, the rest of the line, or the rest of the displayed text.

Pseudo Graphics

Alphanumeric terminals can display pseudo-graphical information in addition to text. For example, if one wishes to plot a graph, vertical and horizontal "lines" can be formed from the characters "1" and "-", and "curves" can be formed from the characters, singly or in columns. The resulting curves are rough not only because their points are unconnected but also because their points can be located at only the usual character positions. Some alphanumeric terminals have a "limited graphic capability", which usually means that they can draw horizontal and vertical lines or can display dots at a higher density than the normal characters or both. This is handy for displaying graphs, tables, forms to be filled in, and a variety of other figures.

Arbitrary Pictures

Another technique for displaying forms and other fixed figures is to use pictures rather than computer-generated versions. Television-based and plasma panel terminals provide for displaying pictures simultaneously with computer-generated text and line drawings. The pictures may be arbitrarily chosen natural scenes; hence this capability is useful in computer-aided instruction and other applications where arbitrary visual reference material is helpful.

GRAPHIC-ORIENTED APPLICATIONS

Graphic terminals, i.e., those that can display line drawings as well as text, have a wide range of applications, costs, and capabilities. The simplest applications treat a graphic terminal as if it were an alphanumeric terminal.

Applications in which the terminal is used primarily to examine, but not manipulate, computer-generated images require very little interaction on the part of the user but may place heavy demands on the terminal. In some such applications, a set of curves may be displayed that represents the solution to an engineering design problem, or a computer- or man-generated diagram may be displayed to give the user a quick look before he has it drawn by a high-precision x-y plotter. Other applications require a high density of displayed points and many gray levels or color, for studies of image-processing techniques or for the display of solid figures, i.e., computer-generated figures that appear to be artist's drawings in that they are shaded and give an impression of solidity.

The manipulation of diagrams is important in such applications as signal and data analysis (where waveforms and other data are examined in various representations, scales, and dimensions), in

Line Drawing

Lines, like characters, should be legible and pleasing. Some terminals move a phosphor-exciting electron beam to produce a continuous, uniform line; others represent a line as a sequence of dots or horizontal and vertical line segments. Dots should be small and close together to avoid a string-of-beads effect; line segments should be small to avoid a staircase effect. Line widths and the contrast ratio between a line and its background should be chosen to provide for resolving nearby characters and lines.

The user can know that different lines have different meanings by providing for several types (e.g., solid, dashed, and dotted), for colors, for several line brightness levels, or a combination of these. Brightness levels should be chosen to be distinct, and this requires brightness ratios of at least 2 to 1. Because more than about four levels is not particularly useful from the user's viewpoint, it is better to maximize the brightness ratios (given a particular brightness range) than to produce more levels.

In order to construct truly arbitrary line drawings, it must be possible to draw a line between any two points on the screen. Actually a terminal provides only a finite number of addressable points, restricting the allowable starting and ending (and sometimes intermediate) points of lines. Usually a range of 500^2 to 1000^2 is adequate, and sometimes a density as low as 250^2 is sufficient. However, where curves must appear very smooth, and there is not a special hardware curve generator, or in applications such as image processing and drawing solid figures, considerably more addressable points may be required.

Figure Manipulation

In highly interactive graphic applications, the user may have the ability to "drag" and sketch figures as well as to point and type. This requires an input device that is continuously tracked as it is moved over the display screen or associated surface; it also requires a terminal or computer that can handle the tracking. In dragging, the dragged figure should follow the motion of the user's input device, and so it must be continually redisplayed at its new position. In order for the figure's motion to appear continuous, it should be redisplayed many times a second. When the user is sketching, a line, composed of dots or short segments, may be displayed as if emanating, like ink, from the moving input device. Thus sketching also requires a responsive display terminal that can quickly update a small portion of the displayed image.

Some applications deal with such complex figures, or must manipulate figures at such a high rate, that functions which could normally be performed by software must be performed by display-terminal hardware. One such function, called clipping or scissoring, cuts off a line at specified (or display-screen) boundaries even though its description indicates that it extends outside these boundaries. The extension could appear in an unexpected place on the screen or could interfere with another figure if the terminal did not have this feature. Windowing, which makes use of scissoring, provides for filling one rectangle (possibly the entire screen) with that portion of a figure that lies within another rectangle. Zooming provides for sequential windowing with a set of consecutively smaller or larger concentric rectangles. Finally, some terminals can rotate figures in two or three dimensions.

HARDCOPY

Often when one leaves a display terminal, whether to think or to let someone else use it, he likes to take away with him reminders of his working session. A device that produces a hardcopy (i.e., paper or film) representation of what appears on the terminal's display screen can therefore be very handy.

A hardcopy device for use with an alphanumeric terminal can be a small printer. One type of graphic hardcopy device recreates the display-screen image on a small CRT and then reproduces it on paper or film using photographic or xerographic techniques. An x-y plotter is another common, but slow-speed, graphic hardcopy device.

CURRENT DISPLAY TERMINALS

A number of display technologies are now available. The oldest, based on the random-access refreshed cathode-ray tube (CRT), provides pleasing, readable images that can be quickly displayed and changed; but complex images can flicker, and these terminals require a storage mechanism and are expensive. Television-based terminals are inexpensive, present flicker-free images, and can display natural scenes, gray tones, and color; but they are slow, require a considerable amount of storage, and images are not of high quality. Storage-tube terminals display high-quality (but dim) images without flicker and without requiring storage; however, because they create and erase images slowly and erasure is not selective, they are not suitable for highly interactive applications. Plasma-panel terminals, becoming available only recently, present projection of slides; they are slow, however, and resolution is not adequate for some applications.

A number of input devices are available for entering text, pointing, moving objects on the display screen, and sketching. The most popular are the keyboard (which is a component of nearly every terminal) and the light pen. Graphic tablets, which can be high-data-rate, versatile devices (e.g., they can be used with a storage tube, whereas a light pen cannot), are now becoming more widely used than previously because of reduced costs. Touch screens are the development stage.

A number of alphanumeric terminals are available, at less than $5000, that display easily-read characters, provide adequate textual context, and have good built-in editing features. Most are video-based. Graphic terminals are available in a wide range of capabilities. The high-performance terminals use random-access refreshed CRTs. Although such terminals cost more than $50,000 not long ago, several are now available for $8000 to $15,000. The medium-performance, low-priced (under $10,000) terminals are video-based or use storage tubes. Several terminals and terminal systems include minicomputers and so can act as fairly powerful, stand-alone information processing devices.

REFERENCE

1. Groner, Gabriel F. *"A Guide to Display Terminals That Enhance Man/Computer Communications,"* R-1183, The Rand Corporation, Santa Monica, CA, 1973 March.

THE COMPUTER IN LEARNING-THE ORDINARY MORTAL

by Alfred M. Bork

University of California
Irvine, CA, US

The student in a computer science class, particularly an advanced class, can put up with many idiosyncrasies in the computer. But he or she is hardly an "ordinary mortal" in terms of computer usage. However, the student of physics, sociology, art, or English, who employs the computer as part of the learning experiences in his class, is certainly an ordinary mortal using computers.

It is with such mortals that I am concerned. The students using computers to learn about subject matters other than the computer itself. Much of my own experience is in physics classes,[1] where we have at Irvine used the computer in a variety of modes during the last few years.[2] But hopefully the experiences I have had can be generalized to other areas. I will discuss the following aspects: batch versus timesharing, terminals for mortals, timesharing systems, editors, and responsive languages. Mostly I will compare currently available items, although I will make some suggestions, too. While I have students in mind as my ordinary mortals, I believe some of these ideas apply to other mortals.

BATCH VERSUS TIMESHARING

I will offer no arguments, but simply claim categorically that timesharing use causes less pain to the casual student user than does batch. Many arguments for batch strike me as being in the "sour grapes" category, attempting to justify it because timesharing is unavailable or weak. There are clearly some intermediate cases; a minicomputer, used by a single person, is often closer to timesharing than to batch, and the "personal computer" of the future moves even further here. But quick interaction is educationally superior to slow interaction.

TERMINALS FOR MORTALS

For the student timesharing user, the terminal almost *is* the computer; he or she works almost entirely through the terminal, so it helps to shape attitudes about the computer process. A well-designed, well-working terminal which is a joy to use causes pleasure and increases student learning.

Unfortunately the factors which usually determine the choice have little to do with increasing pleasure; the arguments are usually financial and frequently wrong. The most commonly selected student terminal is the Model 33 Teletype. This device is, I believe, noisy, slow in printing, unpleasant to use, and often in need of repair; I don't believe it should *ever* be purchased for student use. Even economically it is questionable, given the necessity for frequent repair, and given the fact that less gets done at 110 baud than at 300 or 1200 baud. The alphanumeric CRT display, such as the Datapoint, ranks a little higher on my scale of usability with students - it at least can type as fast as ordinary mortals can read - but it lacks hardcopy (often still valuable with students in spite of arguments about future paperless worlds) and has only teletype capabilities.

The two terminal types which are particularly competitive for student use today are the faster hardcopy units and the graphic units. Our experience with a Texas Instrument 720 has been very satisfactory; perhaps my only complaint there is the "gray on gray" writing. But graphics offers an entire new dimension to computer usage for ordinary mortals. Until recently it has been possible only for the wealthy, but terminals like the Tektronix 4010 and 4013 make graphic output practical for all student users. In *many* student situations the desired output *is* graphical, not numerical, and so graphic terminals open entire new realms of possibilities. Our experience with both TI and Tek terminals indicate that they are well-engineered, with almost no downtime.

Current terminals can certainly be improved.[3] One unfortunate assumption is that some fixed collection of characters, usually ASCII, will satisfy everyone. *Everyone* has his own favorite symbols which are "natural" to the area being studied, and computers should allow the symbols appropriate to that area. The Plato terminal does a good job with half the task, displaying any character shape under program control, but it still does not display the "new" characters on the keyboard. Another needed improvement is less expensive graphic hardcopy devices. And, given the proliferation of terminal types, terminals should be able to identify themselves on request from the computer; our mortals should not be forced to tell the computer what terminal is being used.

TIMESHARING SYSTEMS

A good timesharing system for the professional is not necessarily a good system for the novice. Unfortunately, computers are often picked by the professionals.

What does the ordinary mortal want in a timesharing system? First, easy accessibility. Typically a small program must be entered, and then interpreted, or compiled and run. It sounds simple, but details differ widely from system to system. What role, for example, do ASCII control characters play? Are they *essential,* or are they needed only for the advanced user? Are the error messages understandable to the beginner? How readable is the system and language documentation for the beginner?

Although the choices are critical, there appears to be almost no research in choosing a timesharing system for general student use. So views, including my own, are likely to be partially subjective. Irvine has a somewhat unusual situation; both a Xerox Sigma 7 and a DEC PDP-10 system are available, affording opportunity to compare the two systems with a variety of users. A student group also made, as an advanced computer science project, a detailed comparison between the two machines.[4]

My feeling, and I believe an accurate representation of the student study, is that the Sigma 7 Universal Timesharing System is definitely superior for the ordinary mortal. The PDP-10 comes into its own for advanced users who need its powerful forms of LISP and other esoteric languages; at Irvine its main usage is in upper division and graduate computer science courses.

Some aspects of timesharing systems are financially related. The novice is better off with more languages available, because he can call existing language background. But a general purpose timesharing system costs more than a single language system, so the choice is often one of funds available. Most one-language systems are BASIC systems, and I don't regard this as desirable.

EDITORS

The beginner quickly learns that most programs do not work when first written, and so needs the services of an editor. Like timesharing systems, editors can be optimized to particular classes of use. Again, few empirical studies have been made involving ordinary mortals. It would be interesting to determine how long *secretaries* take to become familiar with various editors. Fail-safe features are important - an editor should not allow a new user to inadvertantly wipe out large program sections.

RESPONSIVE LANGUAGES

The problem of languages and students is complex; students use the computer in a variety of ways, and the language needs differ. We can roughly distinguish between the situations where the student *writes* programs in some standard programming language, using the computer as an intellectual tool to solve problems, and situations where the student interacts with programs developed to assist in learning. It is fashionable to present this as an either/or choice, but both modes can be useful in learning.

The choice of language for student programing is clearly subject-matter dependent. Thus a word processing area might find SNOBOL the language of choice. For the second grader, turtle geometry has some interesting possibilities. In considering science students, I believe that APL, and possibly PL/I, are clear choices over FORTRAN and BASIC.[5] All are easy to learn for beginners, particularly if a reasonable subset is chosen, and reasonable learning methods are employed.[6] APL allows users to think naturally of calculations as involving *collections* of numbers, rather than individual numbers, and it provides ample room for intellectual growth as programming needs increase. APL also allows natural and powerful extensions to graphics.[7]

The student need not be concerned about the language facilities used to write dialogs for student-computer interaction.[8] But the *degree* of such interaction is important; the computer must be prepared to treat a very wide range of student responses to questions, both correct and incorrect, if the program is an effective teaching tool. The computer scientist often believes that the problems of such effective response analysis requires natural language processing. But our experience in the Physics Computer Development Project suggests that the pedagogical judgment of what a student is likely to say at a given point is the critical factor. Furthermore, use with sizable numbers of students and saving of unanalyzed student responses is also an important part of improving responsiveness. One tendency that must be resisted in preparing dialogs is that of easing the programming task at the expense of restricting the student. Thus if a formula is to be typed at a given point, the *program* should try to recognize many different variants, rather than provide the student with very detailed information about how he should enter the desired expression.

FINALE

Finally, I think it well to confess that we are all amateurs at this "ordinary mortal" problem. We have much to learn.

REFERENCES

1. R. Blum and A. M. Bork, *Computers in the physics curriculum,* Amer. J. Physics **38**, 959-970, 1970.
   A. M. Bork, *The computer in physics instruction,* New Trends in Physics Teaching, Vol II, Paris, Unesco, 1972.
2. A. Bork and R. Ballard, *The physics computer development project,* J. College Science Teaching (in press).
3. A. Bork, *Terminals for education,* EDUCOM, Bulletin 7, 1972.
4. L. Milburn, P. Stone, K. Welch, C. Brydum, J. Collins, S. Owen, D. Walker, W. Bambeck, K. Henriksen, W. Miller, N. Murvin, D. Nagel, E. Stack, *Sigma 7 vs PDP-10,* Student report, UC, Irvine.
5. A. Bork, *Science teaching and computer languages,* Unpublished.
6. A. Bork, *Introduction to computer programming languages,* J. College Science Teaching, 1971.
7. A. Bork, *Graphic APL for the sigma,* Proceedings Xerox User Group, Vol I, 1972.
8. A. Bork, *Teaching conversations with the Xerox Sigma 7,* Unpublished.

HEALTH CARE DELIVERY- THE IMPACT OF RECENT TECHNOLOGICAL DEVELOPMENTS

by BRUCE D. WAXMAN

National Center for Health Services Research and Development
Rockville, MD

If one reviews the more heralded applications of technology to medicine, one quickly realizes that the emergence of an extremely ingenious technological device is not tantamount to its successful application. This seems especially true of the myriad of innovative devices introduced to our health care delivery system. Despite demonstrations and evaluations which convincingly show a potential for increasing productivity, at improved or maintained levels of quality, too many of these innovations never realize their hoped-for potential.

Some technologists would have us believe that this is a fact of life resulting from the creation of a health care delivery system that is over-fragmented and geared to serving populations of a size well below the threshold where technology becomes profitable. A major advantage of technology, *in creating economies of scale,* is lost when custom design becomes preferable to mass production, when machine standardization and compatibility become secondary to individualized system needs. It is not surprising that many of our leading technology advocates are now admitting that despite whatever advantages technology has to offer, within the current structure of our delivery system, machines are more likely to increase, rather than decrease, the cost of medical care.

The logic for large investments of Research and Development dollars for the primary purpose of increasing medical care productivity becomes questionable unless there is a comparable demand to increase the total amount of consumer services provided. Some of the more successful technological developments are based on health delivery organizational systems which were restructured to take advantage of relatively large population bases. Others, more limited in the scope of their application, will be discussed in terms of their special adaptiveness to communication, transportation, and medical information technologies.

Many of these successful examples continue to be supported as major program activities of the National Center's Health Care Technology Division. As is so often the case, a high technology research and develoment base established to address health delivery problems in terms of cost, access, and quality of care, tends to spawn its own set of technological problems, that are in turn best resolved by the established cadre of scientific expertise, facilities, and delivery systems. The National Center is giving priority to merging these technological resources with the technological demands created by growing emphasis on national programs directed at better utilization of physician manpower, physician extender concepts, peer review, health maintenance organizations, and Federal, State and Local Data Systems.

Hospitals in this country that have automated their chemistry laboratories have demonstrated their ability to hold the line in terms of laboratory costs per unit of output. Despite a very rapid 15% yearly rate of growth in their workload, compounded by shortages of space and skilled personnel, unit costs have been reduced by the introduction of automation technology. Studies show that the average cost of a clinical chemistry test, using manual procedures, approximates fifty cents at a daily volume of 250 specimens. The introduction of automated procedures cuts this unit cost in half, although the cost to the patient may actually increase since the system now performs upward of eight tests on the same specimen, not all of which were ordered. At daily volumes approximating 1500 specimens per day, automation begins to result in significantly lower unit costs; and above 2500 specimens, costs drop below 10 cents per test. Although the number of hospitals in this country of sufficient bed size to justify these processing volumes is small, there are good indications that the recently expressed interest of smaller, geographically proximal hospitals to merge their laboratory facilities will produce the desired economies. The motivation for merger might well come from the analysis of data such as that being generated by Dr. Seligson at Yale, correlating decreased turnaround time for laboratory repprts with reductions in the average length of hospitalization. I'll report on additional developments concerning automated laboratory services later in my talk, but first I would discuss the somewhat analogous technological evolution that occurred in the area of automation of physiological signs.

For the past three years, the National Center has supported a project, based in St. Luke's Hospital, Denver, Colorado, to investigate the economies associated with the systematic community-wide introduction of a computer-assisted electrocardiogram analysis system. The results of this experience are very aptly described in a 130-page final report, copies of which are now available from the National Center. I found the report one of the better analyses documenting the impact of introducing a technologically advanced concept in a previously existent, highly competent component of the health care system. As in the case of the automated chemistry laboratories, the encouraging economic projections were realizable only through some restructuring of the traditional health care delivery organization. In 1969, Denver area hospitals manually processed over 120,000 electrocardiograms, and projected a 12 percent annual rate of increase for this diagnostic procedure. Significant cost reductions were demonstrated through the centralization and automation of their electrocardiographic diagnostic services; however an inverse relationship between unit cost and volume was observed. In Denver the break-even point is 300 EKGs per day. Further increases in their daily processing volume will offer an economy of scale that cannot be duplicated by conventional EKG systems. Cardiologist manpower savings up to 75% were reported by placing this automated system in series with a rotating panel of cardiologists representing all the participating hospitals. The cardiologists reviewed the centrally-filed EKGs independently of the hospital origin of the patient. The implications of these statistics are clear. Given a sufficient volume of EKGs, recent technological developments have made it possible not only to reduce unit costs, but to effect an appreciable savings in medical manpower.

I will now discuss an aspect of medicine that traditionally has absorbed a lion's share of the technological R and D investment, although on a national scale it represents a relatively small spectrum of the total health care delivery system.

Data from the surgical intensive care unit at the University of Alabama indicate that introduction of automated monitoring and blood infusion programs can reduce post-cardiac surgical mortality to as low as 7%. This favorable figure has been maintained in the face of an average post-surgical stay in the intensive care unit of one day, and a conservative patient-to-nurse ratio of 3 to 1. Routinization and mechanization of these relatively complex medical manpower tasks have effected (1) a reduction in length of stay in the intensive care unit, and (2) decreased medical manpower requirements, both essential prerequisites to economic savings. Although representing only a small fraction of the overall health care need, intensive care techniques are especially adaptive to technical innovations, and even without major R & D efforts, continued refinement of existing technologies can be expected to effect further improvements.

In the almost explosive growth of hospital information system technology, the shared, modular systems utilizing small to medium-sized computers appear to offer the most economically promising solution. Information handling as opposed to patient management is stressed, the objectives being to achieve automation in a modular or incremental fashion through terminals connected via telephone to larger timeshared systems servicing many hospitals. For example, the census and bed control module at the Framingham Union Hospital (MA), by its requirement for daily physician updating of estimated length of stay, has achieved a 14% utilization increase for the 150 medical/surgical beds in this 288-bed hospital. Similar technological support modules applied to radiology and pharmacy information handling can be expected to achieve comparable cost reduction through rationalization of information flow within these departments.

At the Massachusetts General Hospital Outpatient Clinic, an automated medical history is saving physician time. New patients are given a paper history and the results are keyed into a computer terminal which quickly generates an annotated printout of the history information. Here in Washington, patients of a five-man group practice in internal medicine produce a similar history output by entering their responses directly into a computer terminal. In both instances, the physician further annotates the historical report during the physical examination, but at a considerable savings of his time. The coupling of automated history taking and triage procedures with paramedicals will result in even further savings in medical manpower.

Physician consultative aids are available to support the practicing physician with knowledge of immediate use to him in the diagnosis and treatment of his patient. The major benefit of these programs is the transference of specialized and dynamic knowledge from the medical center, with its ready stores of literature and subspecialty consultation, to outlying medical care locations. Their use for library reference, poison control, and audio response to pertinent subject review is now state of the art, and protocols for fluid and electrolyte management, hypercalcemia and coagulation disorders, and aid with complex differential diagnosis are being widely demonstrated.

Flow sheets of clinical logic have been constructed which enable the paramedic, using a computer terminal, to collect (in a logical fashion) patient historical, physical, and laboratory data for a large number of common complaints such as earache, sore throat, cough, backache, fatigue and headache. By emulating the decision moves used by physicians in the handling of these complaints, the protocols often enable the paramedic to handle a given patient from presentation to exit. Points at which a physician's intervention is required are automatically flagged. In addition to supporting the paramedic, computer protocols permit the physician to periodically review both the performance level of the paramedic and the clinical logic of the computer. F. J. Moore reports in his "Information Technologies and Health Care" that the number of allied health workers varies inversely with the physician/population ratio. Further deployment of automated support and audit systems will expand the population base that a physician can adequately serve, and greatly increase patient throughput in areas with few physicians.

In these examples of the technological contribution to our educational process, it is evident that learning efficiency for a given subject is directly proportional to the immediate relevance of that subject for the learner. For the physician, protocols such as Bleich's fluid and electrolyte, hypercalcemia, and coagulation disorder programs, Worley and Ringe's differential diagnosis programs, and Barnett's cardiopulmonary resuscitation programs, have their greatest impact when used as consultative aids for each specific patient. Paramedic protocols offering the greatest impact appear to be the operating support protocols such as the Lincoln Laboratories' Chronic Disease Management program and Dartmouth's Acute Disease Management and Audit programs. Patient education protocols showing the greatest promise are epitomized by the highly interactive programs directed at weight reduction, diabetes management, pediatric immunization, or situational depression.

It would not be difficult to extend this description of technological applications to health care for several more pages. There are certainly many other applications, equally innovative, that have demonstrated technology's impact on the cost, accessibility and quality of care available in this country. Within the National Center we have attempted to place these innovations in real-world settings offering a maximum opportunity for further deployment and tranferability of the technology to other health care systems. During this deployment, or transference, stage it is becoming increasingly more evident that flexibility to adaptation is essential if widespread deployment is to be achieved. This has resulted in the creation of a completely new set of technologically oriented projects. For example: the centralization of laboratory services creates logistical problems perhaps best solvable by the twin technologies of transportation and communication, but equally responsive to miniaturization technology. On the one hand we have the specimen transfer and information handling problems of centralized laboratories that process several million samples per year. On the other hand is the concept of sophisticated miniature analyzers, mass-produced at a unit cost low enough to permit analysis at the primary site of patient care, the physician's office and outpatient clinic. This latter idea is attractive to physicians who believe they can perform better if they have the patient and his data in one place at the same time. Obviously any further decentralization along these lines obscures advantages I will shortly be citing about the centralization of the medical record.

Systems for computer analysis of the electrocardiogram are also highly susceptible to modifications generated by communication and miniaturization technology. Primarily developed for hospital in-patient applications, miniaturization of acquisition systems and even mobile self-contained computerized EKG systems are now within the realm of technological practicality. The ubiquitous telephone, modified for automatic card dialers, auxiliary Touch-Tone keyboards, and computer-generated voice response, has the capability of revitalizing the physician's office as the primary place for patient care at a level of quality deemed impossible only a few short years ago. The problem-oriented medical record, adapted to computer techniques, and integrated with essential administrative data, offers significant advantages for the patient, the primary care physician, and the health care system manager. Grossman has suggested that four distinct levels of care are potential beneficiaries of such an integrated record.

> "At the level of daily patient care, medical records can be searched and sorted by computer to group patients for special purposes. Patients who should receive influenza vaccine each winter, people with chronic diseases, endocrine abnormalities and those over the age of 65 ... With such a record system, the computer can search the patient files and produce a list of patients meeting these criteria.
>
> At the level of the individual physician's practice, the system serves as a source of feedback. Computer-generated reports can list in decreasing frequency the diagnoses the physician has made, the number and type of tests he has ordered, and the number and the type of medications he has prescribed. Reports can be produced by specialty so that an individual can compare his practice to that of the group as a whole, contributing to a sense of group enterprise rather than individuality among practitioners.
>
> At the levels of the health care institutions, accessibility of aggregate medical information brings an entirely new type of data to the administrative planning process. Because of the availability of exact diagnostic distribution, placement of personnel can be based on actual case loads and trends rather than predicted needs. The data also contribute to the control of costs, and medical and lay administrators can know much more precisely the individual utilization of available resources.
>
> Finally, at the level of health care evaluation, the availability of structured medical data provides for an opportunity to build definitive criteria for quality of care. For example, in assessing the chronological course of diabetes mellitus, assessing the levels of blood sugar over time, or checking if the blood pressure of hypertensive patients falls below a certain required therapeutic level. The online availability of these status records precludes the need to pull a patient's paper medical record for use with scheduled patients, unscheduled walk-in patients, and telephone inquiries."

A model for this integrated record system is currently operating at the Harvard Community Health Plan, where an online status report and laboratory system permit cathode-ray tube displays for use by the member physicians in their offices, by the plan's administrative staff, and by paramedics for triage and educational purposes. The statistical reports generated closely approximate the four levels suggested by Grossman; *Medical Aggregates* (listing of diagnoses, tests ordered, and medication in order of frequency for each staff member); *Utilization* (the further division of aggregate table to reflect specialty visited, and frequency of visits, cross-tabulated with age, sex, and demographic data); *Membership Data* (alphanumeric lists of membership by specific characteristics); and *Special Studies* (listings of women taking sequential birth control pills who should discontinue their use, profiles of the Welfare-supported population within the Plan, mailing lists of members meeting criteria for participation in emergency preventive health programs). These ongoing services also parallel the data components described in the general guidelines of Federal-State-Local-Health-Systems, requiring that "Systems be developed in association with efforts made to provide the information needed for patient care, quality assessment, fiscal responsibility, certification of need, and health planning". The Harvard model appears equally adaptable to other major health care delivery systems such as EMCRO, EHSDS, and Health Maintenance Organizations.

Any description of a model health care delivery system, complete with hundreds of remote terminals and cathode-ray tubes, equally accessible by physicians, paramedics, administrators, and patients, immediately brings to the technologist's mind the concept of added system utilization through the introduction of a two-way visual communication system. For this final portion of my discussion on recent technological developments I would like to comment on the impact we might expect from the potentially explosive role that broadband communication systems might contribute to a health delivery system. First let me mention some of the questions health care technologists are asking. Can physicians remotely supervise patient care for a relatively large population of bedridden individuals by communicating directly with paramedical personnel through interactive television systems? (Adaptation of television sets for this use could radically reduce the cost of manpower dedicated to in-hospital and other confined patients.) Are physicians, linked by Picturephones or other broadband modalities to the hospitals with which they affiliate, able to review X-rays, EKGs, and laboratory results, and discuss these with the hospital's staff without leaving their offices? Can patients who are under examination in one physical location benefit from online consultation from remotely located specialists? Can our considerable reserve of nurses turned housewives be brought back into the health care system by the installation of an interactive visual screen in their homes, linking this vitally needed talent to physician or patient? Is there a sense of "personal presence" when patients and medical care professionals are in face-to-face communication via television, and will the availability of such service ensure more equitable access to health care without further disruption of an already overburdened transportation system? (For example, in Chicago, the cost of transporting an elderly patient from a nursing home to an out-patient clinic and back is eight dollars.) These and other questions are presently under investigation in a series of projects supported by the National Center which address the extent to which visual communication can be substituted for travel.

In all of these applications of technology to the delivery of health care there is an implied interdependence between those technologies whose primary impact is in the area of patient services and those technologies with primary impact on the educational process. It is the entrainment of the health care professional into using computer technology in his own educational process that most facilitates the use of similar technology in his medical practice patterns. All the examples I have chosen have promoted further trial and use of technical practices and devices, and are exercising a spreading train of influence in the emergence of other technological innovations. The impact of this technology, measured in terms of the cost, quality, and accessibility of health care, will be better measured when its deployment reaches a more representative portion of the patient population. We may at that time find that the greater impact has already occurred, in the changing health delivery concepts of administrators, physicians, paramedics, and patients familiar with the ways of automated, push-button, interactive, visual, two-way, audio response technology.

SPECIALIZED LANGUAGES:

AN APPLICATIONS METHODOLOGY

by RICHARD H. BIGELOW, NORTON R. GREENFIELD, PETER SZOLOVITS, and FREDERICK B. THOMPSON

California Institute of Technology
Pasadena, CA

One objective of the information processing community is to aid the problem-solving activities of its clients. In this paper we will discuss a methodology for serving the needs of the "user", that is, the end-user: the manager running an organization, the accountant understanding the financial condition of a company, the anthropologist studying a culture, the engineer designing some equipment, or the meteorologist predicting the weather. Each of these users has his own particular, idiosyncratic problems. The computer should be an effective tool for him in dealing with these problems. Our methodology is designed to provide each of these users with an appropriate interface to the computer, with a language which is natural to his view of reality.

In this paper we examine the nature of today's ubiquitous applications packages, discuss our notion of applications languages and present some of our experience with the REL system, which has been designed to incorporate our views on specialized user languages.

This research has been supported by:
Office of Naval Research contract #N00014-67-A-0094-0024
National Science Foundation grant #GH-31573
Rome Air Development Center contract #F30602-72-C-0249

APPLICATIONS PACKAGES

The bare hardware of a computer, from the point of view of the user, is impotent. An operating system augmented by a few language processors, e.g. FORTRAN and COBOL, is hardly more useful. Indeed, when constructing complex applications for an end user, the standard programming technique is to first build a set of data structures and utility routines which then become the user's environment and make the computer habitable. We will call any consistent set of such structures and routines an *applications package.*

The need for many such applications packages is clearly demonstrated by their existence and wide usage. Examples are standard programs for payroll and inventory control in business, the SPSS[1] package for statistical analysis, subroutine libraries and languages such as NAPSS[2] for numerical analysis, and APT[3] for machine tool control. Hundreds of other illustrations may easily be found.[4,5]

All these systems have a common property: they provide operators which perform meaningful unit operations needed by their users. Their primitives draw up payrolls, compute correlations and solve differential equations. We wish to examine how the user invokes these primitive operations to fulfill his requirements.

In the most unsophisticated system, the user invokes a unitary operator: e.g. "produce the payroll". This is a complete program, perhaps operating on large bodies of data, with well-understood, structurally constant results. This type of applications package supports only a single aspect of its users' reality. Being optimized around a single task, it is not easily modifiable to meet even simple contingencies and it often quickly becomes inadequate. Problems change in structure as well as in data. Because the user has no concept of the computer representation of sub-parts of the whole problem, he is left with the use of a partly obsolete operator, or he becomes dependent on his programmers to make even the most minor structural changes in his problem solutions.

Concerns of typical computer users consist not of single, all-encompassing operations, but of a number of lower level tasks which in combination allow the solution of a range of related problems. For example, the accountant does not care merely to be able to produce his quarterly financial report. He has to be able to investigate data on various aspects of the company's fiscal status to understand his problems. His products are not just periodic reports, but also the tax and cash flow calculations, projections, special briefings, etc. Similarly, a physicist manipulating his experimental data looks not for a single answer, but for a multitude of indications and partial results which may help him understand the processes he is studying.

A computer system which supports such investigations must embody a number of different primitive operators, to correspond to the variously complex conceptual units of the user. It must also allow the hierarchical combination of these primitives to build up the operations which match higher-level user concepts.

The common production of standardized subroutine libraries in many fields attests to the widespread acceptance of this view. Such libraries, along with the standard algebraic computer languages, allow the construction of hierarchically-composed calls on the primitives, offering flexibility and power. What, then, are the inadequacies of these sophisticated applications packages to the user?

On the one hand, a computer system for a particular user must embody a large set of the conceptual primitives of his problem area in order to be useful to him. In addition, however, that system must also exclude the incursions of as many as possible of those computer concepts unrelated to the problem area. All of today's generally available programming languages have a strong bias in their syntactic and semantic capabilities to fit the needs of their designers, namely computer scientists. Their natural primitives include the control of storage, input and output, the declaration of procedures, data types, etc. Every one of these concepts is foreign to the problem area in which the nonprogrammer user is working. Thus, although the subroutines in a library may well represent valid primitives to our user, the irrelevant concepts of program control, procedure calling, and data management intrude upon and disrupt his problem solving.[6]

Current users struggle with this disruption in different ways: the accountant must work through a programmer, removing himself from direct contact with his data; the physicist often *becomes* a programmer, sacrificing his productivity as a scientist to develop competence in a field of only incidental interest to his work.

The information science community can provide better technical solutions and more viable tools.

APPLICATIONS LANGUAGES

To be most effectively utilizable, the computer must metamorphose to be each user's own conceptual machine. It must embody exactly those primitive notions which the user finds fundamental; it must support that structuring of complex problems which the user finds natural. And because the user must be able to communicate easily with his machine, it must provide for communication in a language which embodies the user's conceptual primitives and the means of composing them clearly and concisely.

It is not generality that the language must provide. Indeed it is exactly in its ability to reflect the biases, limits, and idiosyncratic representation of the user's reality that a specialized language finds its greatest strengths. The user brings with him a host of presuppositions, the knowledge of his field, of which he is only peripherally aware, but whose logic underlies all of his problem-solving activities. General languages *know* nearly nothing about the problem domain. All checks, all limits, all structures must be explicitly expressed by the user. In any high-level application, the amount of knowledge which the user has about his data is enormous. To enter it as explicit instructions to the computer and to probe his data in a system which recognizes none of his tacit knowledge is unconscionably tedious.

The implicit inclusion of the tacit knowledge of a specialized problem domain is the advantage which gives the applications language both expressive conciseness and computational efficiency in the problem-solving tasks of the particular end user.[7] With such a language the user can concentrate on his problem instead of the programming details. There is no intrusion of foreign concepts from the implementation - the user manipulates structures and operators that are familiar and relevant. The power of the language opens new options and capabilities in his use of the computer, and the naturalness of the language allows him to exploit those capabilities himself, bringing his own implicit knowledge and intuition to bear without having to work through a programmer.

At the same time, the embodiment of the user's presuppositions implicitly in the prior programming of the primitive operators results in increased computational efficiency. It is often erroneously assumed that higher-level, user-oriented languages entail increased computing times as well as excessive implementation costs. Quite the contrary. The existence of specialized knowledge of the field of application allows more global optimization of the basic primitives. And once programmed, these primitives can be composed in the solution of wide-ranging problems, being reused a multitude of times without involving any new programming tasks. One can appreciate the extent of such savings by considering the compaction of records and optimization of access to peripheral storage which the programming of specialized primitives can embody, savings in ultimate computer time which can amount to orders of magnitude for large data bases.[8]

The fear has been expressed that the widespread development of such languages would lead to a large number of small user communities, each with its own highly specialized language, each unable to communicate data and methods of solution to the others. Consequently, the argument goes, we should concentrate upon standardizing our languages rather than specializing them, to allow the easy exchange of data, algorithms and personnel.

We find two related answers to this line of argument. First, we do not believe that our current experience with sharing data or programs justifies the requirement of adherence to rigid standards on the part of all computer users. Specialized languages already tend to arise in response to natural divisions which exist among groups of users. Hence, between groups isolated by specialized languages, it is already unlikely that they would profit from sharing of common technique and common data. Second, the increased capabilities provided a group by a specialized language may well justify accepting the cost of relative isolation. It is the user community's responsibility to regulate language development to achieve an economic balance between specialized capability and communication. Between groups where communication and sharing of data is desirable, their various specialized languages can explicitly facilitate precisely such common access and cross-talk.

Currently, the economic factors underlying the decision of whether or not to create a specialized language are dominated by implementation cost. Technical advances of the sort we will describe can reduce this cost sufficiently to allow that decision to be made on the grounds discussed above. We now examine the task of implementing specialized languages.

METALANGUAGES

From the implementor's viewpoint, a computer language consists of a set of procedures containing the semantic primitives of the language, the set of data structures to which these are to be applied, and a syntax which allows the user to compose his operations and to apply them to his data. The task of the language implementor is to analyze the natural requirements of the user in these three areas and to design and code the procedures, data structures and syntactic processor to realize the language.

We can examine the language writer's problem just as we looked earlier at the end-user's problem. We note that current programming languages do not have operators and data structures in their semantics which specifically support language implementation. Because their facilities are much more primitive and detailed, the construction of applications languages is difficult and costly. The language implementor needs a specialized language, just as the user does. The primitive concepts of this language must be parsing, storage management, permanent and temporary data base management, semantic compositions, etc. Again we emphasize that this implementor's language is not a generalized language. Not all implementors will want the same parser nor the same data base management scheme. However, for particular classes of language implementors, those implementing similarly-structured user or *object* languages, a particular implementors' or *meta-* language is useful.

A metalanguage structures and supports the task of the applications language implementor in the same way that the applications language structures and supports the task of the user. It allows the implementor to concentrate on the problems of designing his language and supports its implementation. For example, provision within the metalanguage of an efficient parsing algorithm coupled with a simple means of expressing syntax rules will allow the programmer to utilize a natural syntax in his language. He is no longer forced to a simple syntax by the high cost of implementing anew a complex parser.. The metalanguage can embody much of the tacit knowledge of the language implementor about the internal structure of the language. For instance, a rigid coupling between rules of grammar of the object language and the invocation of the associated semantic primitive routines allows the metalanguage to know the calling and return structures of these semantic routines, and to use this knowledge to allow a more concise description of the routines and to perform error checking or optimization on the object language. The metalanguage also directs the attention of the language implementor to the central issues of his task: the construction of the operators and data structures that are significant to the user and a natural syntax for combining them.

REL - THE RAPIDLY EXTENSIBLE LANGUAGE SYSTEM

The REL System has been developed to give concrete realization to the ideas presented in this paper and allow us to get actual experience with the use of such a system. We will not further describe REL here, but will only enlarge upon those aspects which relate to ideas discussed in this paper. For a more complete description of REL, see references [9,10,11].

The REL System provides a metalanguage for the implementation of sentence-driven, syntax-directed, interpretive and extensible applications languages[12]. Within the REL environment, a language is represented by a set of general rewrite grammar rules, their corresponding processing routines, and the data structures of the associated data items. The grammar rules structure the operation of the language, define the valid syntactic constructs which the user may employ, cause invocation of the syntactic and semantic processing routines, and define which data types may be related in the language. As an example, consider the following grammar rule:

<class;relation_image> => <relation> 'OF' <class>

This may be a rule of grammar of a language which expresses aspects of a relation calculus. The rule, written exactly as shown here, states to the REL system that:

- the syntactic construct "name of a relation" followed by the word "of" followed by "name of a class" is valid.
- such a construct represents another data item of the type "class",
- this new item may be computed by applying the program named "relation_image" to the two old data items.

Notice how this metalanguage forces the implementor's attention to exactly the problems which should concern him: the primitive entities of his language, e.g. "class" and "relation"; the primitive operations of the language, e.g. "relation_image"; and the syntax by which these can be combined, e.g. "regions of salesmen", "vendors of components".

We have emphasized that the user's language should fit his needs naturally. That means that he must often be able to define new operations on the basis of his previously existing operations to express new tasks and methods of solution. REL provides the applications programmer with a powerful tool to implement this ability for his language. Using whatever external syntax he finds natural for his users, the programmer can invoke an REL system utility which will add to the user language's grammar new rules which express the desired definitions of the user.

REL APPLICATIONS LANGUAGES

We have used REL to implement a variety of languages and have found it to be very supportive of them. Indeed, even if we had wanted to develop only one fairly complex language, we would have found it desirable to separate the REL system and general language processor facilities from the syntax and semantics of the particular language. Doing so has given us a framework in which to design our languages that has been at least as important as the support we have gotten to actually write the code.

The languages that have been implemented under REL to date include REL English,[13,14,15] the Animated Film Language (AFL),[16] a language for solving ordinary differential equations, [17] and a discrete simulation language.[18] We present a few examples from the first.

REL English is a technical English question-answering language for the analysis of complex sets of highly interrelated data. Its primitive operations are based on the data and semantics of a relational algebra. Thus the language was designed with a view to serving users with messy, large-scale data-related models. REL English's current users include a cultural anthropologist, a research hospital, and elements of a military staff.

The syntax of REL English is a complex, quite natural, deep case grammar which provides our users with powerful but concise statement and query capabilities. The primitive data entities of the language are individuals, classes, and binary relations. REL English has all the common notions of sentence structure, time, function words like "all", "what", and "the". It does not include any particular vocabulary but provides the ability for the user to introduce new words which denote individuals, classes and relations from his own problem domain. Further, it has the capability for defining new verbs in terms of relations and the verbs of being, and it provides the ability to extend itself by new syntactic forms which represent composed operations of the language, as specified by the user.

Note that this much REL English is common to a wide variety of users. Relating to the earlier discussion of the cost of implementing specialized languages, we remark that to this point the cost of adding yet another English-based language to REL is merely the effort of deciding that the relational data structure and an English statement and query capability are natural to the user's problem area. To specialize to the requirements of a particular user, the extension facilities of REL English are used to introduce the relevant user concepts to the language.

Our example will be the familiar one of the personnel data base. The initial preparation of the language consists of acquiring a copy of REL English and adding appropriate terms:

```
employee := class
department := relation
immediate supervisor := relation
salary := number relation
```

We can then include all of the basic data on each person, usually taking it from some fixed-format file. At this point, the personnel manager can ask the usual questions:

```
What is Sue Jones' salary?
When was John Smith Bob Jones' immediate supervisor?
How many departments have employees whose salary is over 20000?
```

The manager will soon extend this simple language with meaningful and useful terms:

```
def: senior employee: employee whose salary is at least 18000
def: subordinate: converse of immediate supervisor

Are all managers senior employees?
What proportion of senior employees are female?
Which managers have more than five subordinates?
```

The user can, of course, produce reports. The statement:

```
What is the ratio of male employees to female employees in each department?
```

produces a columnar listing of the departments and their male female ratios. Other involved conceptualizations can be expressed by verbs:

```
earn := verb (<object> is the salary of <agent>)

Does some employee earn more than his immediate supervisor?
```

The capabilities represented here allow the user to efficiently explore the interrelationships which are meaningful to his task.

The above is a small illustration of the type of applications language which we have implemented in the REL system. Each of the other languages mentioned have quite different syntax and semantics. Although our experience to date is limited, these applications have been found to be directly and effectively usable by their intended users and inexpensive to implement.

CONCLUSION

The continued development of more sophisticated software and better, less expensive hardware should lead to a great increase in the number of computer users. As a tool for organizing and managing large, complex human problems, specialized computer languages promise to increase our effectiveness in handling a complicated world. Indeed, only by the support of specially tailored "natural" languages will the large group of new computer users have the ability to effectively deal with this growing resource. Whenever possible, the burden of making man-machine communication tractable should fall on the machine, where the burden is manageable through the use of specially designed metalanguages and applications languages.

Our experience with REL gives us confidence that the notion of a metalanguage for the implementation of whole classes of applications languages is legitimate and valuable. We intend to continue exploring the wide range of end-user oriented languages which find a natural home within our system, and we envision the future construction of other metalanguages (or programming systems) for different classes of applications languages.

We would like to make a few final comments about the impact of the above ideas on the computer professions. We expect a redefinition of the relation between systems programmer, applications programmer, and user. The user has problems to solve, which he can state in some language specialized to his universe of discourse. The task of the applications programmer, in our view, is to provide the user not with solutions to individual problems, but with computer languages and capabilities to allow the user to pursue the solutions of his problems in terms of concepts which are natural to his problem domain. The task of the systems programmer is to build efficient language processing systems and their associated metalanguages so that the applications programmer can concentrate on the structuring of the data, preparation of the processing algorithms, and specification of the syntax natural to his user. The current work in computer systems such as REL will facilitate the task of the applications programmer.

Finally, the power (and thus the responsibility) of the applications language implementor is great. As our everyday language affects our thoughts, our computer languages guide and limit our work. An appropriate and flexible applications language can greatly enhance the work of a user; a poor and rigid one can impoverish it. The future of our ability to effectively use one of our most powerful tools, indeed, of our ability to cope with an informationally overwhelming world, is at issue.

ACKNOWLEDGEMENT

The authors acknowledge the assistance of Ms. Sara Gomberg in the preparation of this paper.

REFERENCES

1. Nie, N.H., Bent, D.H., and Hull, C.H., *SPSS: Statistical Package for the Social Sciences.* McGraw-Hill, New York, 1970.
2. Symes, L.R. and Roman, R.V. Structure of a language for a numerical analysis problem solving system. In *Interactive Systems for Experimental Applied Mathematics,* Klerer, M. and Reinfelds, J. (Eds.), Academic Press, NY, 1968, 67-78.
3. *APT Part Programming.* McGraw-Hill, New York,1967.
4. Sammet, J.E. *Programming Languages: History and Fundamentals.* Prentice-Hall, Englewood Cliffs, NJ, 1969.
5. Sammet, J.E. Roster of programming languages, 1972. *Computers and Automation* 21, 6B (1972 Aug 30), 123-132.
6. Dmytryshak, C.A. The universal consulting language - alias the investment analysis language, *Proc. Fall Joint Comput. Conf. 1972,* **41**, Part I, 525-535.
7. Thompson, F.B. and Dostert, B.H. The future of specialized languages, *Proc. Spring Joint Comput. Conf. 1972,* **40**, 313-319.
8. Greenfeld, N.R. *Computer System Support for Data Analysis.* Doctoral Dissertation, REL Project Report No. 4, Calif. Inst. Tech., Pasadena, CA, 1972.
9. Thompson, F. B., Lockemann, P.C., Dostert, B.H. and Deverill, R.S. *REL: A Rapidly Extensible Language System.* Proc. 24th ACM Natl. Conf., 1969, 399-417.
10. Dostert, B.H. *REL - An Information System for a Dynamic Environment.* REL Project Report No. 3, Calif. Inst. Tech., Pasadena, CA 1971.
11. Thompson, F.B. and Dostert, B.H. *The REL System,* Fourth International Symposium on Computer and Information Sciences (COINS - 72), Miami, FL, 1972 Dec.
12. Szolovits, P. *The REL Language Writer's Language: A Metalanguage for Implementing Specialized Applications Languages.* Doctoral Dissertation, Calif. Inst. Tech., Pasadena, CA, forthcoming.
13. Dostert, B.H. and Thompson, F.B. *The Syntax of REL English.* REL Report No. 1, Calif. Inst. Tech., Pasadena, CA, 1971.
14. Dostert, B.H. and Thompson, F.B., "Syntactic Analysis in REL English: a computational case grammar", *Statistical Methods in Linguistics,* 8(1972), 5-38.
15. Dostert, B.H. and Thompson, F.B. Verb semantics in a relational data base system. *Proc. ONR Symp. on Text Processing and Scientific Research,* Pasadena, CA, 1972 Nov.
16. Bigelow, R.H. Greenfeld, N.R., Szolovits, P. and Thompson, F.B. *The REL Animated Film Language.* REL Project Report, forthcoming, Calif. Inst. Tech., Pasadena, CA.
17. Bigelow, R.H. *Computer Languages for Numerical Engineering Problems.* Doctoral Dissertation, REL Project Report No. 5, Calif. Inst. Tech., Pasadena, CA, 1973.
18. Nicolaides, P.L. *RELSIM - An On-Line Language for Discrete Simulation in Social Science Research.* Doctoral Dissertation, Calif. Inst. Tech., Pasadena, CA, forthcoming.

ENSURING INPUT DATA INTEGRITY IN A HIGH -VOLUME ENVIRONMENT

by ROBERT F. STOVER and S. KRISHNASWAMY

Honeywell Information Systems
Framingham, MA

SCOPE/INTRODUCTION

The cost of errors in entered data can be significant in a company's data entry operation. Currently available program-controlled data entry devices and systems afford the user an opportunity to detect the error, but they generally cost more, and the error detection may also impact throughput.

The objective of achieving selective data integrity at lowest overall cost implies an understanding of errors, their detection and correction.

This paper is based on multistation key-disk systems because of their generalized, extensive error detection and reformatting capabilities, subsets of which are found in other, single-station devices. The points should be applicable to all devices.

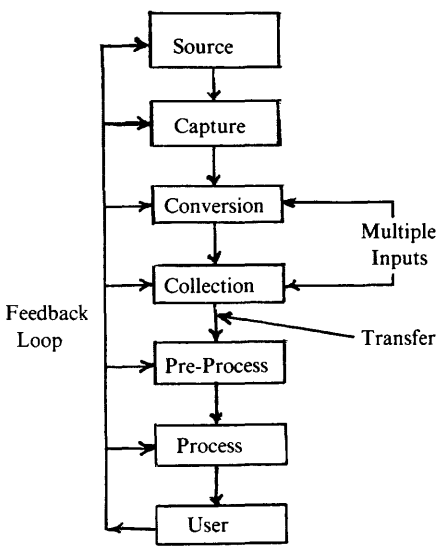


Figure 1. Schematic of the Data Entry Process

ELEMENTS OF THE DATA ENTRY PROCESS

Figure 1 shows the schematic of a generalized data entry process which really encompasses all the necessary steps from the source of data to the processor, together with the essential feedback from each step to a previous step as appropriate.

Descriptions of the Elements

- *Source:* The actual source of data, like storage bins or point of sale, etc. The source is assumed to be a "pure" source with no processing.
- *Capture:* The process of first entering selected data from the source into the system, for example, writing up the sales order.
- *Conversion:* The process of converting the data into a form in which it can be read and manipulated by electronic processing machines. This can be transcription (keying) or mechanical (OCR, OMR).
- *Collection:* The process of collecting the converted data from data various sources or "batching", or "pooling", if appropriate.
- *Pre-Process:* A possible intermediate reformatting and labelling step to get the data ready and organized for processing. Traditionally this has been done in the computer that does the processing. "Collection" may follow this step instead of preceding, or may be part of it.
- *Process:* The data processing function with an associated central Data Base.
- *Transfer:* The link between the boxes (Figure 1). The method of conveying the information from one box to another, generally involving some form of transportation like US Mail, trucking, data communication, etc. In its broadest general interpretation this function is present between any two boxes. It may imply cost and delay or may only be incidental movement

The data processing function is included here only because traditionally it has actually been required to perform some functions like collections and preprocessing that are really data entry functions. This paper will *not* cover processing errors or details of the implementation of the feedback loop.

SOURCES AND TYPES OF ERROR

Considering each element of the data entry system in turn, the following kinds of error are possible:

Capture

Vital data may be missed or not captured in sufficient detail. The wrong data may be picked up or the data could be entered in the wrong area of the form.

Conversion

Mechanical devices like OCR could read erroneously or not at all if the input data does not fit a prescribed form. Keying of data may introduce skipped field, keying (typographical) errors, misaligned data and duplicate records.

Collection and Preprocessing

Mechanical errors occur when data is moved around within or between memory and external storage. Imperfect software could misinterpret entered data or introduce errors. Error detection and correction algorithms and procedures could themselves introduce errors. Automatic duplication of data in a field from data previously entered is an example of this. If subsequently the first occurrence of the data is corrected, all subsequent occurrences could be in error without appropriate software correction. Similarly, if a field used in arithmetic operation (hash totals, extension) is changed subsequent to the operation, the field holding the result would be in error. Data loss could also be caused by external problems like mechanical failures (disk head crashes) or electrical (power) failures.

Transfers

Errors could also be caused during data transfer, as in communications. On a larger scale, large batches of entered data, collected or pooled and transferred to the data base of the processing element, must be properly labeled and identified to ensure that the processor is fed with the right data set.

Indirect Errors

The user's data entry procedures and changes to them are potential error sources. The 80-column card constrains the entry format to the processor requirement, not the format of input data. Effectively the operator is editing while keying. If input exceeds 80 columns, the operator is forced to look up and rekey identifying field on each "card". In some cases the information is reduced to codes on the form. All these reduce the operator's comprehension of keyed data, raising the probability of error. The user is also naturally liable to change and improve his procedure, and any potential errors during this period must be minimized by a good data entry system.

Illegal Data

There is always the problem in any data entry system of a person attempting to pass illegal data such as nonexistent charge accounts. The system must also provide the necessary check digit algorithms to prevent this.

Errors can therefore occur in any element of the data entry system itself, due to the complexity of the input forms or keying procedures. The errors may range from the total absence of essential data to the entry of illegal data and include invalid data, incorrectly entered data, or errors that are actually introduced by the data entry systems hardware or software itself.

VALUE OF ERROR

It is of prime importance to realize that all errors are not of equal importance. It simply is not economically sound to strive for error-free data just because that is the "right" thing to do. Instead, more care should be placed on designing a system in which error detection, throughout, and cost of undetected errors are balanced.

The nature of the data provides quick clues as to the nature of the potential problems. Textual data generally can tolerate errors because it is for reference purposes, i.e., human use, and errors can be detected since the symbols have easily recognized meaning. Numeric data must be precise and it is relatively easy to make errors. Coded data may be even worse since there are fewer applicable check methods.

As stated, the objective is to get a required level of data integrity at minimum cost. This implies balancing error detection procedures against productivity. One must consider where the points of error detection are, where the knowledge to correct the error lies, and how well the detection procedure can isolate the error.

DETECTION OF ERRORS

Once the user has determined the errors that affect the system crucially enough to be detected and corrected, a number of methods are available. The selection of methods depends on the nature of the errors and the points where they are to be checked.

The following discussion is along the same lines as that on the sources of error.

Capture

Data errors incurred during capture are caught during conversion time. To prevent omission of data, the data entry (conversion) equipment can be programmed such that the user may specify that data "must be present" in specific crucial fields. The system must also provide means of record insertion and deletion to correct for totally missed or duplicated records respectively.

Capture (and entry) of invalid data can be reduced to a degree, particularly for numeric data, if the attributes of that data are known. Thus, one can specify the range of values (range check) for the data to be entered, or the values that are legal (value set), or provide validation controls such as check digit algorithms.

Conversion

Keying errors can be caught by keying controls, validation controls, and finally by verification.

Keying controls can check if the data is to be in upper or lower case, whether the field can be skipped or duplicated from a prior occurrence, and the legal types of data in the field (like alphabetic, numeric, blanks or special symbols).

In addition, one could also specify minimum and maximum number of characters in the field, and if the field is numeric, specify that it is to be included in a hash total. A hash (or batch) total does not really check each field as it is entered, but gives an overall check for a defined collection of records.

Crucial data that needs to be error-free, but is checked either incompletely or not at all by validation during entry, can be checked by two forms of verification - visual and key. Visual verification consists of providing the operator with a display of a record on a screen to permit correction either during initial entry or during a subsequent verification and correction run.

A visual display by itself is not effective in detecting errors, as operators seldom look at displays while keying. If an error is detected by other means, such as validation controls, a display could be useful in correcting the error. A display is particularly useful to an operator familiar with the data being entered. In such cases, the operator would be able to spot the error and could possibly correct it generally if the error is in keying (wrong case, wrong key, etc.) Even here the use is limited, because if the error is in the validity of numeric or coded data the operator really cannot correct it any better than without the display. The value of displays is not clear in high-volume situations like tab room operation where the data is not meaningful to the operator.

Key verification is the process of rekeying and comparing the records created in the two passes, character for character. It is really no more than a check on the keying. The keypunch verifier forces the user to rekey the entire card. Modern data entry systems allow the user to specify the verification of only specific fields, bypassing noncritical data. It is also possible to bypass verification of fields that have already passed validation tests such as hash totaling.

Conversion and Preprocessing

Though it appears trite to say so, processing errors are best prevented by good design of hardware and software. Mechanical errors are detected by parity and cyclic redundancy checks (CRC) in the hardware and by defensive programming to protect the data area and itself against equipment malfunctions.

Errors from the detection algorithm itself can be prevented by good systems design and procedures that allow for all possible convolutions of error conditions and detection/correction procedures. A corrected first record of a duplicated field can be propagated through all duplicated fields either automatically, by resolving the duplications after all corrections are over, or by forcing the operator to correct every successive dup field. Arithmetic errors can be resolved by a separate total pass or by maintaining running totals that get corrected every time a component field is corrected, etc. If this type of error occurs frequently, a user must also be able to force reverification of a field or record that has had major corrections made.

The data can be protected from external causes such as power failure by incorporating extensive checkpoint recovery procedures in the system itself, and by duplication (redundancy) of critical library information.

Transfers

Transfer errors such as communication errors are generally taken care of by built-in detection and correction procedures. On a larger scale, proper identification of batches of data by appropriate labeling is done very well in shared processor systems, which can be programmed by the user to allow the creation of various types of labels from the supervisor's console or from within the system itself so that the resultant data stream meets the compatibility requirements of the host processing system. This not only reduces errors but actually lessens overall cost by cutting down or eliminating a separate editing pass at the main processor or elsewhere in the data entry system.

Indirect Errors

The power of the processor in a shared processor data entry system can be used extremely effectively to allow the user great latitude in changing his data entry procedures, keying procedures or input forms without introducing errors. Most key-disk processor systems have extensive reformatting capabilities. Input data is stored in intermediate storage on disk, reformatted as directed by the user, and output.

If data entry procedures change, new data is added, some old data removed, or if input forms change, the user only needs to alter the output editing formats and the output would still fit the processing programs. Similarly, if a new application program uses the same input data but in a different format, this too can be accomplished by merely using a different output edit form. In all cases, the software modules that do individual editing on the data are already written and part of the system; all the user must do is enter an editing form that tells the system what editing facilities are to be used and in what sequence. The use of disk for intermediate storage, and the editing formats, release the user from the stifling grip of the 80-column card format. The greatest boon to the user is that the shared processor key-disk system can effectively decouple the output format from the input so that the operator can "key what you see" and yet the output is computer compatible. Single programmable stations are generally unable to provide the same complex or extensive reformatting capabilities as shared processor systems because of cost considerations. But they can also be programmed to allow for changes in input procedures and the program can be stored in auxiliary storage if necessary (like cassettes) to be used one at a time.

Illegal Entries

Illegal entries are generally checked with check digit procedures and cross-field validation procedures in which the type of information in some critical field is keyed to coded information of another. Error detection methods therefore are generally based on the nature of the data such as length, value, etc., to provide validity checks and on reverification procedures. Errors induced can be avoided or reduced by extensive checkpoint restart capabilities and reformatting.

CORRECTION PROCEDURES

This section categorizes the correction procedures commonly available, with general comments on the characteristics of each.

Spot Correction

Spot correction procedures are almost always the first method if a proper one exists. All keying controls fall into this category. Also, there are validation procedures which provide for spot correction.

A validation error will stop the entry operator immediately and allow it to be rekeyed. Usually this corrects the problem. However, if the data itself is wrong, then the operator must be allowed to proceed anyway since it cannot be assumed the operator can correct or even identify the error. This separates this class of error from keying controls.

Error Marking

As just mentioned, it is not always possible to correct errors when detected. The next best procedure is to mark them for later correction. This may be done in a variety of ways. The entry operator can be instructed to physically mark the document, overkey the field with an error indication, or delete the record and remove the document. A slightly better procedure might be to allow a format position following the field which is checked, so that a special character can be inserted if an error is given. A more elegant procedure is available on key-to-disk equipment. As data is stored on disk in an intermediate form, it is possible to append control characters in a way that is essentially invisible, i.e., not occupying any format position, and automatically controlled. Thus if a validation error is signalled and the operator does not correct it, the field is so marked. This mark is then used generally in two ways. First, the batch can be later reopened and the system instructed to search for the first error record (or the next error record, etc.) Second, during output the reformatting features can test for the error flag and do somthing entirely different if an error is present. This allows great flexibility in error control procedures.

Second Pass

Simple verification is unique because it requires no knowledge of the meaning of the data. It merely checks to see that the entry operator and verify operator transcribed the data the same. It requires full rekeying, which no other method does.

Data Separation

A technique is available on key-disk systems, particularly, to separate data containing errors from those that don't have errors. This usually is done on a batch basis and usually involves batch balance errors. The system informs the supervisor which batches have errors and which don't. Then the supervisor can pool the error-free batches onto tape under control of one reformatting procedure. For the batches with errors a separate reformatting can be used, and then the data is typically output on a printer. This procedure is designed to allow sight checking of the data. When the error is found the batch can be corrected and output under the first procedure.

CURRENT EQUIPMENT CAPABILITIES

Table I lists four different types of data entry equipment. The capabilities are listed in three groups for each. The first group are those commonly available and the second group are additional feature available on some systems. Group three are additional features which prevent errors in the handling, status, and records of larger units of data such as complete batches.

There is a substantial increase in capabilities of the key-disk systems over the card punch and the key-to-tape, which are hard-wired devices that cannot be easily or cheaply enhanced. Key-disk systems share a normal processor which can be reprogrammed as required.

Source data entry systems can be program-specialized to one application, but this could be expensive and attended by debug problems.

Some source data systems, however, are very simple and quite standard because the operator understands the data and can correct detected errors without aids.

Card Punch	*Key toTape*	*Key to Disk*	*Source Data Entry*
Verify Case control	Verify Case control One-field batch balance Check digit	Verify Case control Batch balance Check digit Allowable character set	Operator knowledge
One-field batch balance Check digit		Multiple batch balance Subtotal balances Crossfooting Special character sets Range check Value set Automatic program linking Reformatting Error testing on output Must-be-present field Minimum length field Cross field validation	User programmed
Permanent copy of data		Hard record of batch status Record of supervisor actions Detailed statistics on each batch Labeling procedures	Essentially mated to processing programs

Table I. Data Entry System Error-checking Capabilities

IMPROVING THE PROCESS

A system must be understood to be improved effectively. Better-managed data entry facilities have always kept detailed records on each phase of operation. Yet the process itself of keeping these records is a laborious and costly process subject to some error. Modern data entry systems are capable of recording very complete statistics as a normal part of operation. The planning of any data entry system must include provision for statistics collection as feedback for system improvement, cost justifications, etc.

The statistics provided by a data entry system should be available as either raw or processed results. Raw statistics include such information as operator identification, batch identification, start and stop time, number of records, number of keystrokes, number of keying errors, number of validation errors, number of verifying errors, and so forth. This allows the user complete freedom to do later processing as he requires.

Processed statistics could analyze the raw statistics by operator, for example, providing common measures of work done, keying rates, error rates, etc., for each person. Good performance can be quickly recognized and rewarded. Better control of training can be realized on an individual basis by recognizing problems earlier. Perhaps operators can be matched to the difficulty of jobs. This data may allow the supervisor to achieve better data integrity levels without any system changes.

The systems analyst will more likely want to see the statistics analyzed on a job type basis. By averaging over many operators this gives direct information on the difficulty of a particular job. If the keying rate on one job is substantially lower than others, a need may exist to redesign the forms or alter the associated procedures. Keying rates are affected by data type, so this conclusion does not always follow. Jobs of similar types of data should be compared. Careful inspection of the types of errors incurred will discover other problems. It may be the people originating the form are careless, or the form requires data frequently not known, or the user misinterprets instructions on the use of the form, etc.

Finally, the analyst has a complete source of statistics on the volume of data by job and total errors by type on each job. Since this data is constantly being generated by the system, the analyst always has a current picture of his part of the process. A major problem in proceeding from this point is to realize that it is only part of the data integrity problem. Since no entry system is perfect, there will be errors going on to succeeding processing steps. The problem is to establish quantitative measures of either acceptable error levels or a cost/error basis. These measures will probably be difficult to establish and will usually be estimated on the low side. It is difficult to trace the full effect of an undetected error and assign true costs. However, it is necessary to have these numbers to give the systems analyst a solvable problem.

Ultimately, these concepts must be reduced to costs for evaluation. A useful approach is to assign costs based on a keystroke. The number desired has four main components:

S = Operator Salary/Month

D = $\frac{\text{Data Entry System Cost/Month}}{\text{Number of Operators}}$

R = Average keystroke rate in keystrokes/hour

H = Working hours/month for an operator

$C = \frac{S + D}{HR}$ = cost of a keystroke

Typical numbers might be:

$C = \frac{650 + 70}{160(8000)}$ = \$0.00056/keystroke for keypunch equipment

$C = \frac{650 + 150}{160(10000)}$ = \$0.00050/keystroke for key-to-disk equipment

Several comments may be made here. Keystrokes are physical key depressions and not the number of format positions, e.g., not 80 to the card. Note that doubling equipment costs is more than balanced by a 25% gain in keying rate. Such gains are commonly experienced in direct conversion of a job from punch to key-to-disk before changing the job to take advantage of new capabilities. All of the numbers in the definition of C are easily available or measurable. Using this definition allows the data integrity cost to be attacked as a function of only two major factors, number of keystrokes required to achieve a given level of data purity and the cost of the errors that remain (which must be provided or a negotiated estimate made). Questions of rates, etc., are eliminated from further discussion to clarify the problem at hand. Also note that if equipment is used more than one shift the cost is simply lowered for each operator.

Each field in the document has its own needs with respect to data integrity. The type of errors possible, the cost of an undetected errors, and the applicable detection methods must be considered on a field basis. Table II gives four commonly used types of error detection, with some of their characteristics. It illustrates the concepts one must consider in making a proper choice (all items in Table I must be viewed in a similar manner). Several types of errors can apply to a single field.

Type	Isolates Error to	Error Type Detected	Approximate Expected Number of Undetected Errors/Batch (for 1 field)
Full Verification	Character	Transcription	$\frac{FN\ E^2}{10}$
Batch Balance	Batch	Transcription	$\frac{FE^2\ N(N-1)}{40}$
Check Digit	Field	Transcription, Data	$\frac{NE^2\ F\ (F-1)}{20}$
Range Check	Field	Gross data	NFE

E = Probability of keying error on a specific keystroke.
N = Number of records in a batch.
F = Average number of data keystrokes in the field.

Table II. Typical Error Detection Methods

It should be noted again that the data entry system can be used to measure these expected error levels directly by controlled experiments. Let us hypothesize a simple example and see how such calculations might be applied.

C = $0.0005/keystroke

160 hours/month

Batch = 200 documents

100 Batches/day

E = 1/2000 = .0005 (an average of 1 in 2000 keystrokes is in error)

Only one type of job is being done, one document containing:

Field	*Item*	*Max. Length*	*Average*
1	Acct. No.	10	10
2	Customer Name	20	12
3	Amount	10	6
4	Selling Dept.	3	3

For this simple analysis we will ignore control keystrokes such as skip, dup, record release, etc., which depend on type of equipment being used.

Field 1 must be correct or considerable problems will arise. Billing will be incorrect in two accounts, with considerable customer dissatisfaction. By the time the error is found, traced, and corrected, it is estimated that $100 of labor and computer costs are involved, plus intangibles.

Field 2 can tolerate errors with little problem. It is used for sight checking and not for detecting the proper account.

Field 3 clearly must be correct. Errors would cause other totals to be out of balance in later processing, causing much data to be rechecked. An error is estimated at $500.

Field 4 can be in error with little problem. It is used for cross-checking only and, if in error, it can be traced with modest effort.

Results of applying the given formulas to this example are shown in Table III.

Even such a simple analysis highlights several interesting points. It is good discipline to be forced to quantify arguments on data integrity so the choice can be analyzed. Undoubtedly one will comment that a certain figure in the example is arbitrary. So will another, but pick a different parameter. One point of the analysis is to bring out hidden differences in opinion which might be resolved by some direct measurements. Doing the analysis with two different sets of parameters may show that the controversial number is unimportant to the decision. Then one can refine the accuracy of the important ones.

For example, the $100 and $500 costs/errors are very rough estimates, i.e., 1-digit accuracy. However, great changes in these numbers will not affect any of the conclusions drawn except in Case 5. It is obvious from Cases 1 and 2 that error checking is required. Case 3 shows that errors are only important in certain cases. Case 4 shows a check digit to be a valuable addition. Case 5 is a mild surprise. One can say that $500 is unrealistically high and tipped the scales. A more penetrating observation in Case 6 is that if the data could be provided with 4 subtotals as well as a grand total, and if the data entry system had subtotal checking capability, the cost/per batch would be 3.50 and a clear winner. The reason is that the batch balance error is sensitive to batch length, and the error is more highly localized. The result is independent of volume as would be expected, but we must reapply the volume figure to appreciate the actual daily savings realized.

Items such as "must be present" field, minimum length specification, special character sets, etc., which considerably improve data integrity, can be added to the above simple analysis.

SUMMARY

We have presented the sources and types of error possible in a data entry system, the possible methods of detection, and procedures for correction. We have also indicated a quantitative method of evaluating the cost of errors in determining the extent of error checking needed in any system. With intelligent analysis of possible error situations and their costs, the user can set acceptable error thresholds and meet the objective of selective data integrity at an optimum overall cost.

Case	*1*	*2*	*3*	*4*	*5*	*6*
	No Checks	Full Verify	Partial Verify	Partial Verify, Check Digit	Check Digit, Batch Balance	Check Digit, Sub-totals
Entry cost	3.10	3.10	3.10	3.20	3.20	3.20
Verify cost	-	3.10	1.60	0.60	0.38	0.09
Field 1 errors/batch	1.0	1.0	1.0	1.0	1.0	1.0
Errors undetected	1.0	0.00005	0.00005	0.00027	0.00027	0.00027
Error cost	100.00	0.005	0.005	0.027	0.027	0.027
Field 3 errors/batch	0.6	0.6	0.6	0.6	0.6	0.6
Errors undetected	0.6	0.00003	0.00003	0.00003	0.00150	0.00037
Error cost	300.00	0.015	0.015	0.015	0.75	0.18
Total cost/batch	403.10	6.22	4.72	3.84	4.35	3.50

Table III. Cost Comparison

DATA INTEGRITY IN THE GIANT SYSTEM

by JED R. ALLEN and VERDON R. WALKER

Management Systems Corporation
Salt Lake City, UT

The Genealogical Information and Name Tabulation (GIANT) System has been operational since late 1969. The GIANT System is operated by the Genealogical Society of The Church of Jesus Christ of Latter-day Saints. Computer processing is done on an IBM 360 Model 65 computer.

The principal inputs into the GIANT System are "individual" records (records of births, christenings, or some other identifying events for individuals) and "marriage" records (records of a marriage or related event for couples). These records are submitted by patrons of the Genealogical Society and are usually a result of their personal genealogical research aimed at identifying and recording their ancestors.

For certain localities where there is a concentration of ancestors of members of the Church, the Genealogical Society itself has also input records on a controlled basis, thereby making them more accessible to persons attempting to identify their ancestors.

An example of the contents of a typical individual and marriage record is now shown.

Individual Record

Individual -	Thomas Crain
Father -	William Crain
Mother -	Joney Cowle
Birth Place -	Andreas, Isle of Man, England
Birth Date -	9 April 1761

Marriage Record

Husband -	John Johnson
Wife -	Alice Wright
Husband's Father -	Thomas Johnson
Husband's Mother -	Elizabeth ---
Wife's Father -	Robert Wright
Wife's Mother -	--- ---
Marriage Date -	12 Oct 1573
Marriage Place -	Fressingfield, Suffolk, England

The ultimate repository of these records, after they have been input into the computer system, is the Genealogical Mass File. This file not only serves as a repository but also as a clearing-house, in that incoming records are compared to those already in this file and if they are duplicates they are rejected. The Genealogical Mass File currently contains over 15 million individual records and 3 million marriage records. Approximately 3 million records are being added to the file each year.

The purpose of this paper will not be to provide a detailed description of all phases of the GIANT System but rather will be limited to the discussion of these aspects of the system:

- The steps taken to detect the duplicate input of records that are already in the file.
- The handling and organization of the Genealogical Mass File, which currently occupies more than 42 IBM 2314 disk packs.

DUPLICATION DETECTION

The Genealogical Society maintains and continually adds to a large collection of source genealogical records. These records are available in its library system for use in ancestral research. The same source could be used at different times by many different people. As people identify and submit records of their ancestors further and further back in time, the probability increases that ancestral lines of apparently unrelated people will begin to merge. These factors mean that there is a certain probability that an incoming record may already be in the Genealogical Mass File.

In order to maintain the logical integrity of the file, much care is taken in the GIANT System to prevent duplicate records from being added to the file. The Genealogical Mass File will provide a data base of genealogical records organized in such a manner as to be useful for future analysis and research.

In order to be able to handle variations in the spellings of names of people and localities, records are processed by a locality subsystem and a name subsystem before they are compared to records already in the file. The locality subsystem is based on a large computer catalog of locality records,containing variations of town, county and country spellings, each with an associated standard spelling. The actual locality spellings in each incoming record are compared to the locality catalog and the standard spellings found there will replace the actual spelling in the record. If the actual spelling is not in the catalog it is printed out for evaluation by geographers. The geographers will add the actual spelling to the catalog with its appropriate standard spelling. The new actual spelling could be a variation of an existing standard locality or it could be a new locality not yet in the catalog.

Along with the standard spellings, the locality catalog also contains a longitude-latitude coordinate code for each town in the catalog. This code is also added to the record and serves as a fixed-length code for the town. The coordinate code also provides the capability for distance calculations which, although not currently used, will be of value in the future for making searches within specified distances of a given locality.

Some examples of variations of locality spellings are given:

New York

N Y	N. York	New York
N Yk	N. Yk.	New York.
N York	N. Yrk.	Newyork
N Ykr.	N. Y.	Nw Yk
N Yrk.	N.york	Nw Yrk
N. Y.	N-Yrk	

Rensselaer County

Renns.	Renslr	Renssaler
Rennsalaer	Renslr.	Rensselaer
Rennselar	Renss	Rensselar
Rennsl. Co.	Renss. Co.	Rensseleer
Rens	Renssalaer	Rensselser

Petersburg Town (4245N 07320W)

Peterburg	Petersburg	Petersburgh

The name subsystem operates in somewhat the same manner as the locality subsystem. The basis of the name subsystem is the name catalog. This catalog contains a given and a surname portion, and within each there is an additional geographic breakdown according to linguistic group such as, U.S.A. and Canada, British Isles, Central European, etc. Name variations can be dependent upon the general linguistic area, so the provision has been made that different standard spellings can be assigned to the same actual spellings for differing localities.

As in the locality system, all incoming names are matched to the name catalog. Names which are already in the catalog will pick up the standard spelling and a fixed length standard spelling code from the name catalog. If the actual spelling is not already in the catalog, then it is printed out for manual evaluation. The standard spelling is determined and added into the catalog. In many cases the standard spelling of the name is the actual spelling. There are currently over 1,250,000 surnames and nearly 500,000 given names in the name catalog.

An example of some of the variations of names are given below:

George

Gaorge	Geog	Geordg
Garge	Geoge	Georg
Geaorge	Geogre	Georg.
Gearge	Geoirge	George
Geirge	Geor	Georgh
Geo	Geor.	Georgi

Hamilton

Heamilton	Hamilinton	Hammilton
Hambelton	Hamiliton	Hamolton
Hambledon	Hamilten	Homilton
Hamblton	Hamilton	Hammelton
Hamelton	Hamliton	

In the locality system the actual spellings are not retained but are actually replaced with the standard spellings in the individual and marriage records. In the name system the actual spellings input are always retained in the record. The standard spelling code is used, however, in the actual comparison made for duplicate records.

After localities have been standardized and the name standard spellings and codes have been added to the records, they are ready to be compared to records already in the file. Any record in the file will have previously had it's locality and name spellings standardized. A statistical approach has been taken in making the decision of whether two records are actually of the same individual or couple. A table of "uniqueness factors" is used in the GIANT System. These uniqueness factors are currently relatively broad but can be refined as more data is available for analysis. Basically a uniqueness factor is assigned to each item of identification, such as, month of birth, given name, surname, father's given name, etc. This uniqueness factor is roughly the proportion of people who may have that identifier in the population being considered.

The uniqueness factors for the common information between an incoming record and a possible duplicate record in the file are used to computer the probability that the common information would identify not more than one individual or couple. There are some items of identification that cannot conflict without invalidating an otherwise duplicate situation. There are other items which can conflict and if the uniqueness value is sufficient, the duplicate condition will be accepted. In general, it can be stated that the records available from genealogical sources being used ordinarily contain sufficient information to permit unique identification and duplication checking that is the statistically soundest.

GENEALOGICAL MASS FILE HANDLING AND ORGANIZATION

There are currently over 18 million records on the Genealogical Mass File. This amounts to slightly over 1 billion characters. The file is growing at the rate of over 3 million records per year.

The Genealogical Mass File has both a static file portion and a dynamic file portion. Records are in sequence on each of these files by locality, then name, and then date. The majority of records are always on the static file with only incoming new records being added to the dynamic file. The static file is segmented into segments of two disk packs each. During actual processing, only one segment of the static file is mounted at a time. There are currently 21 segments in the static file.

Each static file segment is organized in an "indexed sequential" manner. The standard IBM indexed sequential access method has not been used, however. Rather, records are blocked into fixed length blocks of 1682 characters. Four blocks occupy one track and 80 blocks occupy one cylinder of the IBM 2314 disk pack. Blocks of data are read using relative block addressing. One block per cylinder is an index block indexing the other 79 data blocks on the cylinder. There are also 5 index blocks on each segment which index the 400 cylinders on the segment.

Since all new records being added to the Genealogical Mass File go into the dynamic file, there is no need for empty or overflow areas to be left in the static file. Other than a small amount of unused space left in each fixed length data block when the variable length records are blocked together, all space on the static file is used.

In comparing incoming records to the static file, the indexes are always used to eliminate the necessity of reading every record in the file. The incoming records are in the same sequence as the file, so all processing against each segment is done in turn. After each segment is completed, it is dismounted and the two disks of the next segment are mounted.

The records in the dynamic file are always blocked in the same manner as the records in the static file. The dynamic file is always read sequentially, however, so no index blocks are needed. Each time that a group of new records is processed against the Genealogical Mass File, the entire dynamic file is read in sequence and all records from it are merged with those new records to be added. These records are then written out on a new output dynamic file.

The static-dynamic file approach does require the checking in two places for duplicate records. It does eliminate, however, the need for writing new records in each processing run into a file that is as large as the static file. If this were required, it would mean the leaving of empty spaces through the file to accommodate the new records to be added. The problem would be particularly acute in the GIANT System since the input does tend to cluster. A person may submit a large number of records with the same surname from the same locality. Unless they duplicate, they would all go into the same area in the file. This problem is even more severe with batches input by the Genealogical Society which could contain several thousand records from a particular locality.

The dynamic file, of course, becomes larger each time a new group of records is added. Periodically a Genealogical Mass File reorganization is run in which all records on the dynamic file are merged with the static file and a new larger static file is created. The dynamic file then begins over from zero. Currently this Genealogical Mass File reorganization is done about twice a year in the GIANT System.

A major advantage of the static-dynamic approach is that the static portion of the file can be considered as a read-only file during regular processing. This contributes to the integrity of the file by insulating it from logical programming errors. It will also lend itself well to the possible future use of read-only mass storage devices.

CONCLUSION

The operation of the GIANT System has not been without some problems. However, the integrity of the data being processed by the system and being stored on the Genealogical Mass File has always been maintained to a very high degree.

It is hoped that the techniques and approaches used that have been described in this paper may be of value to others who are designing and operating similar large data handling systems.

DIGITAL RECORDING RELIABILITY FOR INFORMATION EXCHANGE APPLICATIONS

by ROBERT T. McKENNA

Goddard Space Flight Center
Greenbelt, MD, US

In the past several years both government and industry have been using computers with digital tape drives which record at 800 characters per inch (cpi). As more equipment of this type is used, complaints of read failures from users increase. The most frequent complaint is that users experience problems while attempting to read tapes which have been written on a computer other than their own, or sometimes within their own facility.

The purpose of this paper is to identify the basic cause of this problem. It also will offer some solutions and alternatives to alleviate these complaints.

At Goddard Space Flight Center (GSFC) our primary mission is launching and performing telemetry data processing of nonmanned scientific satellites. In performing this function we ship each year approximately 75,000 digital tapes of processed data to experimenters involved with the onboard experiments, located in government agencies, universities, and foreign countries. We generate these tapes on computers built by several different manufacturers. Likewise, the experimenters read these tapes on many makes of computers of both domestic and foreign manufacture.

We have been doing this type of processing for twelve years, during which we have encountered all types of interface and compatibility problems pursuant to interchanging information through these digital tapes. Because of these problems, we invested engineering time and effort to develop special hardware for the purpose of making measurements of the recorded data. These measurements are made of the amplitudes, character skew, character spacing, asymmetry, special format characters, interrecord gaps, and specified recording formats. This specialized equipment is located in a laboratory which we call our Digital Tape Unit Test Facility. This facility not only performs routine tests but also does special tests on tapes and tape drives to resolve problems existing between computers managed by NASA as well as other government agencies and industry. Presently we test 250 tape drives on a routine scheduled basis and perform 200 to 300 special tests each month.

Our test results have shown that the most common cause of compatibility problems is a condition called skew. Skew is a product of two factors whose end result is that the characters are not recorded perpendicular to the edge of the tape. These two factors are static skew and dynamic skew. Static skew is determined by the physical alignment of the recording head and tape guides, and also the alignment of the individual track recording gaps within the head assembly (gap scatter). Dynamic skew is a result of the wandering and squirming of the tape as it passes across the recording head. Additional factors are the asymmetry and pattern sensitivity of the recording head.

The sum of these two is total skew, usually just called skew because in application they are inseparable. Figure 1 shows tape characters and the effect of skew.

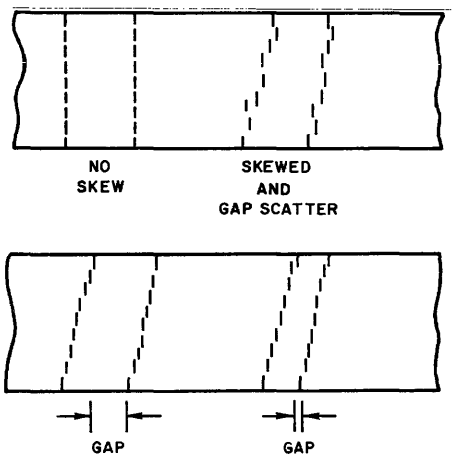


Figure 1 Tape characters and effects of skew.

Having identified and described "skew", what part does it play in computer tape compatibility? First, it shows up to the computer user as parity errors and changes of record lengths. The physical problem is that the reading system cannot separate one tape character from another, resulting in the splitting or omission of characters. Referring to Figure 1, you can see that as the packing density increases, the space or window between characters decreases. A reasonable distance or gap between characters must exist. This is to allow for reading termination of a character and preparation to read another, as well as providing a tolerence for the read head assembly's skew.

I will present some facts and figures which will demonstrate the almost impossible task a manufacturer is confronted with in trying to maintain consistently reliable tape drive operations. I do not say that it cannot be done, but if it is done it requires constant adjustment and maintenance of varying degrees, sometimes approaching equality of down time with up time.

Figure 2 shows two methods of recording. The basic difference is that phase encoding has a transition for both "ones" and "zeros", whereas NRZ has a transition only when a "one" is recorded. The advantage of phase encoding, as related to skew, is that each track is self-clocking and the gap or window between characters is not as critical.

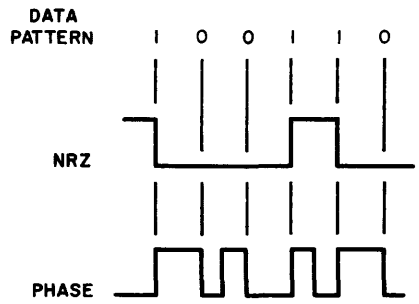


Figure 2 "Non Return to Zero" and "Phase Encoded" recording methods.

800 cpi packing density requires a character to be written every 1250 microinches ±3 percent. From Figure 1 it can be seen that to separate one character from another, a space must exist between characters so that no ambiguous condition or indecision can occur as to which character a bit belongs. This is fundamental. In addition, a tolerance for the skew and gap scatter of the read head must be allowed. Therefore the character skew must be less than half of the character spacing, minus an additional tolerance to provide for read head skew. For these reasons, the digital recording specification (American National Standard for Recorded Magnetic Tape for Information Interchange) specifies that the total write skew is to be less than 425 microinches. Figure 3 is a timing diagram, showing the read process when reading a tape written with a skew of 425 microinches.

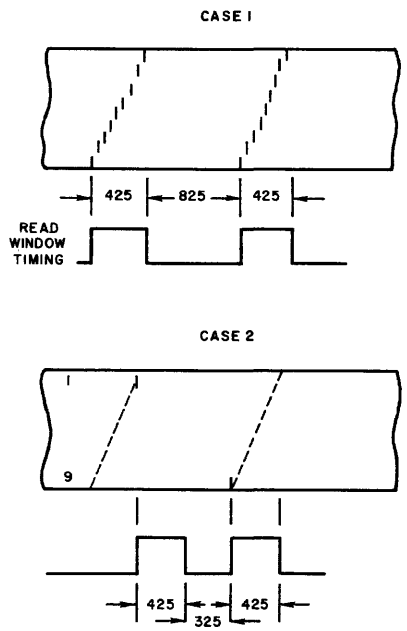


Figure 3 Timing diagram of read process.

Case I is typical. The read window is opened upon receipt of the first bit, which in this case is in track 9. The window must be open for at least 425 microinches in order to read a tape written at the maximum skew tolerance. This provides a nice wide distance of 825 microinches between characters to provide for the read head skew. Case 2 shows the critical problem.

The recording specifications state that the character spacing tolerance can be ±3 percent. The manufacturers specify ±3 percent of nominal tape speed for their drives. This means that -6 percent can be expected for the character spacing as related to time. The effect of speed variation is to reduce the time between characters. Case 2 is the worst case; a character with a bit in track 1 only is followed by one with a bit in track 9. Upon receipt of the bit in track 1, the read window must be opened for a period of at least 425 microinches as related to the tape speed, leaving only 400 microinches to the next character, whose first bit is in track 9. Within this 400 microinch gap you must subtract the -6 percent tolerance, giving the effect of subtracting 75 microinches. Now the space between the characters is effectively 325 microinches. Therefore, to maintain reliable operation, read head static and dynamic skew must be less than 300 microinches. This is closer than the write skew tolerance, which is 425 microinches.

Now we know what tolerances these tape drives must perform to, let us apply them to existing equipment and see if they can reliably and consistently hold the tolerances. Figure 4 shows histograms for six computer systems at Goddard Space Flight Center. These systems are made by the three largest manufacturers. The tape drives are their standard drives, maintained by their onsite customer engineers. The point is that these tape handlers are representative of the best equipment produced and are maintained in accordance with the best maintenance methods as provided by the manufacturers. It must also be pointed out that all of these manufacturers quote skew tolerance specifications with write skew less than 425 microinches.

You may evaluate this to be fact or fiction from the histograms in Figure 4. The data shown was accumulated during 1971. The horizontal axis represen 52 weeks. The vertical axis is the write skew measurement made at 112.5 inches/s. The test equipment calibration error for skew measurement is 45 microinches.

57 tape drives were tested (each from two to six times a week) for a total of 12,963 tests, as follows:

(1) Three blocks of 960 characters each are loaded into core. These consist of three patterns, one for each recorded record: all ones, checkerboard, and a pattern sensitivity test pattern.
(2) The computer outputs these patterns to each of its tape drives, writing these three patterns over until 960 records are written.
(3) These tapes are then taken to the Digital Tape Unit Test Facility where the skew is measured.
(4) The normal routine is that each drive is tested twice per week.
(5) Should the skew of a tape exceed the established threshold, that drive is then taken "offline" and maintenance is performed.
(6) When maintenance is complete, the drive is retested and must perform within specifications before being placed online again.

To interpret the histograms: Each week is represented by two vertical columns with the letter "H" used as a filler. The value range is from 01 to 99 with an * for values in excess of 99. So, for any week, it shows the number of tapes tested and the corresponding write skew measured. In making these measurements, the digital test transport moves the tape at 112.5 inches/s. So we must convert the physical dimensions of microinches to time base measurements of microseconds. Now the skew is measured in microseconds as shown on the vertical axis.

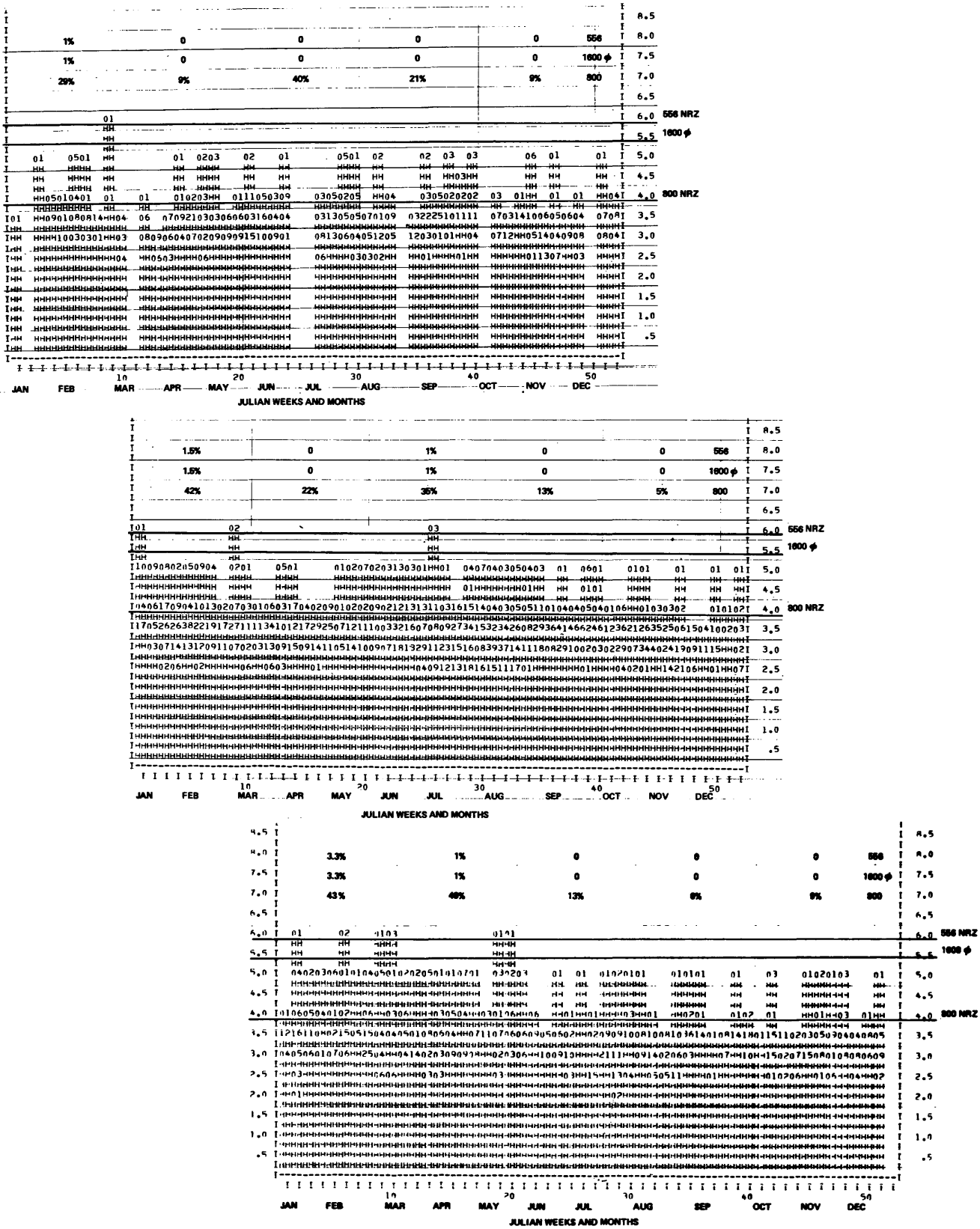
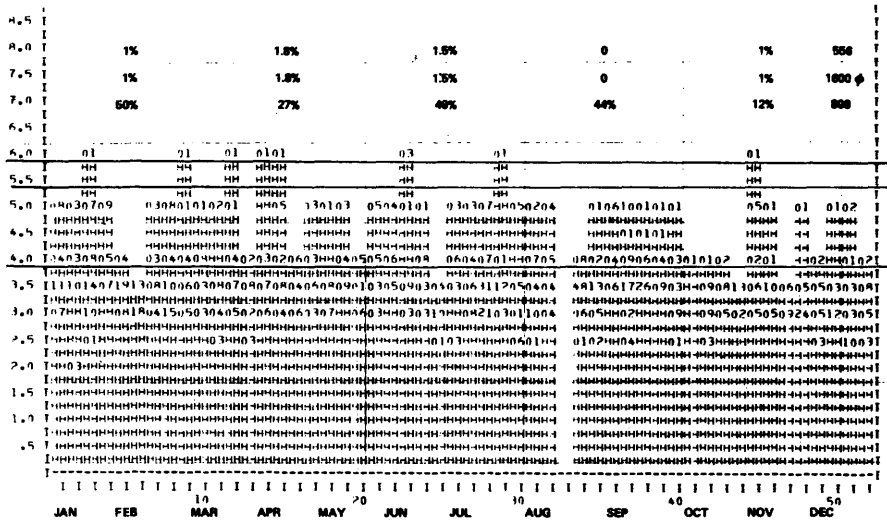
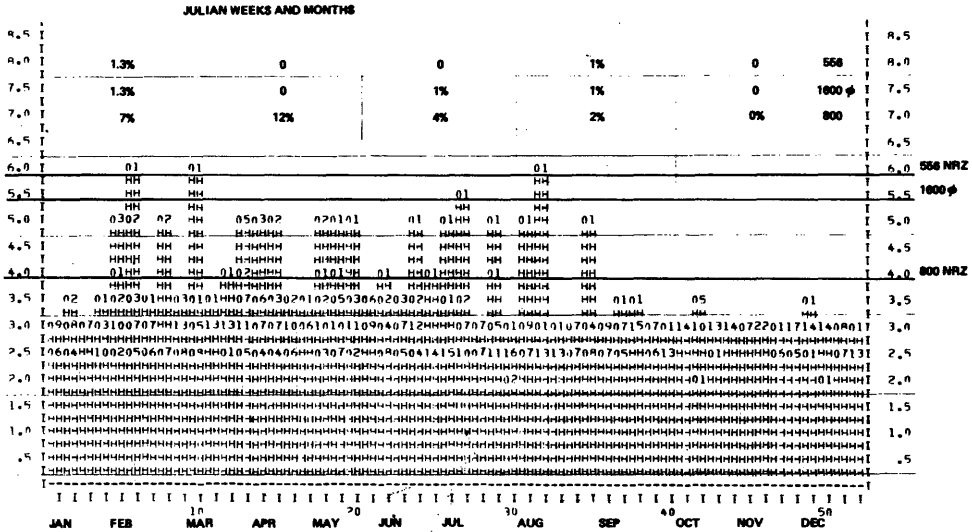
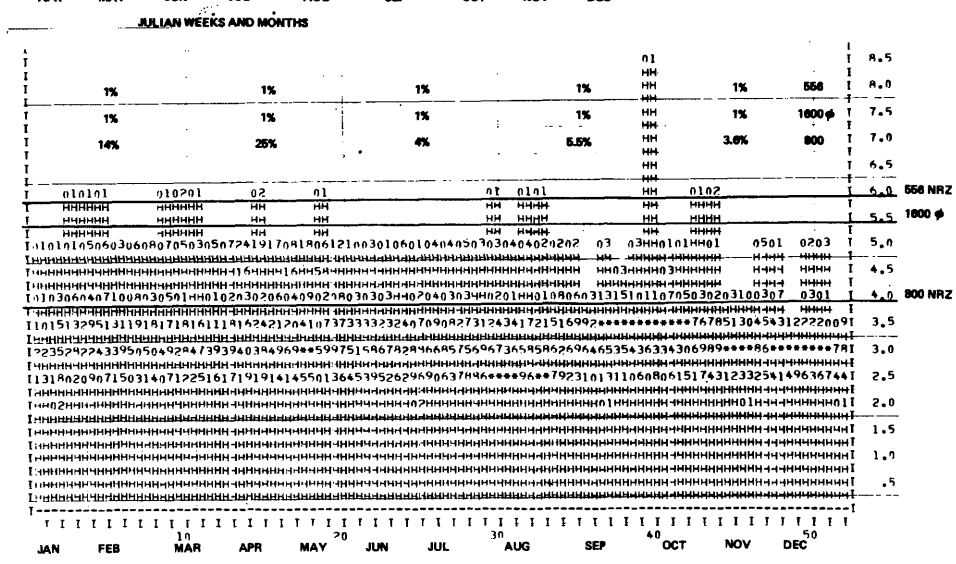


Figure 4 GSFC computer systems histograms.


1%
1.8%
1.8%
0
1%
556
1%
1.8%
1.8%
0
1%
1600 ϕ
50%
27%
40%
44%
12%
800
556 NRZ
1600 ϕ
800 NRZ
JAN
FEB
MAR
APR
MAY
JUN
JUL
AUG
SEP
OCT
NOV
DEC
JULIAN WEEKS AND MONTHS


1.3%
0
0
1%
0
556
1.3%
0
1%
1%
0
1600 ϕ
7%
12%
4%
2%
0%
800
556 NRZ
1600 ϕ
800 NRZ
JAN
FEB
MAR
APR
MAY
JUN
JUL
AUG
SEP
OCT
NOV
DEC
JULIAN WEEKS AND MONTHS


1%
1%
1%
1%
1%
556
1%
1%
1%
1%
1%
1600 ϕ
14%
25%
4%
5.5%
3.6%
800
556 NRZ
1600 ϕ
800 NRZ
JAN
FEB
MAR
APR
MAY
JUN
JUL
AUG
SEP
OCT
NOV
DEC
JULIAN WEEKS AND MONTHS

Lines are drawn across the histograms at the maximum skew tolerance for 556, 800, and 1600 cpi phase-mode recording. Looking specifically at 800 cpi NRZ, I have calculated the percent of failures for tapes tested to meet the write interchange specification (425 microinches, or 3.8 microseconds at 112.5 inches/s). The downward trend as time progresses could be an effect of improved testing procedures and maintenance. In our experience, a failure rate of 3 to 4 percent is considered tolerable for our application. This would vary at other facilities, depending upon needs and applications. I do not present these figures as typical of all systems, but rather of a well-adjusted system, because of the unique and persistent maintenance at our facility.

Many government agencies, and occasionally industry, have asked us to resolve compatibility problems. This is done by testing both reading systems and writing system to find which one is not operating within specifications. From this experience, but lacking statistics, it is my opinion that many systems would operate with a disastrous rate of unreliability should they be required to interchange data using 800 cpi digital tapes NRZ.

In addition to these write performance tests, we also conduct a read performance test that measures read head skew and read window width. This is done by generating a special test tape, written so that ten records are written with zero skew. Then successive 10-record groups are written displacing track 1 by 56 microinches for each ten records until track 1 is skewed 1000 microinches. Then each successive track is treated the same. The net effect is that we have rotated a character so that it is skewed in one direction, then in the other. We now read this tape using a special computer program which types out the read analysis. This gives both read head skew and read window width.

The data collection for the read performance is not available for presentation now. When it is, one can overlay the histogra any system's written tape results on another's read ability test histogram and quickly determine the compatibility or probable failure rate percentage. Significantly, the write performance histograms show that these systems cannot consistently write within specifications. So it is probable that they cannot consistently read within the 300 microinches tolerance.

Having identified the problems and described the precise tolerances that a tape drive must maintain in order to write tapes which meet the recording specifications, what can a facility do to help itself overcome these problems? Several things:

(1) *System Planning.* When planning for data interchange, consider the difference in reliability. Skew tolerance for 1600 cpi phase mode is 625 microinches, as opposed to 425 for the 800 cpi NRZ mode, and thus much more reasonable to maintain. From the histograms it is evident that the same drives can consistently meet the specifications for 1600 cpi but not for 800 cpi NRZ. Don't overlook the additional advantage of being able to correct one or more bit dropouts using phase mode.

Systems which use incremental recorders should never record at 800 cpi. For several years I have tested many models and makes of these recorders. In my experience, they are not able to record at 800 cpi without very high incidence of failures, often approaching inability to process any data at all. The problem is that the tape is being subjected to an instantaneous speed variation condition at recording time. The solution is to record at 556 cpi or preferably 200 cpi, and then submit the tape for duplicating on another computer system to generate the required output tape. This defeats the requirement to generate the output tape at the original source, but after one experiences many frustrations and delays by not being able to process the original tape it may not seem so unreasonable.

(2) *System Maintenance.* To have good maintenance you must not simply rely on the customer engineers to perform only the maintenance contract specifications, independent of user control. A good working relationship must be established between customer engineers and the data center manager. This can be done by someone without a technical background. Making a list of critical adjustments and calibration is required. Then a schedule should be made for the required frequency of these maintenance tasks. Someone must follow through to ensure that no oversight or deficiency exists. This may sound basic, and is in fact presumed to be a normal method of maintenance, but I have visited facilities where maintenance on these critical adjustments is performed only remedially. If the start time and the read and write skew alignment are checked once or twice per week, fairly reliable interchange can be expected in most cases. The master skew alignment tapes, the basic tool of the customer engineer for aligning the read and write skew, are sometimes worn so badly that measured skew of these tapes exceeds 225 microinches. It is good practice to retire these tapes on a scheduled basis. The schedule depends on the number of drives within a given facility, as the wear rate in turn depends on the number of drives serviced.

(3) *Trouble-Shooting Technology.* It is important that a computer facility manager have some ability to identify the most common types of problems. This requires that he covers only some of the basic types of compatibility failures. E.g., read parity errors. Where do they occur? Throughout the data record or only at the beginning? In this way a problem could probably be identified as a skew problem or a start-time problem. Some implementation of follow-up procedures is necessary to ensure that each piece of equipment is maintained on schedule and not neglected. One may also write a tape on each drive and see if it will read on all other drives within the facility. This does not guarantee compatibility with another system, but it is better than no test at all. A latent benefit is customer engineers' recognition that the computer manager is interested and concerned, which usually yields better and more responsive maintenance.

(4) *Tape Logistics.* Many good tapes are destroyed by storage facility temperature variations. This can cause physical damage to tapes, recognizable by characteristics of spoking or cinching. Do not permit tapes to remain in trucks or shipping containers outside of your environment-controlled facility any longer than necessary.

ARCHIVAL PERFORMANCE OF NASA GFSC DIGITAL MAGNETIC TAPE

by WILLIAM B. POLAND, JR. GILBERT E. PRINE and THOMAS L. JONES

NASA/Goddard Space Flight Center
Greenbelt, MD

Litton Industries
College Park, MD

Wolf Research & Development Corporation
Riverdale, MD

INTRODUCTION

For most of the last decade, the Goddard Space Flight Center (GSFC) has accumulated data from scientific spacecraft at a rate of approximately 10^{12} bits per year. This has resulted in an influx of instrumentation magnetic tape at a rate of approximately 2000 miles per week, and an efflux from the Center to experimenters of approximately 2000 partially filled reels of processed digital computer tape per week. The same data are archived in compact form, along with computer programs, orbit/attitude data and other related items. Scientific data tapes have now been accumulated in substantial quantities since 1958, along with tapes for manned and other spacecraft, and now constitute an archive of several hundred thousand reels of computer tape.

For a limited period in the early 1960's, a part of the accumulating archive was kept in space which did not have well-controlled temperature and humidity However, by far the larger portion of the GSFC archival tapes have been kept continuously in air-conditioned warehouse space in plastic containers packaged approximately 7 to a cardboard box.

Virtually all of the tapes (including 100% of those studied) employ 7-track format and were recorded at a density of 200 or 556 cpi NRZI. They were obtained by competitive procurement based on a NASA specification. They were recorded primarily on Univac III-C and VIII-C tape units, but some were recorded on IBM 729 and CDC 607 tape units, and a few other types.

The purpose of this study has been to assess the archival performance of digital tapes in such a way as to determine the major categories of error mechanisms operative in our storage environment, to identify the strategy for optimizing use of the tape, and to determine quantitative expectations for operating tape performance as a function of age under the conditions which obtain in the GSFC archive. It is important to determine quantitatively the level of performance of digital tapes in order to be able to specify the level of performance which future magnetic tape or other storage systems must achieve in order to offer an improvement. Such data in quantitative form do not appear to exist in the current literature. Since large quantitities of tapes are being "rehabbed" (i.e., rehabilitated - erased, cleaned, reconditioned, rewound on new reels, and returned to service), it presumably will be more difficult in the future to obtain a large sample of randomly selected tapes from the GSFC archive.

This study is primarily applicable to the GSFC tape archive, but it is felt that our tape storage practices are typical for many installations.

Since about 1967, a program to study archival performance of digital tapes has been pursued at a modest level of effort. This report summarizes the initial part of the program in which the investigation was confined to determining the archival performance of digital data tapes recorded in normal operations.

EXPERIMENT DESIGN AND TEST CONDITIONS

The study consisted of two major divisions based on random samples of archival data tapes retrieved and analyzed over the period from 1967 to 1971.

Error Mechanisms

First, approximately 1200 reels of tape were tested on the GSFC Digital Tape Unit Test Facility (DTUTF). In these tests, both the mechanical and magnetic conditions of the tape were evaluated, using an IBM 729 Mod. VI 7-track tape unit, and areas containing detected magnetic errors were examined for corresponding mechanical damage (including cinching), dimples, oxide defects, and errors not accompanied by a visible defect. Of these tapes 7% were "defective", i.e., had 50 or more parity errors per reel, and overall 44% had at least one parity error. The distribution of parity errors with respect to error mechanisms is summarized in Table III.

Performance Evaluation

The second part of the study dealt with operational behavior of the archival tapes in a realistic data processing environment. Approximately 390 tapes from the archive were tested in the Univac 1108 telemetry computing facility. These were randomly selected data tapes recorded in support of various spacecraft projects from 1960 to 1970.

The measured quantity was the number of blocks (records) containing errors. This number depends strongly in some cases on the number of passes over a read-head which the tape unit must perform before a given block is considered to be erroneous. In these tests, only "permanent" errors were counted (i.e., only those errors which persisted after eight completed passes over the read-head). Tapes containing 50 or more blocks in error were merely categorized as high-error tapes and not studied further.

The tapes selected were recalled from the archive (the GSFC warehouse or the Washington National Records Center [WNRC]) in the normal fashion and were submitted for processing along with the evaluation program (TAPECHEX). Where possible, the tapes were selected by a semirandom scheme designed to give uniform coverage in time without introducing systematic data errors. From 1967-1971, 4 tapes were chosen from boxes containing 7 tapes stored at intervals of approximately one month. A random selection scheme determined the position in the box of the four tapes used, and the types of data tapes were selected to produce an even mix from the tapes archived during that time.

For the post-1966 tapes it was possible to determine the date of storage to within two weeks from the bookkeeping records, giving consistently even time coverage.

The data were gathered in such a way that various criteria might be applied in evaluating tape performance. Since no usable precedents were found, it was decided to apply an operating criterion which has grown up in the telemetry data processing area at GFSC as a matter of practice and expediency; a tape will be rejected if more than three parity errors are detected throughout its (nominal) 2400-foot length. As a result of rounding and unit conversion this criterion has been modified to 1 error/300 metres (0.0033 errors/m). as the maximum acceptable error rate. The technique employed is to count not the actual parity errors but rather the number of blocks (i.e., records) in error per unit length. Since the errors are usually widely spaced, the rate of occurrence of blocks in error approximately equals the rate of parity errors.

Several factors were not controlled or evaluated in the course of these tests: e.g., write current, head wear, read threshold, head magnetization effects, and playback amplitude.

STATISTICAL METHODS

To describe the probability distribution of tape length between errors, irrespective of position on tape, two interacting distributions were used, Weibull[1] and Poisson.[2] Although the most general expression for Weibull distribution contains three parameters, two sufficed to describe reliability with respect to tape length. Time-related behavior was determined by dividing the data into separate age groups, obtaining the Weibull parameters for each data set, and fitting a nonlinear function to the computed probabilities for the required error rate (0.003 errors/m).

Length between errors was determined after sorting the reels on days in storage and then testing them as one continuous piece of tape (i.e., as if the reels were placed end to end).

The Weibull probability function can reduce to other distributions in special cases (e.g., normal, negative exponential), and thus is ideal for fitting data, with one restriction: change in error rate must be monotonic with respect to a given variable for valid representation of reliability. Given the Poisson distribution parameters, and independence of the distributions, one composite or smoothed distribution function can be derived. A solution of two independent distributions can be generalized into a method for combining any number of distributions.[3]

In order to observe change in error rate with respect to position on a tape, a set of histograms was computed. Figure 1 shows the observed error rate compiled for 25-metre intervals along a reel of tape as related to time in storage. Four age categories were used for expediency. The error rate clearly displays at least two distinct and consistent features for all age groups. It is high in the initial 100 metres and low in the next 200. Another pattern appears in three of the age groups (300 to 400 m), and other features are indicated which might be delineated by more data.

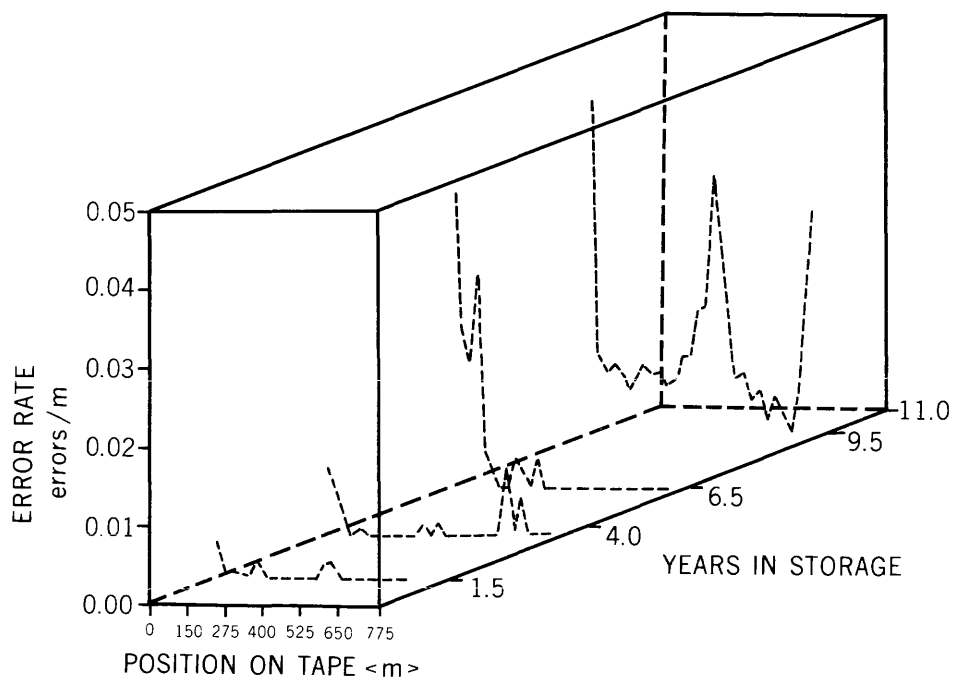


Figure 1. Isometric Representation of Error Rate, Position, and Age

To satisfy the requirement for a monotonic error rate, the analysis was carried out in such a way that each tape was effectively divided into zones defined by position in a reel, and the performance of similar zones for all tapes in an age group was analyzed.

The presence of correlated errors decreases the validity of the estimated parameters. Some editing was performed to help preserve the validity of the analysis. Errors occurring in consecutive blocks were treated as the occurrence of a single error mechanism.

Computer Program

A computer program was developed to handle the Weibull parameter estimation. The program separates the data into specified age groups, computes the length between errors from the position data for each zone, and then finds the Weibull parameters for each age group. Printer plots of positional error rates for each age group are displayed for evaluation.

ORGANIZATION OF THE PROCESSED DATA

For this study, error rates were investigated for five position zones on the tape and four age groups. The zones are:

(1) 0 to 100 m, the outer end region;
(2) 101 to 300 m, a generally good data region;
(3) 301 m to 400 m, thought to be the cinching region;
(4) 401 m to 600 m, a good data region except in the oldest age group; and
(5) 601 m to the inner end, the hub region where little data was available.

The age groups used were from (1) 194 to 987 days in storage, tapes archived about 1969-1971; (2) 1110-1731 days, 1967-1968; (3) 1869-2901 days, 1964-1966; and (4) 2929-4006 days, 1961-1963. The unsmoothed data for the analysis are summarized in Table I and Figure 1. A fuller presentation of the data and analysis are contained in the source documents.[4,5]

	Position (in 25 m)	1.5 Years Rate	1.5 Years # Reels	4 Years Rate	4 Years # Reels	6.5 Years Rate	6.5 Years # Reels	9.5 Years Rate	9.5 Years # Reels
	1 x 25 m	48.4	111	87.3	80	388.3	51	416.1	86
Zone	2	0.0	106	44.6	76	194.5	47	97.5	82
1	3	4.2	101	14.7	58	153.0	40	74.6	77
	4	6.7	76	0.0	49	271.1	34	85.7	70
	5	0.0	53	9.7	41	49.0	33	71.3	68
	6	24.2	51	0.0	41	25.0	32	55.8	66
	7	11.1	43	0.0	39	0.0	32	75.0	64
Zone	8	0.0	34	0.0	39	0.0	31	81.6	64
2	9	0.0	29	0.0	36	36.9	25	69.8	63
	10	0.0	2	0.0	33	19.0	21	77.2	63
	11	0.0	28	0.0	32	0.0	21	51.8	62
	12	0.0	27	0.0	31	39.8	21	66.6	61
	13	0.0	27	13.2	31	0.0	20	93.9	60
Zone	14	17.4	26	0.0	30	0.0	20	96.5	58
3	15	20.0	20	17.6	25	0.0	20	155.6	58
	16	0.0	20	0.0	22	0.0	20	160.3	56
	17	0.0	19	0.0	18	0.0	20	325.0	54
	18	0.0	17	0.0	17	0.0	20	246.8	47
	19	0.0	15	0.0	14	0.0	18	139.9	47
Zone	20	0.0	12	0.0	12	0.0	18	64.8	45
4	21	0.0	11	0.0	11	0.0	18	77.7	42
	22	0.0	6	0.0	11	0.0	18	41.8	40
	23	0.0	2	0.0	11	0.0	18	56.7	36
	24	0.0	2	89.2	10	0.0	18	13.6	34
	25	0.0	1	0.0	8	0.0	18	43.6	28
	26	X*	X	53.0	8	0.0	18	16.3	27
Zone	27	X	X	0.0	7	0.0	18	0.0	23
5	28	X	X	0.0	2	0.0	10	41.1	13
	29	X	X	0.0	1	0.0	1	139.3	7
	30	X	X	X	X	0.0	X	285.8	5

Note: X indicates no tape in position interval and thus no error rate estimation.

Table I. Unsmoothed Position Error Rates (Errors/Metre x 10^{-4})

From the analysis now described, two different time-varying probability distributions showing the overall aging behavior can be obtained, one for the entire tape and the second excluding zone 1, the first 100 metres. Both time distributions can be fitted very precisely to the function illustrated in Figure 2 and 3.

If the "life" of the tape is taken to be the median life (i.e., the age for which reliability is 50%), this function has convenient properties; on a log-log plot (Figure 3) the tape life A_0 falls directly below the intersection of the asymptotes, and the slope of the descending asymptote equals the exponent.

In order to determine an effective strategy for archival recording, it is necessary to choose a block length which is short enough to have a reasonable probability of avoiding errors but long enough for efficient packing of the data on the tape. This can be estimated from the error probability distribution for length; i.e., the average probability that a given interval will be free from error irrespective of its position.

Figures 4 through 7 show some length distributions for selected cases. Unfortunately, the data do not provide values for lengths less than 25 m.

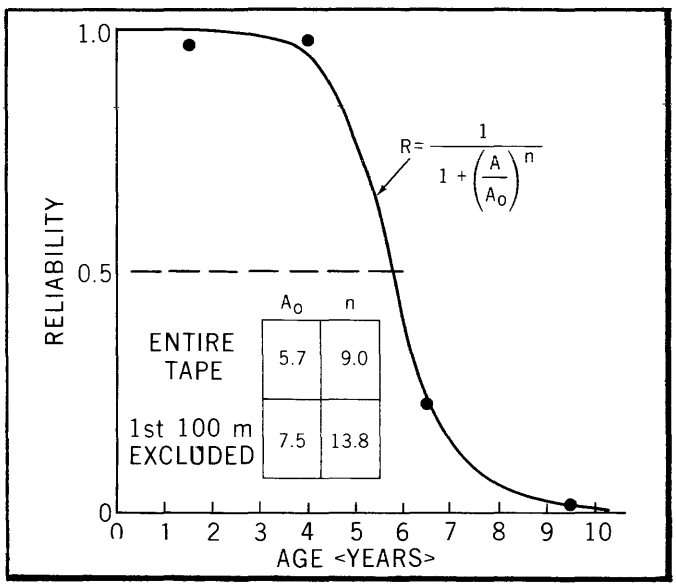


Figure 2. Linear Plot of Reliability vs Time in Storage

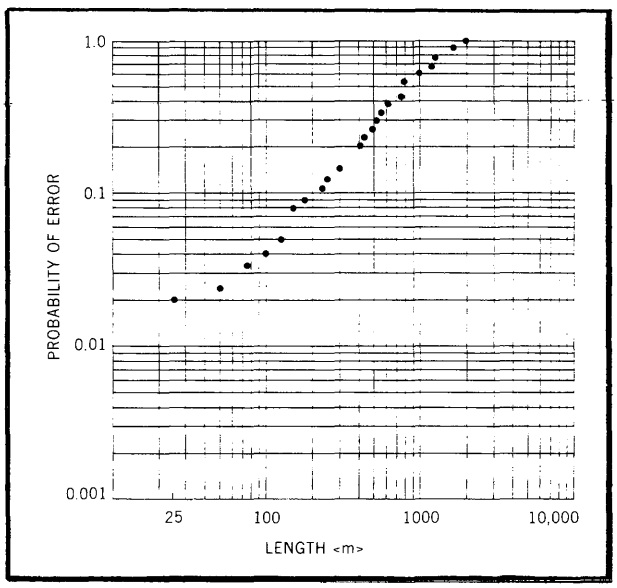


Figure 4. Probability of Error vs Length, Zone 1, Age 1 to 5 Years

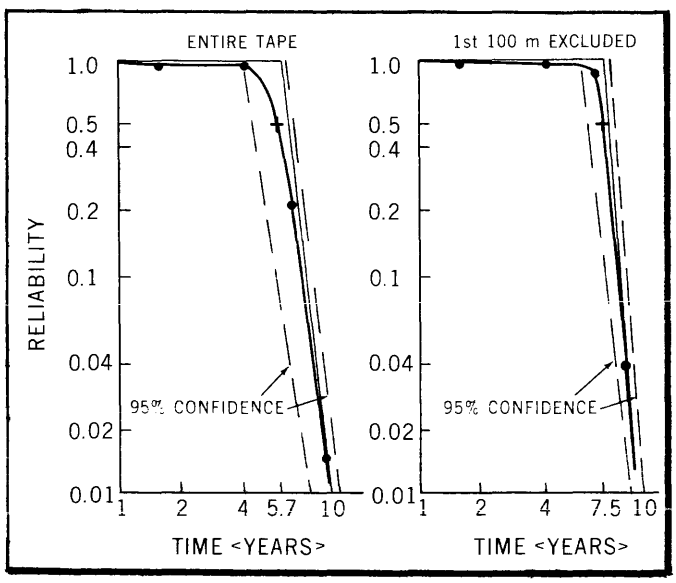


Figure 3. Logarithmic Plot of Reliability vs Time in Storage

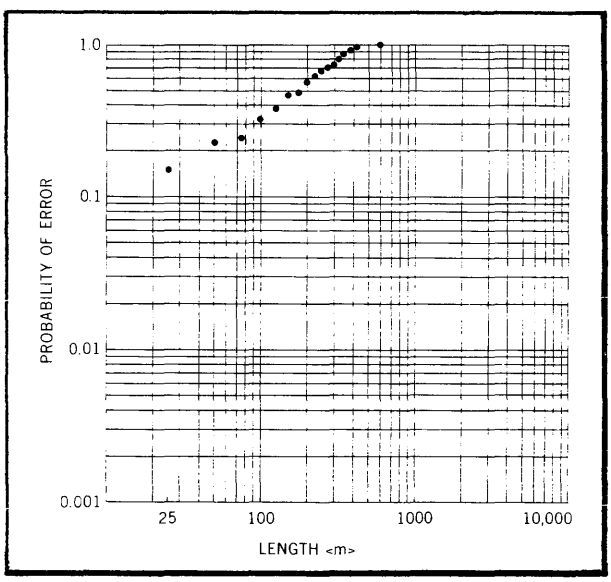


Figure 5. Probability of Error vs Length, Zone 1, Age 9 to 11 Years

Model R $= 1/[1 + A/A_0)^n]$, where
R = probability that a tape has less than 1 error/300 m
A = time in storage
A_0 = median tape life (time when R = 0.5)
n = exponent

A. Total Tape Data

Estimated parameters: $A_0 = 5.7$ years, n = 9.0

Average Time in Storage (years)	95% Confidence	R_{obs}	R_{fit}	$R_o - R_e$
1.5	0.9987 0.6002	0.967	1.0	0.0327
4.0	0.9919 0.9281	0.975	0.9600	0.0149
6.5	0.5898 0.0168	0.2286	0.2347	0.0061
9.5	0.0315 0.0005	0.0145	0.0100	0.0045

Analysis of Variance:		
	Regression mean square	2.310
	Error mean square	0.0013
	F ratio	1777*
	Correlation	99.97%

B. Tape Data with First 100 m Excluded

Estimated parameters: $A_0 = 7.5$ years, n = 13.8

Average Time in Storage (years)	95% Confidence	R_{obs}	R_{fit}	$R_o - R_e$
1.5	- -	1.000	1.000	0.0
4.0	- -	1.000	1.000	0.0
6.5	0.9862 0.3670	0.879	0.878	0.001
9.5	0.0757 0.0160	0.039	0.037	0.002

Analysis of Variance:		
	Regression mean square	3.8
	Error mean square	5×10^{-6}
	F ratio	7.6×10^5 *
	Correlation	virtually 100%

*Well over 99% significance level

Table II. Data for Time Reliability Function

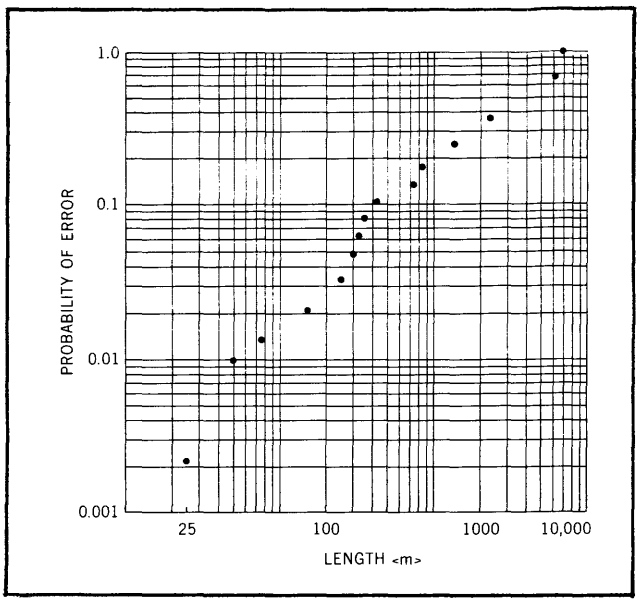


Figure 6. Probability of Error vs. Length, Zone 2, Age 1 to 8 Years

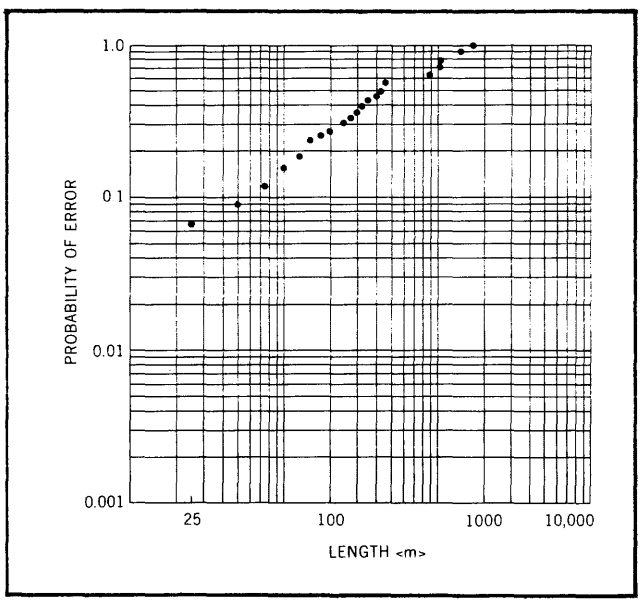


Figure 7. Probability of Error vs. Length, Zone 2, Age 9 to 11 Years

RESULTS

The data discussed above suggest the following major conclusions about the archival tapes at GSFC:

- *Performance.* The performance of tape wound on a standard 10.5 in. reel depends significantly on position in the reel. Five distinct regions are identifiable by performance, the odd-numbered regions being characterized by relatively poor performance compared to the even-numbered regions. The error mechanisms associated with regions 1, 3, and 5 are not yet identified, but there is evidence that they are mechanical rather than magnetic in nature.

- *Error Mechanisms.* Tapes used in the GSFC computer facilities and placed in the archive have been subjected to a controlled environment typical of "laboratory" conditions. Temperature, humidity, and ambient level of dust and other contamination during use are considered not atypical of those found in most computing facilities and are often referred to as "semi-clean" room conditions (there was some departure from temperature and humidity control for some tapes placed in the archive prior to 1965). The identifiable error mechanisms occurring in tapes retrieved from this archive may be placed in the following categories: tape flaws exhibited by the tape itself and attributable to the manufacture or age-dependent degradation; trapped debris, arising fom the tape itself or acquired from the environment; abuse resulting from improper handling; cinching. The relative frequencies of failures in various categories are listed in Table III.

	Normal Reels	Defective Reels
Mechanical Damage (including cinching)	53%	45%
Dimples	23%	20%
Oxide Defects	10%	25%
No visible defect	14%	10%
	100%	100%
Reels containing errors	40%	7%

Table III. Error Mechanisms - Average for all Ages

Note: "Normal" reels have 50 or fewer parity errors per reel; "defective" reels have more than 50 parity errors per reel.

- *Median Life.* The median life of tapes in the archive, based on an arbitrary performance criterion of 0.003 blocks in error per metre derived from operating practice, shows that the tape performance maintains a substantially constant and acceptable level for the first four years of its life and then fails to meet our criterion at an age greater than about 5.7 years. This life can be extended about 30% if recording is excluded from the first 100 meters of the tape; median life is then about 7.5 years.
- *Technology.* An unknown fraction of the tapes entered in the GSFC archive were subjected to environmental stress (temperature and humidity) prior to 1965. The tapes in this group show a reasonably typical error profile, but are quantitatively inferior to those entered more recently. It is not clear from our limited data whether the poor performance of the group of oldest tapes results from excessive environmental stress, inferior tape technology, a continuing decaying process dependent on age, or a combination of these factors. It may be noteworthy that cinching can be induced rapidly by temperature and humidity stress. However, since the overall error pattern existing in younger tapes appears to be accentuated in the oldest group, we believe a continuing decay process offers the more probable explanation for the observed data.

RECOMMENDATIONS

The following items would appear to be good practice in preparing a tape archive based on the results obtained:

- The data of Figure 1 suggest that archival performance can be improved by avoiding regions 1, 3, and 5 of the tape. If region 1 is omitted the median life will be extended by 1.8 years.
- If all of the tape must be used, a substantial improvement in performance can be obtained if cinching can be avoided. Means often recommended for alleviating cinching are storage in a temperature-controlled environment, programmed tension wind, or rewinding a tape at appropriate intervals. The data suggest a 4- or 5-year interval may be sufficient for this purpose. When data must be retained reliably for longer than the periods indicated in Figures 2 and 3, special provision should be made to condition the tape or transcribe it.

It would be highly desirable to obtain data from other facilities in a fashion which would permit comparison of results. The following parameters would be suitable for this purpose:

Recording density	556 cpi, 7-track
Record current	saturation per ANSI Standard
Read threshold	20% of normal playback amplitude
Error detection	count blocks in error

ACKNOWLEDGEMENTS

The authors wish especially to acknowledge the help of Dr. William E. Wells of Wolf Research and Development Corp. for his advice and assistance in working out the statistical parts of the analysis, and to Mr. Robert T. McKenna of GSFC and A. Fentriss of Computing and Software, Inc., for their assistance in collecting and processing the data. Thanks are also due to R. G. Holmes of GSFC and others who have contributed to this study in the past.

REFERENCES

1. Wallodi Weibull, *A Statistical Distribution Function of Wide Applicability,* J. Appl. Mechanics, **18**, No. 3, 1951 Sep, 292.
2. D. R. Cox and P. A. W. Lewis, *The Statistical Analysis of Series of Events,* Methuen & Co., Ltd., London, UK, 1968, pp. 182.
3. John E. Freund, *Mathematical Statistics,* Prentice-Hall Inc., Englewood Cliffs, NJ, 1962, pp. 169.
4. T. Jones and G. Prine, Final Report: Analysis of Archival Digital Magnetic Tapes; Contract NAS5-11736-Mod 178. 1972 Dec, Wolf Research and Development Corporation. Prepared for NASA, Goddard Space Flight Center, Greenbelt, MD.
5. Source Document for Archival Tape Study. Contract NAS5-11736-Mods 178, 182, 1976, PCM 520-W-69728. Wolf Research and Development Corporation. Prepared for NASA, Goddard Space Flight Center, Greenbelt, MD.

COMBINING REMOTE & CENTRALIZED DATA OPERATIONS ECONOMICALLY

by ROBERT S. HULSE

Hewlett-Packard Company
San Diego, CA

INTRODUCTION

For many different reasons (obtaining increased sales, providing better service to customers, or achieving more timely and wider distribution of products), companies have seen fit to geographically locate and decentralize many of their operations - sales offices, manufacturing or service operations, storage and distribution facilities. Although decentralization is a must for the continued success and future growth of such companies, there are still some functions, such as payroll processing, which are data dependent and which are performed more efficiently and less costly in a centralized operation, mainly because to do otherwise would involve the performing of numerous duplicative functions at each decentralized facility.

As one example of combining remote and centralized data operations economically, this paper describes a "combined" payroll operation which economically links the data collection function performed at remote decentralized locations with established data processing methods of the central facility, thereby obviating the need for performing duplicative data processing functions at the remote outlying locations. The method of operation described herein is equally applicable to inventory control applications, or other record-keeping business transactions that involve remote collection and centralized processing of data.

NEED FOR AN ECONOMICAL "COMBINED REMOTE/CENTRALIZED" OPERATION

By centralizing its payroll processing operation, a company would be able to achieve the following measurable benefits over having numerous decentralized payroll processing operations:

(1) Development Dollar Savings

A significant savings in "program development dollars" could be achieved if existing payroll programs at the central facility are used by the remote facilities instead of these facilities developing their own payroll-processing programs, and their own summary and statistical reporting programs, such as for vacation and tax reports.

(2) Data Storage Savings

Dollar savings on data storage and data maintenance charges could be realized by the remote facilities if data from these facilities are stored and maintained at the central facility instead of at each remote facility. This eliminates the need to maintain additional storage equipment and duplicative data files at the remote facilities.

(3) Data Processing Savings

Data processing dollars would be saved if payroll processing, with its attendant file accesses and report generating, is performed as a single, complete centralized operation than as independent, disjunctive operations performed at the remote facilities as independent program "runs".

Despite these obvious savings, and the opportunity cost that would also be saved if the resources used for individual program development, data storage and data processing at the remote facilities were used instead for other needed projects, there are instances where the turnaround time associated with centralized processing would be prohibitive, especially where the remote facility does not have a data communications capability enabling prompt communication of the payroll information to the centralized facility for processing. For example, because of the short time between the time of data entry and the time of receipt of paychecks, the processing of *weekly* payroll at the central facility might be unworkable. Also, the cost of having a computer at the remote site to communicate with a central computer may be prohibitive in light of insufficient volume of work at the remote site to justify its own computer.

To achieve the cost savings associated with centralized processing, therefore, an economical combined remote/centralized operation is needed that provides prompt turnaround times and has a low cost data communications capability.

ONE ANSWER: AN ECONOMICAL STAND-ALONE UNATTENDED REMOTE DATA STATION

An economical stand-alone data station (e.g., a communications card reader for approximately $3000), such as shown in Figure 1, would permit economical communication of payroll, inventory, or other data to a centralized facility without the need for a computer at every remote location.

Such a remote, stand-alone card reader would operate unattended, and would be capable of:

(1) Being polled by the central computer, at the computer's convenience.

This provides several advantages: First, by being polled at the central computer's convenience, the need for developing realtime interrupt subroutines, and increased line handling and data storage capabilities for the central computer is made unnessary. Also, the need to establish scheduled periods for data communications between the remote facility and the central computer would be obviated. Furthermore, by permitting the central computer to poll the remote facilities at its convenience (i.e., ask for data from the remote facilities only when it is ready to do so), this gives the central computer greater flexibility in scheduling and performing its other operations. Collaterally, such flexibility also enables new functions to be added to the central computer that otherwise could not be scheduled in.

(2) Transmitting data by telephone at high or low speeds.

This permits data to be transmitted at selected higher speeds of 120 or 240 characters per second over regular "dial-up" facilities or over a company's existing tie-lines. The data speed would be switch-selectable and would also include lower speeds of 110, 150, 300, 600 or 1050 baud, conforming to the capabilities of the central computer and its multiplexer.

(3) Retransmitting one or more cards of information upon command from the central computer.

This permits retransmission of data from the remote facility in the event data received at the central computer is erroneous. The retransmission request would be honored even if the central facility had to break telephone contact temporarily with the remote facility in order to service a realtime request for service from another source. The retransmission would occur when telephone contact is resumed.

(4) Rejecting into a select hopper any card improperly filled out.

This provides the central computer the opportunity to examine the contents of each card, to request retransmission in the case of transmission or data error, and to reject the card in the event the error is not corrected. The card, if erroneous, can then be returned to the submitter for correction, and later re-entered.

(5) Accepting marked, preprinted or punched cards.

By having the capability to accept marked, preprinted or punched cards, card preparation is simplified. Nonchanging information, such as employee's name and number, can be preprinted or prepunched, while the changing information can be "pencil-marked". In the case of error, the marked information can be easily erased and remarked.

(6) Compacting the data so as to minimize transmission costs.

By having the capability to indicate precisely where data ends on a card, unnecessary spaces or other information on the remainder of the card need not be transmitted. This reduces the amount of data transmitted to only "essential data", thereby reducing the time and cost of data transmission.

(7) Operating in series with other card readers, if necessary.

This permits several card reader stations at the remote facility to use a single telephone line and modem in communicating with the central facility. This "multi-drop" method of operation not only saves transmission cost by making the need for duplicate telephone facilities unnecessary, but also provides for growth in the facility's operations by being able to handle both increased data volumes and differences in data function (e.g., payroll data collection, inventory data collection, etc.) performed at the different card reader stations.

A SUCCESSFUL REMOTE/CENTRALIZED PAYROLL OPERATION

An efficient, economical payroll operation is now described, featuring the use of a H-P Model 7260A Card Reader as an unattended remote data collection station linking the Finance Department of the H-P San Diego Division to the central data processing system located at Corporate Headquarters in Palo Alto, CA. The remote reader is linked by regular telephone and two 202C modems to the central computer facility (Figure 1).

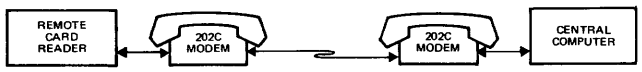


Figure 1. A Stand-alone Data Station

Payroll data is punched on 80-column cards, or optionally marked on 40 or 80-column cards (Figures 2 and 3). The carriage return character punched or marked on the card terminates the data on the card, thereby "compacting" the data and preventing unnecessary spaces or other information on the rest of the card from being transmitted.

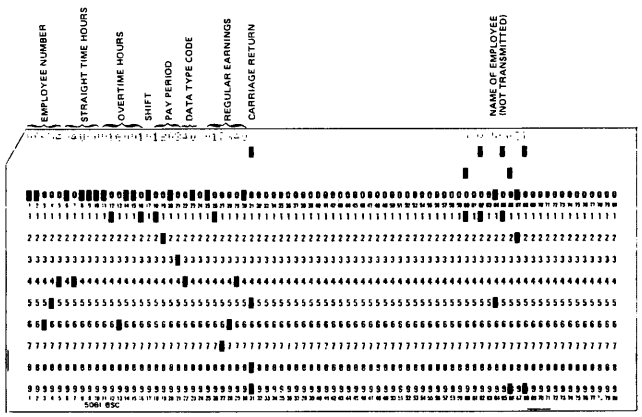


Figure 2. Punched Data on 80-Column Card

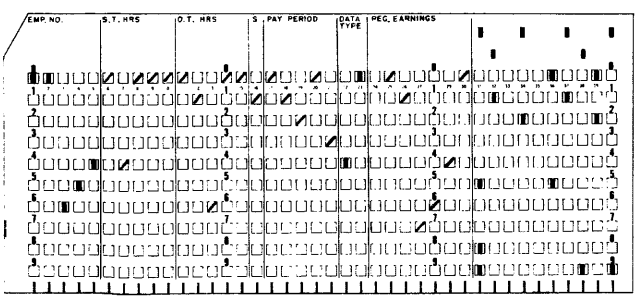


Figure 3. Marked and Punched Data on 40-Column Card

After preparation, the cards are placed in the input hopper of the card reader, and the "Line On" (Power "On") button is depressed. From this point on, the reader is able to operate unattended, and to communicate with the distant centralized computer whenever that computer is ready to receive data.

At its convenience (i.e., at scheduled periods or times when regular workload has decreased sufficiently), the central computer polls the remote reader and "commands" it to transmit the information contained on the cards in its input hopper, a card at a time (Demand Mode). The computer checks to see if data was in fact sent, and if so, whether it was sent correctly. If not, it requests retransmission. If after a reasonable number of attempts the data received is still incorrect, the card containing the erroneous data is rejected; transmission continues until the input hopper is empty or the output hopper is full, at which time communication is ended. If status information was sent instead of data, indicating a reader, telephone or modem malfunction, the computer terminates transmission and turns off the reader until the problem is corrected. (In periodically attended operations, the computer may even ring a bell in the reader to signal that a problem has occurred there. The person in attendance would then manually correct the problem.)

If data was being received successfully at the time the input hopper was found to be empty, or the output hopper full, it is stored on magnetic tape for later processing by the payroll program. As indicated in Figures 4 and 5, the application program controlling the data transfer from the card reader may be any simple control program. It may be as flexible as desired, tailored to individual applications such as payroll, inventory control or others, and/or to different modes of use of the remote reader - e.g., as an attended or unattended device, or as a single or multi-drop device, with or without a remote printer terminal.

```
      DIMENSION IBUFF(80)                    }
      NPICK=0                                }
      NTRAN=0                                }
      NREJ=0                                 }  Prepare to poll Remote Card Reader Station
      ISTAT=0                                }  (reserve storage for Input Data, initialize
      LEN=80                                 }  Counters and define Control Words)
      IASCI=100128                           }
      IRET=20012B                            }
      IREJ=30012B                            }
      IOFF=40012B                            }
      IREAD=100128                           }
      CALL CWRIT(IASCI,ISTAT)                   Select ASCII Code
      CALL CREAD(IREAD,ISTAT,IBUFF,LEN)         Request a Card of Data or Status Information
      IF(ISTAT-140000B)150,200,150              Test for Card Reader Malfunction
      IF(ISTAT-120000B)160,200,160              Test for Telephone or Modem Malfunction
      IF(ISTAT-110000B)170,250,170              Test Input or Output Hopper Status
      IF(IBUFF(22)-32000B)240,180,240        }  Check Data
      IF(IBUFF(23)-30000B)200,190,200        }
      NTRAN=0                                   Reset Retransmission Counter
      WRITE(15,400)IBUFF                        Store Data on Magnetic Tape
      NPICK=NPICK+1                             Count Number of Cards Received
      GO TO 130                                 Prepare to Read Another Card
      IF(NPICK-0)240,200,240                    Check for Valid End of Data
      ENDFILE 15                                Write End-of-File on Magnetic Tape
      STOP 0                                    Successful Stop
      CALL CWRIT(IOFF,ISTAT)                    Turn off Card Reader
      STOP 7777                                 Error Stop
      NTRAN=NTRAN+1                             Count Number of Retransmissions
      IF(NTRAN-3)300,300,320                    Check if 3 Retransmissions Made
      CALL CWRIT(IRET,ISTAT,IBUFF,LEN)          Request Retransmission
      GO TO 140                                 Prepare to Check the Retransmission
      CALL CWRIT(IREJ,ISTAT)                    Reject a Card
      IF(ISTAT-104000B)340,370,340              Check if Reject Successful
      NREJ=NREJ+1                               Count Number of Reject Attempts
      IF(NREJ-3)360,360,260                     Check if 3 Reject Attempts Made
      GO TO 320                                 Prepare to Reissue the Reject Command
      NREJ=0                                    Reset Reject Counter
      NTRAN=0                                   Reset Transmission Counter
      GO TO 130                                 Prepare to Read Another Card
      FORMAT(80A1)
      END
```

Figure 4. Simple Application Program (in FORTRAN)

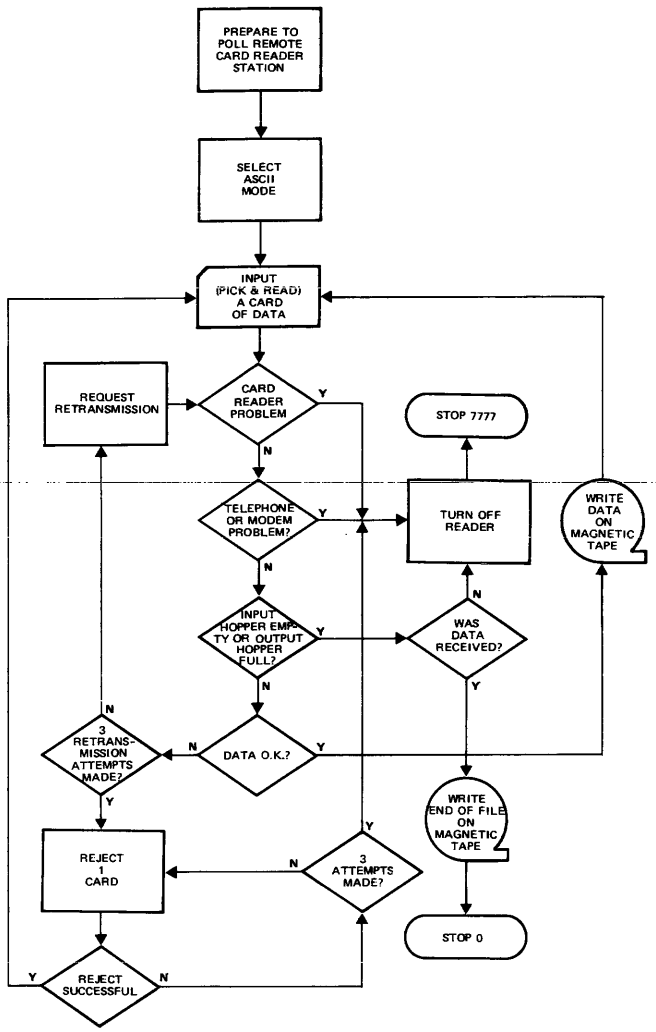


Figure 5. Simple Flowchart of Application Program

TANGIBLE ECONOMIC BENEFITS DERIVED FROM THE REMOTE/ CENTRALIZED OPERATION

Figure 6 shows the actual program development dollars saved when our remote facility decided to perform the data collection and data transmission payroll functions utilizing the data processing capabilities of the central facility. The development dollars shown represent direct labor cost, assuming an average programmer's wage of $1175 a month.

For some, the development dollars saved may be lower, but for many others, it may be significantly higher, depending upon the size (number of employees) of the remote facility and the extensiveness of the overall company operation.

In many instances the difference between the cost of adding disk capability to a "bare-bones" mini system at the remote facility and the cost incurred in making use of abundant disk capabilities at the central facility via remote reader is very significant. For example, the cost of implementing needed auxiliary storage at the remote facility could very well be about $358 per

PROGRAMS	MAN-MONTHS	DEVELOPMENT DOLLARS SAVED
Payroll	11.5	13,512
Payroll Tax Report	1.4	1,645
W2 and 941A FICA Reports	1	1,175
Bureau of Labor Statistics Report	1.6	1,880
Insurance Report	2.2	2,585
Personnel Wage & Salary Report	2.4	2,820
Workmen's Compensation Report	2	2,350
Vacation & Sick Leave Report	3.5	4,112
TOTAL	25.6	30,079

Figure 6. Program Development Dollars Saved

month. ($350 per drive per month on a 1-year lease, plus $8 per month rental for one disk pack.) The cost of using existing disk capabilities at the central facility could simply be the cost of the packs ($8.00 per month). This would represent a monthly disk storage saving of $350, or about $4,000 per year.

In addition to development and storage dollars that are saved, processing dollars (i.e., program execution or computer run dollars) are also saved in a remote/centralized operation. It is estimated that for each out-of-pocket dollar spent either for an independent system at the remote facility or for service from a Service Bureau near the remote facility, the actual "inhouse" incremental cost to the company for the remote facility to use the system *existing* at the central facility is approximately 60-70% less than the cost to the company if the remote facility were to purchase its own system or use an "outside" processing service. This savings in processing cost, of course, varies with the number of users of the central system, the extent of each user's usage, and the extent to which the cental computer is depreciated.

Compared to the cost of utilizing this "buffered remote reader method" for combining remote and centralized operations, the savings that accrue, as described previously, could be significantly greater than the cost associated with this method. Figure 7 shows the approximate monthly cost of utilizing the buffered remote reader method. The figures reflect our experience in using the Model 7260 buffered mark/punch card reader as an unattended remote data collection/data communications device.

With respect to power consumption cost, the reader uses 72 watts when in stand-by mode waiting to be polled, and 135 watts when the motor is "on" and the reader is actually communicating with the distant computer. It is in unattended operation, ready to communicate, approximately 15 hours per day (i.e., from 5:00 p.m. to 8:00 a.m.) for 20 work-days per month. Utilizing a 202C modem for 120 characters per second transmission, data transmission is achieved at the rate of 100 cards per minute (average of 72 characters on a card) or 300 cards within a single 3-minute long distance call limit. Polling occurs once a day.

The monthly cost of operation is determined as:

72 watts x 15 hours x 20 work days per month =
21.6 kilowatt-hrs/month.
135 watts x 1/20 hour x 20 work days per month =
0.135 kilowatt-hrs/month.

Assuming a rate of $2.67 for the first ten kilowatt-hrs used, and 6.006 cents for each additional kilowatt-hr used, the monthly power consumption cost is determined as:

10	kWh	$2.67
11.735	kWh at 6.006 cents/kWh	0.71
21.735	kWh	$3.38

ITEMS	BASIS OF CHARGES	MONTHLY COST
Power Consumption	Approx. 21.7 kWh/month	$ 3.38
Telephone Traffic	Approx. 20 calls at $1 each	20.00
DataSet/Modem	Approx. $45 rental/month	45.00
Remote Reader	Approx. $3000, depreciated over 4 years.	62.50
TOTAL MONTHLY COST (approx.)		$130.88

Figure 7. Approximate Monthly Cost of Operation of Remote Reader

For higher daily volumes of data transmission (i.e., >24,000 characters per day, in a scheduled polling operation), little additional cost is incurred other than the incremental telephone time cost which, though minimal at the 1200 baud transmission rate (7200 characters per minute), would be even less with a 2400 baud modem with superior turnaround capabilities.

The dollar benefit that accrues with a remote reader operation is even more significant when multiplied by the number of remote facilities that can benefit from this method of operation.

OTHER ECONOMICAL METHODS

The payroll application described previously demonstrated the use of the remote reader as an unattended device operating in a "Demand Feed Mode" (card-at-a-time mode). However, other applications may find one of these methods preferable:

(1) Multi-Drop Operation

A multi-drop operation, such as shown in Figure 8, may be performed as an attended or unattended operation. By means of "roll-call polling" via a single telephone line, the central computer, in a single call to the several remote readers, addresses each reader in turn and unloads its data. This method of operation permits different types of data (payroll, inventory, sales, parts or other) to be transmitted to the central computer as a result of a single telephone call from the central facility to the remote facility. Actually, whether the operation is unattended or attended (because of the need to replenish the input hopper in the case of high volume, i.e., greater than 500 cards at a time), the application program that receives the data at the central computer identifies the data from the data type code it receives in the data (see columns 22 and 23 of Figure 2, and lines 170 and 180 in the program of Figure 4). In this way, different application programs can process different data in different ways.

Each remote reader is unique in its multi-drop environment. It can be specifically selected by the cental computer without any communication with the other readers. Likewise, the configuration may be easily changed by the removal or adding of remote readers without affecting the operation of the other readers.

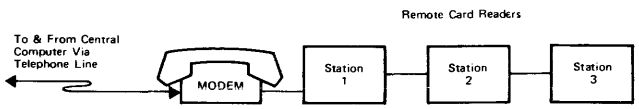


Figure 8. Multi-Drop Method of Operation

(2) Continuous Feed Mode Operation

In addition to the demand feed mode of operation, the remote reader is also capable of "batch data transfer" when operated in continuous feed mode. Once the central computer commands it to continuously send data, data is continuously transmitted by the reader until its input hopper is empty or output hopper is full or transmission is specifically interrupted by the applications program.

(3) Unit Record Method of Operation

In the event a full card of data is to be transmitted each time, the carriage-return code need not be included (marked, punched not printed) on the card. Upon sensing the end of the card, the card reader transmits a carriage-return character following the data characters on the card. (The card reader always transmits a line-feed character upon sensing the beginning of card.)

Many unit record operations lend themselves to this method of operation. Many different applications (educational, medical, retail business or utility, to name a few) use a card as an input data form and a mark/punch reader as a data input device. These and many other applications can significantly benefit from the added capability in data collection and data communications that a buffered, remote mark/punch reader provides.

SUMMARY

A method was described for economically combining the data processing requirements of many decentralized (remote) company operations with the capabilities of a central computer facility. An economical remote mark/punch buffered card reader was used as a data collection device, capable of transmitting data at various speeds and communicating via ordinary telephone lines with a distant central data processing facility. Modes of usage were described, showing cost economies in data transmission, program development, data storage, data processing, and other benefits such as improved central operations scheduling and ease and flexibility in data preparation and data entry.

A LOW-COST APPROACH TO REMOTE DATA ENTRY

by B. V. O'BRIEN

Western Union Data Services Co.
Mahwah, NJ

An efficient approach to remote data entry is assuming increasing importance for typical geographically dispersed organizations. Its importance is becoming more apparent as a result of the increasing visibility of the implications of two well-established trends in data processing.

The first of these is the centralization of computer power for purposes of achieving scale economics, data base management efficiencies, and other considerations. The second trend is the dispersal of the data preparation function out to the original data sources for purpose of achieving rapid data accessibility, as well as the greater data entry accuracy associated with source data entry.

The data system designer faced with the problem of putting a remote data entry system in place must resolve a number of questions for which facts and experience have not yet provided tools. One important question is the relative accuracy of data preparation at the source location where familiarity with the source information is available, as compared to accuracy in a centralized key entry location, where functional control is more available.

The common design approach to this dilemma is simply to overwhelm the remote data entry problem with the kind of operator guidance and error checking that can be accomplished in computerized intelligence. Until recently this has implied online data entry using either a CRT or some form of typewriter terminal. This approach has provided the essentially unlimited intelligence of the central computer to the remote data entry station. However, it also results in extremely high communications, software, and computer overhead costs.

More recently, to alleviate some of these costs, many organizations have gone from this online approach to an offline batch approach using an "intelligent" terminal. This puts the same kind of computerized intelligence level at the hands of the remote data entry operator, while significantly reducing the communications and overhead costs attendant to online operations. However, even with the significantly reduced costs of electronic logic, an intelligent terminal is still a very expensive device. In addition, the software costs, both at the terminal and at the computer center, may equal and more commonly exceed the software cost of the online approach.

We submit that with proper definition of the problem some percentage, perhaps a large percentage, of these remote data entry applications can be adequately served with a much simpler and much lower cost approach to a data entry system. Such an approach not only allows the sophisticated data entry system user to achieve a significant cost reduction but it also (probably more importantly) opens up the world of remote data entry to those many organizations and applications for whom remote data entry was apparently foreclosed by reason of cost.

Let us first examine the key characteristics of the data entry requirement of a remote location. One fairly common characteristic is the need to enter many different types of transactions. Typically no one of these transactions is handled with any significant frequency, and the total daily volume of data prepared at the remote location is typically in the 10,000 to 20,000 character range. Another common and very important characteristic is that the urgency requirement of these transactions varies widely, ranging from minutes in some cases to weeks.

Urgency has a way of falling into categories which are directly associated with functional operations. For example, operations where a customer is waiting, or where an employee is waiting and cannot proceed, generally have urgencies in the one-minute range (i.e. plus or minus a factor of 3; 20 seconds to three minutes). Operations in which an employee is waiting but can perform other functions tend to be in the one-hour range. Operations which start operations elsewhere (orders, reports, etc.) tend to be in the one-day urgency range (overnight to two days). A large percentage of the data entered at the typical remote terminal location tends to be in that next day urgency category. This obviously does not apply to those remote locations which are primarily customer response locations, such as phone bureaus or reservation desks.

Given an application situation with these characteristics (and we submit that there are many such situations in actual practice), a relatively unsophisticated and low-cost terminal is an acceptable and in fact correct system solution. The problem of input errors solved with such complexity in other situations is more than adequately controlled by the operator's inherent familiarity with the source information and a capability to sight-verify the input documents.

Since the data is prepared and stored in an offline mode, transmission speed can be chosen independently from the data preparation speed, depending on the communications network, and the remote computer center can be economically arranged for speeds anywhere between 10 and 120 characters per second.

An example of a low-cost remote data entry system is that operated by a manufacturing firm for its sales offices. Teleprocessing is the basic method for providing these services. At each sales office there is a Teletype Model 33 ASR equipped with a magnetic cassette unit. The terminal is used as a keypunch device in the data preparation mode. It is then used as a remote controlled data transmission terminal for the data collection function and is also used as a remote report printer by the central computer.

In the data preparation mode, individual format paper tape loops are prepared for each type of transaction prepared at the sales office. These include sales orders, amendments, sales summaries and accounting data, such as expense reports. For each type of transaction the operator places the applicable format tape loop in the tape reader, and using a foot switch steps the tape from one data field to the next, printing a prompting message and filling in blank data fields from the keyboard as she progresses, The resultant data is recorded in the magnetic tape cassette.

At the end of the day the tape cassette is placed in the automatic answer mode and is polled from the central computer at night, and the entire day's transactions are collected. After processing each office's daily transactions, the computer places another call to that sales office and delivers a copy of the processed sales order, various summary reports and an edit error report. The edit error report is a listing of the transaction number and the cause of error of each of the errored input transactions. The following morning the operator prints out the various reports, corrects the errored transactions, and resubmits these with the new transactions that evening.

The equipment cost at each sales office is about $150 per month for the Model 33 ASR, the magnetic tape cassette unit, and a 1200 baud data set. The average daily volume of 10,000 characters per sales office is transmitted in about two minutes, using the company's regular WATS lines. The data collection system at the computer centers is a minicomputer-based data collector and spooler obtained on a turnkey basis from a system supplier for a cost in the $2,000 per month range. The resulting system produces 360-compatible magnetic tape which is physically transferred between the data collection system and the company's batch data processing system. The data collection system required no changes in the company's basic data processing software.

This kind of a "plain vanilla" approach to remote data entry has enabled at least one company to implement a data communication system quickly, economically, and effectively.

RESEARCH PROSPECTS IN PROGRAMMABLE ASSEMBLY SYSTEMS

by DANIEL E. WHITNEY

Massachusetts Institute of Technology
Cambridge, MA

INTRODUCTION

In most industrialized nations, the large volume manufacture of discrete parts has been highly automated. Accurate machining, molding and forming allows parts to be interchangeable and thus to be assembled by relatively unskilled people. There now exists a wide variety of special-purpose machines, some extremely large, complex and expensive, for parts manufacture. These parts are assembled by people into substantially identical products. Assembly-line jobs can be quite boring, however, since the substance of the job may repeat every ten to fifteen seconds. The resulting discontent plus high labor cost is exerting pressure to automate the assembly process. Other pressures to automate come from the Occuptional Safety and Health Act, and from efforts to improve product quality and uniformity.

It might seem natural to build special-purpose machines for assembly similar to those currently in use for materials processing and, indeed, a few such machines have been built. (One machine reportedly assembles and tests 2400 automobile cigarette lighters per hour.) Several facts argue against taking this route, however:

(1) From a financial point of view, fixed automation machinery is typically very expensive and its rigidity makes market forecasting errors quite serious. Product design changes are almost impossible to make. To pay for itself the machine must make many millions of items.

(2) From a marketing point of view, there is a trend away from identical products and toward products which are specialized (on short lead time, naturally) for individual customers. While some basic parts will be common to all versions of a product, many others will be different. This leads to what is called the model mix problem.

(3) From a production point of view, the model mix problem has two main consequences. First, production volumes for each version of the product are relatively small and cannot be economically batched because the incoming order stream is random. People are adaptable to a random job stream, but only to a limited extent. Thus the second consequence is that assembly mistakes occur when the wrong part is installed.

(4) From a personnel point of view, it should be recognized that while people are assembling things they are also performing vital inspection operations, seeing that the parts have been made properly, and that previous assembly steps have been carried out correctly.

This all means that both manual assembly and fixed automated approaches have limited ability to cope with emerging production problems. The remainder of this paper classifies and briefly describes current research efforts addressed to these problems.

RESEARCH APPROACHES

It is generally assumed that future flexible assembly machines will consist of one or more mechanical arms with general purpose hands or tool grabbers, all controlled by a computer. Initially individual machines could be installed in existing assembly lines, but as time goes on it is likely that the assembly line concept will be modified, perhaps resulting in a cluster or star arrangement in which parts are fed from several directions rather than one.

Current automated assembly systems are imitations of large-volume automated machining systems and thus share their rigidity. From a control point of view, they operate on the basis of open-loop positional control. This is true even of current industrial robots, which only recently have incorporated the ability to read micro switches and branch to different parts of their programmed motion sequences. This does not, however, permit them to modify those motion sequences. Open-loop control also requires liberal use of expensive jigs and fixtures.

Most assembly research approaches are based on closed-loop organization, in which computers are vital components. A basic assumption in closed-loop control is that sensory information in various modes is being monitored more or less continuously, with the intent of modifying the fine structure of the motion, rather than simply switching programs at isolated times.

Fine-structure modifiability and the presence of computers as components also allow gross-structure modifiability in the sense that the machine can be reprogrammed to perform a different assembly task. Better, a machine loaded with several such programs can switch on short notice from one product to another.

Within the basic premises of reprogrammability and closed-loop control, several research threads seem to be emerging. One is traditional artificial intelligence robotics, with its emphasis on visual systems and scene analysis. Much of this work was originally directed toward interaction with unfamiliar and unstructured environments, such as the surface of Mars. The scene analysis skills developed seem most applicable to the inspection phase of assembly. They would also be applicable to gross positioning if it is assumed that parts would be fed to a machine in random orientations. This is not a likely occurrence, however. Up to now the artificial intelligence laboratories have relied primarily on general-purpose digital computation for sensory analysis and robot arm control. The MIT and Stanford Groups are typical of this general approach.

A growing trend, requiring more expertise in hardware and less in software, is to identify in detail the various sensory and control activities needed for closed-loop programmable assembly and to specify which could be realized in hardware or special parallel computing elements, and which are best left to general-purpose serial computers. The functions deemed necessary depend heavily on the assumption as to the machine's environment.

One can assume that parts are fed in only roughly the correct orientation and that the machine itself must orient them the last few degrees. If so, then either the machine's hand must have the dexterity and sensory capacity to do this (aided by cooperative features applied to the parts for just this purpose), or else the assembly process must be itself aided at crucial points by chutes, jigs, channels, clever use of gravity, and so on.

Either of these approaches could be costly and could bar easy changeover of the machine to a different product. Neither approach, however, would have too much use for vision, which is not of much value during the process of fitting parts together.

Realization that much information is available in the form of forces and moments during the actual process of fitting together has led to another approach, that of depending primarily on force feedback to perform assembly. This approach, depending on the parts being grasped in correct orientation within a very few degrees, determines by means of contact information whatever small reorientations are needed to effect assembly. Small manufacturing differences can be accommodated as well. Since the machine modifies the fine structure of its motions in response to felt forces, it will not jam if presented with grossly malshaped parts. Groups pursuing [illegible] approaches include the Stanford Research Laboratory, Japanese university and industry groups, and at least one industrial concern in the US.

The work at Stanford Research Institute[1] has concentrated on logical feedback of discrete events like touch contact. This is most useful in rendezvous between a machine's hand and an object to be grasped. This has resulted in development of several small touch sensor devices and a computer control language for building branching routines based on sensor returns.

The joint MIT Mechanical Engineering-Draper Laboratory work[2] has emphasized continuous analog signals arising from contact between one grasped object and another. Touch sensors are of little use for this because the object does not move relative to the hand during contact unless it has been poorly grasped. The interesting information consists of continuous forces and torques and may be used to guide parts together, even when tolerances are so close that prepositioning is impractical and vision cannot detect the small motions necessary. Much of the motion modification logic can be realized in hardware, making possible the high bandwidth that will no doubt be necessary if such machines are to operate at high speed.

Future developments will likely see a combination of the touch sensor approach and the force sensing approach. Vision is more likely to be useful in inspection, but less in assembly itself. Issues remaining to be resolved include the economics of providing special jigs for each product compared with providing general sensors capable of guiding the parts together. A similar trade must be made between sensors (or computers) and product design modifications to ease assembly. Finally, the economics of using flexible automation must be studied in detail. The major issues are the cost of labor and the cost of capital. The value of such a machine must be figured on its reliability and the uniformity of its product, plus the value of its flexibility as reflected in lower in-process and finished inventories and quicker response to customer orders.

REFERENCES

1. J.W. Hill and A.J. Sword, *Studies to design and develop improved remote manipulator systems,* Final report on NASA contract NAS2-6680, Stanford Research Institute, 1972 Nov.
2. J.L. Nevins and D.E. Whitney, *The force-vector assembler concept,* to be presented at the 1st Int Conf on Robotics and Manipulators, Udine, Italy, 1973 Sep.

FACTORY AUTOMATION AND DATA COLLECTION

by DAVID A. ENTREKIN

Entrekin Computers; A Division of Cutler-Hammer, Inc.
Fenton, MI

Computer controls have been used in certain process industries for some time. Now computers are being used to control piece-part production machines and automation systems. Employed as reporters and/or analyzers of a machine's operations, they give managers current data for immediate study.

Automatic machines require a carefully planned sequence of actions to integrate the movement of parts, action of clamps, and advance of tools. Inhibitions are included to prevent such things as a tool advancing unless a part is in place and clamped. An automatic control system must be capable of registering each action and supplying signals to initiate the next action or event.

Because certain actions could cause great demage if they occur out of sequence or without other conditions of readiness, the controls automatically interrelate machine events which have occurred and those which are to occur. Stoppages or alternative actions will intercede if something does not happen as planned.

MACHINE CONTROL

When the computer controls, the primary data outputs are mostly decisions resulting in machine actions, such as starting motors, energizing valves, or the movement of parts. Compared to a data processing computer, machine control computers and related equipment are smaller in size and fewer in number. They have a small processor that needs few peripherals except the automatic machine itself This is typically a Teletype terminal with tape reader/writer, to enter programs and changes into the computer, and simple reports such as production efficiency reports and diagnostic messages to be printed out.

An auxiliary memory unit is required for either system, but size, capacity, and cost are very different. Data processing installations usually have several large magnetic tape drives and disk drive units for keeping data ready for high-speed, random access. In contrast, just one disk drive with far less capacity will serve most machine control needs. Typically, one disk unit (usually integrated with the computer in its main housing), will be sufficient to store inactive programs, working information, and permanent activity records such as pieces produced. This disk unit is usually integrated with the computer in its main housing.

OPTIONAL DEVICES

Optional devices such as cathode ray tube displays or graphic plotters may be used as auxiliary output. Peripheral hardware is not emphasized in machine control because the machine sensors serve as primary data inputs, and the primary outputs are valves, motors, and similar components.

ENVIRONMENT

Environment adaptation is another major requirement for a machine control computer. It must reside on a shop floor and tolerate any environment condition that workmen themselves do. It must adapt to shop electrical conditions, such as AC current subject to unpredictable transient surges. Such "noise" must not be mistaken for machine signals. It should output signals as 110 volts AC - directly usable by motor starters and solenoid valves. Otherwise amplifiers or transformers are required, with attendant costs and problem susceptibility.

INTERFACE

Machine control computers work through an interface that links the computer to the machine, and must communicate via standard devices. Their signals and operating modes are far out of phase and character with operating characteristics of a high-speed computer. Bridging this gap requires an interpreter or interface, to receive and logically interrelate signals from the machine's many sensing devices. Interface circuits "save" the electrical logic results until the computer is ready to act. Then the computer output is converted by the interface circuits into appropriate electrical instructions for the production machine's motors and valves - and the reporting Teletype. The interface hardware is "hard-wired", and functions almost as a part of the computer, which has 100% control of it. In recent designs, interface hardware is all solid state; power differences are taken care of; electrical noise is prevented from being misinterpreted as data codes by a constantly-performed data validation routine.

SOFTWARE

Software, in addition to programs for ordinary machine action, must also provide for conditional reactions when something does not happen exactly as it should or when supposed to. Machine programs should be easily changeable, as complex automation systems can be improved or made more efficient with experience.

Poor software causes the computer to go through time-consuming and unnecessary steps. More importantly, software should anticipate all contingencies and provide appropriate adjustments or adaptive reactions. Otherwise, machine shutdown may be the software's only available answer to occurrence of a nonstandard condition.

A machine control computer is real-time-oriented. It must note conditions, make calculations, and feed back responses all within a few microseconds of when a condition occurs.

The software to support these control functions has been deskilled to exactly duplicate relay design logic, which means that (from an engineering standpoint) today's existing control engineers can be instantly converted to applying this new technology.

STANDARD PROGRAMS

A set of machine control parameters and conventions applicable to virtually all automation arrangements can be established. A "library" of subroutines to embody these parameters and conventions allows them to be handled by most computers. The result is a program highly efficient in use of machine time and memory; individual programs can be developed quickly and easily changed. Perhaps the most important thing is how easily the user's people can gain the facility to handle their own program changes without an in-depth knowledge of computers (in this concept, the master file of subroutines is contained on a tape and given to the user along with his systems).

MACHINE MONITORING

While some functions will require the computer to perform complex calculations, the operator will not even be concerned or aware of this. Only simple outputs will result, and ease the operator's job rather than duplicate it. When the computer is applied to an automatic machine it may function as a monitor of this entire scheme of operations, or it may both monitor and control. In either case, the general system concept remains unchanged. As a monitor alone, the computer records and analyzes the machine actions. It makes exceptional diagnostic reports on machine troubles and silently records management data.

When the operation of the machine varies from that defined as normal (as determined by the machine design parameters), the control system sends an indication to the data collector, which will perform a diagnostic report indicating the station number, function involved, and device number, if applicable. This provides an immediate indication of the location and device that is causing the abnormal condition.

DIAGNOSTIC REPORTS

In developing a monitoring system there is a variable quantity of information that can be made available - information which can improve downtime, reduce inventory, improve product flow and scheduling. Improved downtime alone can provide large cost reductions and justify the system installation.

When trouble occurs, the computer can check all sensing inputs to pinpoint the problem area for the operator. In many cases it can summon help from its memory of pre-established service routines to find advice about the problem. Since all machine actions can be constantly timed and compared to standards, the causes of efficiency loss can be spotted immediately.

Diagnostic and Machine Log Reports, created while the machines are running, are primarily used for trouble-shooting.

When machine operation varies from that defined as normal, the computer will print a diagnostic message indicating station number, function involved, and device number. This printout will be done as soon as the malfunction occurs.

Diagnostic reports can reduce downtime by pinpointing the specific areas of failure. Four different diagnostics for each input provide the information needed to reduce downtime and indicate possible machine failure. They determine if a specific limit switch, pushbutton, temperature switch, or pressure switch has not released, not tripped, shows a ground, or is out of its allowable time standard. These four messages basically cover all the possible trouble conditions. The first two, not released or not tripped, pinpoint the immediate cause of a machine being inoperative. Time diagnostics indicate machine degeneration, and potential downtime conditions can be detected before they occur.

CYCLE TIMING

Since timing is such an inherent part of computer operation, it is very simple to establish in a computer control program a cycle time base for all process control.

Our proprietary systems concept of cycle monitoring means that each motion will be monitored in each of its movement planes in every cycle of the control.

An example: A single cylinder slide has two motions, forward and back. We would establish a time standard for the forward motion, for example, say, six (6) seconds, and a time standard for the return motion, of say, eight tenths (0.8) seconds. Now, during the running operation, suppose the forward slide took 5.9 seconds in its forward motion. Since this does not represent a slow-up, we would accept this as being within the normal time standard. Suppose, however, on the next time cycle this device took 6.2 seconds? We would then record in our data base memory that an overtime occurred of a two tenths (0.2) second duration. The number of occurrences and the amount of overtime would be accumulated for recording in a shift log or for reporting on a discrete data request log initiated when an operator might request it later. We have found through experience that about 40 to 50% of preventive maintenance procedures can be initiated from this log.

Examples of malfunctions that could show up in a timing log would be: Improper lubrication, dull tooling, improper set-up, variations in air or hydraulic pressure, malfunctioning hydraulic or air valves, slipping motor belts, etc.

Typical diagnostics are:

Limit Switches	- Limit switches not tripped
	- Limit switches not released
	- Limit switches overtime
Machine Functions	- Under cycle time
	- Over cycle time
	- Downtime & occurrences
	- Machine in auto time
	- Load & unload efficiences
Motor Starters	- Overloads tripped
	- Motor starter not picked up

A Diagnostic Summary Report will list all diagnostics in descending order of occurrence. Therefore, those malfunctions or abnormal conditions that happen the most will be listed first.

All system diagnostics are typed out when they occur. This information is summarized at the end of each shift, or period specified, or on demand. It includes the number of cycles of the machine, the number of parts run and the total accumulated data printed on the Teletype during the period.

BROKEN TOOL DETECTION

We can easily use the cycle timing count of the computer to give an accumulating count of the cycles to which a tool has been subjected - and therefore an indication of its dullness. We'll do even better. By adding some new and unique, low-cost transducers now being developed, it is reasonable to predict that faulty-tool detection will be available in one to two years.

The tool sensors envisioned will detect dull tools from their continuing operational behavior.

TOOL MAINTENANCE RECORDS

The computer, inherently able to record and condense data, can also be programmed to develop tool maintenance records in direct proportion to the data that the plants have available or are willing to make available. For example, take a transfer line that could enter a tool record, by tool number, each time the tool is introduced into the machine. The computer would tally the amount of cycles the tool encountered before it was replaced.

Suppose, then, the tool numbers are maintained through the resharpening process and then input to the computer by the tool changer the next time he changes the tool. Obviously a complete history can be maintained on an individual tool, the amount of cycles obtained, how many times it has been resharpened, an amount of cycles obtained during each machine cycle of the tool. A history on a tool breakage can also be collected for analysis.

TOTAL COMPUTER MANUFACTURING SYSTEM

In the past individual systems were purchased for use by different organizations within the same company. This resulted in redundancy, unlinked file structures, nonstandard documentation, impaired communication links, and often a waste of corporate resources. Recent computer developments have expanded the concept of total data and total data analysis, exposing the previously unappreciated usefulness of large amounts of highly-detailed data. A piecemeal approach is no longer valid; "control" embraces the whole operation of a business. Accordingly, a truly integrated control system must be planned that reaches all levels of the management hierarchy. A plantwide computer control, monitoring, and data-handling system offers centralized scheduling; control of product flow and production flow; a tool maintenance program; machine control; system monitoring and control; diagnostic reporting; and data concentration.

THE INFORMATION CENTER COMPUTER

The Information Center Computer is a general purpose, dedicated timeshared computer that includes a central processor, a memory, and a variety of peripheral equipment. The central processor governs all peripheral in-out equipment, sequences the program, and performs all arithmetic, logical and data handling operations. The processor is connected to one or more memory units by a memory bus and to the peripheral equipment by an in-out bus.

The functions achieved by this computer are:

1. *Scheduling:*

   One function of any system is to aid management in determining what sequence work should follow. Any number of occurrences can affect current work schedules. Effective scheduling can be achieved if managers receive current data on the use of materials, machines, and labor. This type of system would aid management in the decision process, resulting in improved forecasting and material procurement. Advantages would be less inventory, less backlog, increased efficiency, and naturally a higher return on investments.

2. *Product Flow Control:*

   Inventory control, warehousing, material handling, product assembly, and shipping are important factors in any manufacturing process. They represent a significant contribution or detriment to both the company's profit and costs.

3. *Production Flow Control:*

   Tighter control of production is possible then the company's desired outputs and constraints are expressed in a total system approach. Remember that production control is also quality control. When a total production control system is operating within the company's established standards, the percentage of bad parts will be reduced because defective material and faulty machines can be identified almost immediately.

Product liability is becoming a more important aspect of the manufacturing process. The two basic legal theories regarding liability for defective products are those of negligence and breach of warranty. The manufacturer can be liable for both the expressed and implied warranty. A greater social consciousness on the part of our society and the Federal Government's growing interest in consumer protection emphasizes the need for centralization and total recall of production data. A complete record can be kept on each product as it flows through the plant, including all processing with such entries as date, time, weight, etc. From this it is possible to reconstruct a full history of the conditions under which each item was processed. This could be used for quality analysis and process study. If parts were subject to recall, these records would simplify the tracing and permit effective measures to be incorporated into the processing of future parts.

4. *Parts Records:*

It is possible to develop a complete history on a part throughout its life in the manufacturing system, and save this history on tape or disk files for various kinds of retrospective study. Three projects involving parts history accumulation are presently underway at Entrekin Computers. One, which exemplifies the idea, breaks down like this: As raw parts enter the plant they are assigned a lot number. These lot numbers are entered into a computer system on a first-in, first-out basis. These lot numbers are tracked throughout the entire plant and condensed into final, serial-numbered units of shipped commodities. In one instance, the numbered commodity has 118 part lots associated with it. Each lot contains the following history:

- Which machines process the lot
- The amount of rejected parts per lot
- The average gaging tolerance
- The amount of repaired parts classified into what was repaired
- The quality control backup
- The average total machine efficiency during the processing of the lot (with relationship to the calendar base for processing)

If at a future time the customer experiences a product breakdown, the serial number of the assembled product can be used in tracing the faulty part back to its respective casting lot number. This means that a part call-back of finished assemblies can be limited to the specific lot of castings which has included the fatigued part. This makes for a much better supplier-customer relationship and simplifies deficient part callback. In some cases it will be an economical answer to complying with new Federal safety regulations.

5. *Production Control:*

The computer, again due to the broad data base it accumulates, can be advantageously applied to a plant production control work. It can evaluate one machine against another, give a better picture of the interrelationship between several machines, and pinpoint areas for system improvement. It can analyze the inventory requirement of raw materials and determine the amount of process inventory needed for each grouping of machines.

One analytical assistance it can provide - one particularly popular with production control people - is the separation of machine efficiency from part-handling efficiency, for a better evaluation of the process line.

For example, one system we developed, in analyzing machine efficiency, considers only the time it takes to process a part *from the moment it enters the load station* until the part is deposited at the unload station. Time used in loading is considered separately; thus a report will show the exact amount of time lost by machine due to no parts on the load, or parts that remain on the unload list.

Our corporation has been amazed by the types of data that production control has requested from our system data base for use in improving their overall production engineering. What *we* consider a relatively insignificant piece of data often proves to be key information when used by the production engineer, thinking in terms of his total systems configuration. We've seen a line speeded up significantly just because such data pinpointed a small system malfunction.

This is exemplified by an automatic system for feeding parts to a transfer line on a random basis. It was set up to scan the parts coming into the system on a time basis. As soon as the computer detects the time base of the incoming parts spreading out, it notifies the production foreman. He immediately takes steps to get the feeder machines back up to production capacity before the transfer line begins to starve for parts.

INTEGRATED SYSTEM

Logically it would be wise to blend all of the plant manufacturing systems into one integrated system. This system, employing one large realtime computer, would control a number of dedicated computer systems. Essentially this would be a computer-directed supervisory control system.

Each dedicated computer system would be a system capable of running independent of the master information center. These may be satellites with direct control of the machines, and/or data concentrators.

DATA CONCENTRATORS

The satellite system is designed for a control and monitoring mode of operation. In a Data Concentrator System this is not necessarily the case. A Data Concentrator approach differs from that of a satellite system. The Data Concentrator utilizes two modes of operation during the monitoring of a system. For discussion here we will call them MODE-1 and MODE-2.

MODE-1 - is a high-priority operation where the program monitors certain vital inputs at all times. These inputs will indicate completion of a machine cycle, production of a part, emergency stop, and machine down. With this mode of operation a constant monitor is kept on all inputs which produce vital data.

MODE-2 - is the sequence monitor mode. In this mode of operation the computer actually monitors the entire sequential operation of an entire machine. The reason is to detect and collect data which is not available in the first or second mode of operation. In this third mode the computer collects all the information of MODE-1, detects machine parameters, and times every motion of the machine to detect overtime conditions. The computer then interpolates this data and passes it in a meaningful form back to the data collection center.

METHOD OF COLLECTING INFORMATION

The Information Center is a total timesharing system. Each Data Concentrator passes information back to the Information Center during intervals determined by the Information Center.

The collection of information is under program control of the Information Center. This allows the Information Center to bid for information and to determine when a particular machine should enter into the MODE-2 operation. The Data Concentrator will actually maintain the control sequence program, but the Information Center will initiate the program to maintain a "bench-mark" for data collection and assembly.

In the MODE-1 operation there are a limited number of inputs which provide the vital information required for system status and conditions of operation. These inputs will enable the Data Concentrator to detect if the machine is down, the parts produced, the cycle time, machine efficiency, and cycles of operation. If the machine is down, the Data Concentrator will enter into the MODE-2 operation, check all inputs from the machine and identify the actual cause of the problem.

INTERCONNECTION

The Data Concentrators are located out in the plant, interlocked by data link to the Information Center. This system is modular in design and can be field-expandable to add control function or material handling facilities at a later date. If an individual Data Concentrator becomes completely filled, others can be easily added and integrated into the overall system. The Data Concentrators run completely unattended, whereas the Information Center, located in a central control room, would have an operator to route the diagnostic data and monitor the overall system.

The Data Concentrator has two modes of operation: critical and noncritical. Critical inputs are monitored continuously. Noncritical inputs are monitored periodically and when there is a malfunction in the system. Critical are such operations as part-counting inputs and back limit switch monitoring.

Each Data Concentrator timeshares its operation among the machines it monitors. The Information Center monitors a complete machine cycle in its turn. For example, if the computer is monitoring ten (10) machines and each have a 10-second cycle, the Information Center will monitor each machine once every one hundred (100) seconds. However, during normal operation and part counting, other critical inputs are still being monitored.

If a machine goes down, immediate service is rendered to that machine. The computer will examine its final state and determine the malfunction. The computer will print out a standard diagnostic as to the cause of the problem. The computer will detect all inputs for "released" or "not released" conditions. Overtime conditions and grounds are checked periodically. These will be implemented into the total figures in order to present compatible statistics in the Summary Reports.

The Information Center computer enables data collection on a timesharing basis and therefore more valuable use of computer time.

TREND MONITORING

The system has capability to detect apparent operation trends, either deterioration or improvement. This is done at the end of the week through a special report, an analysis of the daily logs. For example, daily failure of a limit switch will indicate a trend toward total breakdown. Other trends could be a daily increase in downtime, or an increase in overtime conditions and their duration, or the improvement of each of these conditions.

These programs can be run on demand at any time. This provides management with a more effective tool in implementing fast reactions to trouble situations and preventing them.

Trend programs also aid in evaluating corrective action to be taken, by reporting on improvement or lack thereof, and in providing figures for cost analysis and determining whether the corrective action was indeed profitable.

SYSTEM RESPONSIBILITY

Management should expect to obtain expert assistance from the supplier in selecting the hardware for the system. The user must have trained personnel to handle the system, but part of the maintenance responsibility should be in the hands of the system supplier. The scope and quality of the services provided in the package paid for by the user are obviously important. They should include system engineering, system maintenance, documentation, application assistance, and educational services.

SYSTEM DOLLAR RETURN

From early studies and past experience, computer systems have a 24 to 48-month payoff. A conservative 36-month payoff from date of start-up could be selected. A dollar value can be placed on the increased reliability of the system and the ease of repair - a direct reflection on downtime and loss of production. Complexity, changeability, and systems backup can have values affixed after actual evaluation. There is no real rule of thumb. Many aspects must be considered; complexity, plant physical layout, variance in equipment, cost, and system balance.

You cannot arbitrarily put a computer on each station or each transfer line or for each plant. In the machine tool industry, machines originally worked as small, single-purpose, single-head machines. This progressed into more complex machines and transfer lines. Ultimately entire plant production was done on one large transfer machine. This is not practical, nor is it practical to use a large number of single-purpose, single-tool machine. The answer lies somewhere in between.

The question of what equipment to use is hard to answer with a simple set of guidelines or a meaningful formula. The total system can be justified by increased production, improved quality, scrap reduction, improved managerial control, or higher overall efficiency developed in the total process. Ability to log production records may reduce warranty expenses. Actually, these should be expected when a total plantwide computer control and monitoring system is employed.

CAM - TODAY AND TOMORROW

by JAMES S. LAMB

IBM Corporation
White Plains, NY

INTRODUCTION

Two of my associates published a brief paper[1] highlighting significant Production Automation literature. It covered more technology than American industry has implemented. More has been written on Computer-Aided Manufacturing (CAM) than anyone can read, and the field is still largely unstructured.

Therefore this paper is undertaken humbly, less to contribute more literature than to encourage successful management direction of early CAM projects. It is addressed to the industrial executive more than to the data processing professional - but the latter must play the key support role for CAM success.

There is a subtle but recognized growing need for a new type of professional engineer, as a "Computer-Aided Manufacturing Systems Engineer". When these people grow in experience and stature, and college curricula evolve to enhance the discipline, manufacturing industries will be able to make quantum jumps in productivity, quality, and job satisfaction.

This paper defines the sector of the manufacturing company involving CAM, and the environment in that sector. It then looks at data processing functions of today which could be applied to production realities, and proposes four steps an executive can take to establish (under control) a CAM action plan today, rather than waiting for tomorrow.

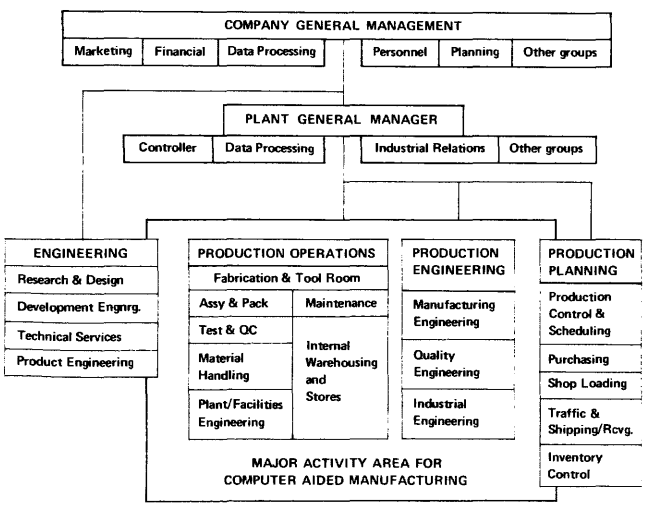


Figure 1. Functional Department Relationships to C A M in a Typical Manufacturing Company

DEFINING THE CAM-RELATED SECTOR OF A MANUFACTURING COMPANY

Figure 1 describes four major sectors as:

- General Management and Administrative (including Marketing, Personnel, Finance, and Data Processing functions);
- Development Engineering (where product requirements are converted to producible specifications for production and/or suppliers);
- Production Planning (often called production control) where master production schedules and sales forecasts are converted to net requirements against on-hand and on-order inventories, and the necessary new orders are placed on plants or suppliers and tracked to completion; and
- Production Engineering and Production Operations - the CAM sector - where manufacturing, industrial, quality and facilities engineering functions continually attempt to help the production operations staff operate the plant as a optimum macro system to comply with the plant manager's key measurements. These include logistical and cost targets, a "good place to work", federal and state regulations, market demands for quality, and service at a competitve price.

The CAM sector can also be defined further from an information flow perspective as shown in Figure 2.

The *Accept* function, for example, can happen with an "in-basket" and "quill pen" system, or can be largely digitized, as in the case of some leading edge systems such as CADAM,[2] where engineering drawings are given to production engineers in the form of magnetic media.

So it can be with each function in Figure 2. The translation of accepted data into production language and the *Generation* of "instructions" to operate processes, for example, can be entirely manual (as they often are today in "routing files"), or done with considerable computer help and provided as digitized input from successful Design Automation approaches.[3]

Communication to the operational work station on the shop floor can be by traditional internal company mail (envelope) or "electronic mail" (teleprocessing). *Operation* of a process can be fully manual, where the operator reads or has previously learned the operation to be performed, but we're all aware of increasing mechanization over the past 10 years where, e.g., transfer lines and automatic welders require only occasional manual intervention for set-up and maintenance. And now we see small punched-

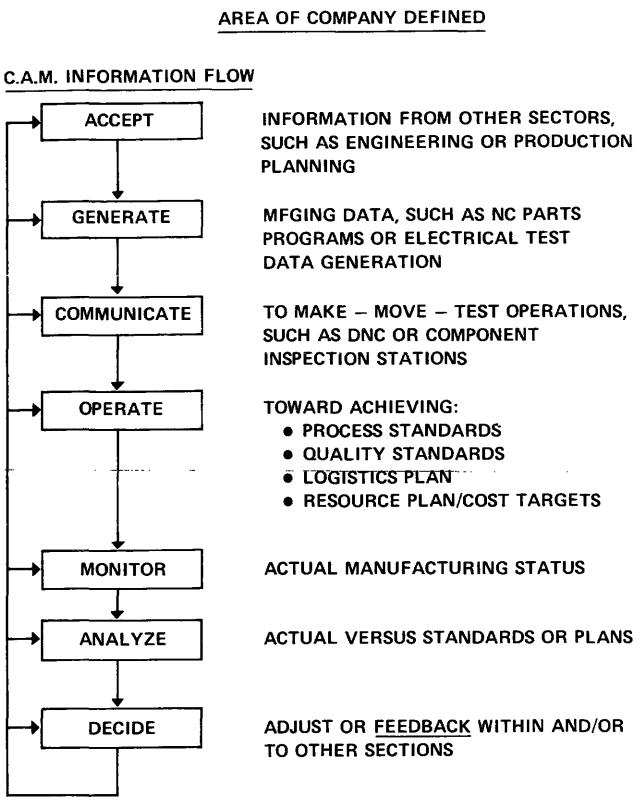


Figure 2. C.A.M. Information Flow

tape-driven computers or controllers doing more and more control of tedious, hazardous, or high-precision operations.

Today, of course, most of these tapes do not get to the minicomputer or controller by electronic mail, but rather by "envelope" or traditional methods. In fact, these early forms of digitized instructions are often stored in a traditional tool crib drawer rather than in an information handling machine, and the source code documentation, in annotated penmanship, is usually stored in another traditional file cabinet in the production engineering office.

The high cost associated with this type of data management has not yet been measured by industry, and in fact is so much "a way of doing business" that one would be hard pressed to justify converting many of these data sets to computer file management today, because he would find it hard to quantify the traditional costs displaced.

The *Monitor* and *Analyze* functions have been addressed in closed-loop control of operations for several years, but until now they have not been implemented very often with human decisionmakers in the loop. The batch mode today is what most industrial data processing centers are planning for, but are not yet ready to meet, the quasi-real time decisionmaking needs of knowledge workers (managers, expeditors, etc.) inside the CAM sector. In fact, most DP centers are still batch and volume output oriented to primarily service decisionmakers in other sectors such as cost accounting, and production and inventory control.

The *Decide* function in the CAM sector is an "online" function, and people in the sector have devised ingenious nonautomated systems to get at approximate facts quickly when critical decisions must be made. These ingenious systems are, of course, imperfect (e.g., red cards for hot jobs, yellow sheets for shortage lists each morning, and key points in the work flow to manually count and inspect throughput), but in 1973 they are all we have in many industrial plants where new high-performance tooling is commonplace (or sought within budget limits).

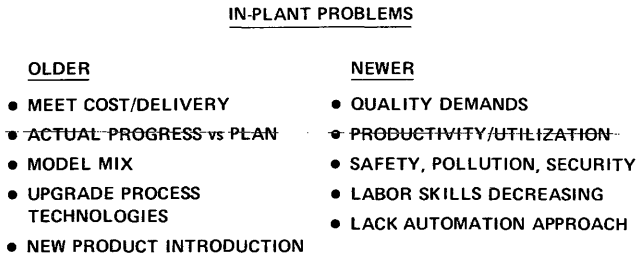


Figure 3 In-Plant Problems

ENVIRONMENT

Figure 3 identifies five older traditional problems and five problems with new meaning in the 70's which confront CAM sector management. We'll discuss the five newer problems. First, the examples of *quality demands* in the auto industry are widely known, and yield problems in semiconductor production demand vastly increased process and handling control.

Second, *productivity* is gradually being understood in the U.S.A. to be more than cheering on our machinists and assembly workers to achieve greater output; we know it somehow relates to the leverage a professional craftsman (or unskilled but willing worker) can gain through his efficient but meaningful interaction with increased capital tooling (e.g., crane operator). Output per man-hour is the classical economist's definition of productivity, but we have a long way to go in developing better measurement tools than that. If, for example, a tool operates with no direct man-hours and yields 4000 pieces yearly, is it a drag on year-to-year national productivity goals of over 3% increase? So the economist's measure may work at the plant or national level, but not always at the "work station" level. Lacking these measurements (even though some fine work has been done by MAPI[4] and others), neither the worker nor the tooling engineer is sure how to justify a case for increased capital outlays to tool up for large productivity gains.

The worker isn't sure he will be better off when retrained for his new job, and the engineer can't prove (he can "predict" to justify) his tooling will yield the expected benefits until he actually tools up, retrains the worker, and operates for a year or so.

Third, *safety and environmental* legislation and energy costs, plus industrial security, provide a suite of problems which are consuming increasing professional staff time and capital budgets. They're topical enough to assume the reader's awareness without more discussion.

While the three previous problems are working against CAM sector people, a fourth problem arises: those who comprise much of the work force are changing in *skill mix*. Most will agree that today's 18-30-year-old group is more populous, articulate, literate, and outspoken than at any time in our history. They also, however, are not enrolling as apprentice machinists, etc., as their grandfather may have done. The labor force is growing, but the old skills must be built into the equipment more and more.

One might conclude that a full-scale automation effort would be a logical plant strategy to address the four preceding problems. Apparently many firms plan to do this,[5] but it brings on a fifth problem. Most firms are *reticent about large automation* efforts because of uncertainty that real results match expected benefits. This lack of widespread computer-based automation experience is somewhat like the shortage of traditional computer systems people a decade ago.

The horns of the dilemma are either too fast an automation approach with poor results, or too slow a move to use today's available technology to solve key problems while competitors do. The basic need for professional CAM skills underlies this dilemma; as we begin to understand this need and fill it, we'll bridge the horns - the central theme of the rest of this paper.

DATA PROCESSING FUNCTIONS IN THE CAM SECTOR

Figure 4 shows a three-part logical systems framework for a CAM system. It suggests a "host" or sharing role for functions such as filing and access services (consider the digital data stored in the tool cribs on punched tape and the related hand-noted source code "upstairs"). It identifies the relationship of "computer terminal" functions for knowledge workers like quality engineers. It addresses data processing functions on the "plant floor" where minicomputers, controllers, transmission cables, sensors, consoles and data entry devices are employed.

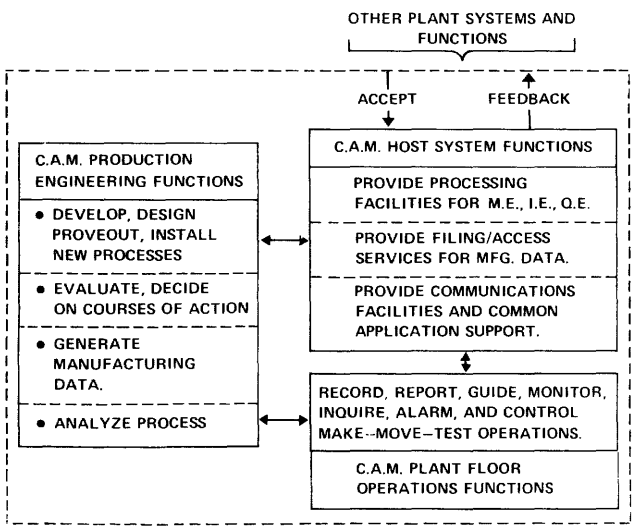


Figure 4. C.A.M. Logical Systems Framework

Figure 5 shows an example of a typical (and topical) application, Direct Numerical Control, in this logical systems framework. Any application of data processing to the CAM sector should provide for necessary functions in each of the three areas.

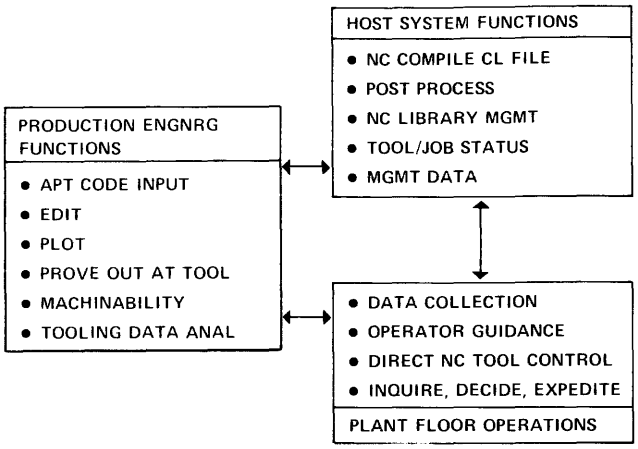


Figure 5. Example of NC/DNC Application Functions Within Logical System Framework

Consider the complexity of planning a plantwide system using the matrix in Figure 6. Six application zones interact with each other across the three logical systems areas. Further, one must address the functional requirements in the information flow discussed in Figure 2. There is no clear way to achieve the potential benefits across a plant without a strong central CAM plan managed by professionals.

SIX APPLICATION ZONES

	PROD'N ENGRG	HOST SYSTEM	PLANT FLOOR OPERATIONS	TOTAL
1. PLANT COMMUNICATIONS e.g. LOGISTICAL STATUS				10-25%
2. MACHINE MONITOR e.g. STAMPING LINES				10-25%
3. MACHINE CONTROL e.g. INJECTION MOLDERS				10-25%
4. MECHANICAL TEST e.g. COMBUSTION ENGINES				10-25%
5. ELEC/ELECTRONIC TEST e.g. RESISTORS, CARDS				10-25%
6. FACILITIES/POLLUTION MONITOR e.g. ELEC. POWER DEMAND				10-25%
TOTALS	← 40-60% →		40-60%	100%

Figure 6. Areas of Potential Benefit in C.A.M.

I prefer to call such a plan a "roadmap". It should directly attack key projected problems in the sector (year by year) while gradually building an orderly interactive data base. Figure 7 shows a DP manager's view of such a roadmap (simplified) from the top down. A CAM systems engineer's view might show much more tooling detail from bottom-up perspective with little except functional emphasis on the inside of the System/370 Information Management System at the top.

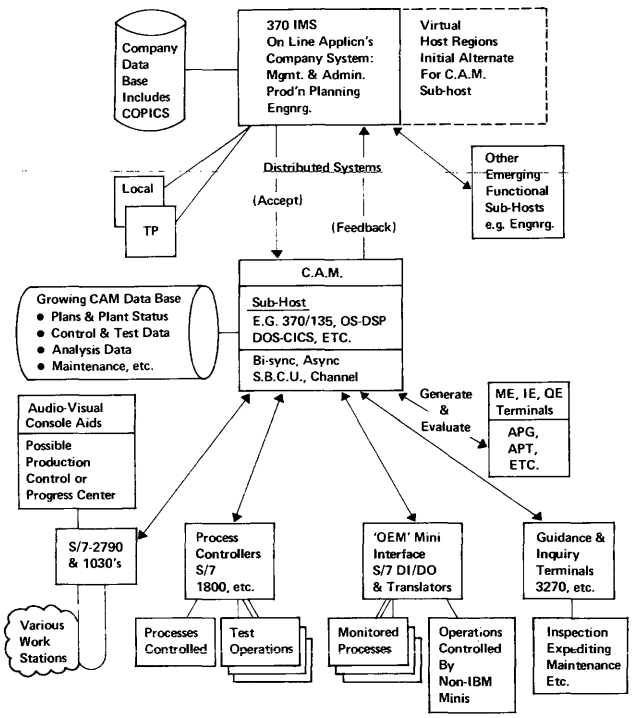


Figure 7. Generalized Example - Customer System Framework or "Roadmap"

Such a plan requires more functional CAM skill than even many above-average plants have inhouse today. Many are going to outside suppliers for help. It is doubtful that the full job in all plants can be done by outside contractors, and I am personally convinced that our real jumps in productivity, quality and job satisfaction will only come when inhouse teams are developed and experienced.

This is the job for professional manufacturing engineers to step up to. They will have to learn enough computer skill and physical interface skill to augment their tooling and costing know-how. Hard as this is, it would be more costly to have computer professionals try to learn the tooling business in the short range. Neither one, however, will succeed without the other, because the expanding world of data processing technology will never be fully mastered by the automation engineer, whose own growing field will keep him busy enough. There will have to be interdisciplinary CAM teams until the jobs are well understood, translated into university curricula, and sold to sophomores who will join the new branch of our technocracy in the 1980s and 90s.

CONCLUSION

Four steps are recommended today to Production Sector Management to begin the long journey toward the large and complementary gains in productivity that CAM will provide:

- Establish rapport between your production engineering management, data processing management, and your data processing supplier.
- Have staff work done for your approval to provide a projected environment for your plant several years ahead. (IBM plants, for example, update annually a 2-year and 7-year plan, including a projection of the environment to help focus on tommorow's problems today). Identify the prioritized top problems, and articulate planned courses of action to minimize problem impacts. State some expected benefits and acceptable paybacks and returns from CAM efforts (a basis for roadmap planning and project justification).
- Establish a professional CAM team (maybe 2 to 3 people initially) under your top tooling manager. Don't call them a committee, but rather give them an operational mission. Get them started developing a roadmap, but don't expect them and the DP manager to have all the answers before 6 to 12 months. In parallel, for experience, let them start one or two CAM projects which address some top priority problems; expect satisfactory results in 9 to 18 months and expect your data processing supplier to do his share to support your CAM team. Make sure they address software, interfacing, maintenance and justification standards.
- Manage! Assure yourself periodically that (1) the rapport is growing, (2) you translate promptly your changing priorities and expectations to your CAM team and "roadmap", (3) your CAM unit is on schedule and growing in the necessary professional skills and experience, and (4) your single "CAM roadmap" or plan can be implemented from the top down by your DP manager, and built from the bottom up by your M.E. manager. But be sure that each funded project moves along the "roadmap" so that "all your parts will fit together" to give you the leverage or synergy you expect.

REFERENCES

1. S.J. Oh and J.L. Hammond, *Computer Control in Manufacturing: An Annotated Survey of the State of the Art,* 1972 Joint Automatic Control Conf. of the Amer. Automatic Control Council.
2. *Lockheed Computer Graphics Augmented Design and Manufacturing Systems,* Lockheed California Co., Burbank, CA, 1972 Dec.
3. A.E. Fitch, *A User Looks at Design Automation: Yesterday, Today and Tomorrow,* Proc. SHARE, ACM and IEEE Design Autom. Workshop, 1969 June.
4. George Terborgh, *Investment Policy: A Challenge to American Management,* Machinery and Allied Products Inst., 1964.
5. Douglas Greenwald, *How Modern Is American Industry: A Progress Report,* McGraw-Hill, 1970 November. (McGraw-Hill data points out that capital expenditures for automation in durable industries (excluding primary metals) will increase from 25% of machinery and equipment outlays in 1965 (mostly mechanization) to 50% in 1975 (presumably with more systemization).

CONTROLLING THE RESOURCES OF A N/C FABRICATION DEPARTMENT: THE GAP BETWEEN COMPUTER SYSTEMS AND REALITY

by CLINT T. HAYS, JR.

Douglas Aircraft Company
Torrance, CA

Being a Fabrication Manager offers some peculiar opportunities to study the language of manufacturing. I contact shop supervision and workers on their level, and interface with executive management for interpretation. From short expressive sentences, dotted with 4-letter words, to eloquent *nonreceptive* dissertations on noncompliance to schedule, I live my life. Then, with Part Programmers who talk about APT, ASP, Arlem, Post Processors, computer priorities, R.J.E. units, main frame failures, high level languages, etc. I become disorientated. Then, finally, the computer specialist, who can do anything, but who has not, within a given time, completed one item. Suddenly, computer people frighten me.

In manufacturing we find some definite desires and goals that are translated into a specific terminology. A quality product, on schedule, below cost, are terms that basically explain a fabrication managers job.

The first item we can give up as a "trade off" is below cost. By the way, that is also the only item that we can give up in a fabrication oriented world, but not forever. The need for quality in aircraft should be obvious to all of us who flew to this meeting. Quality is then a constant item for control, but is a technical problem not necessarily people-oriented.

To be "on schedule" is really where people like me spend most of their time. It is an unforgivable sin in aircraft to miss an assembly start date for a fabrication detail. The tools we use to be on schedule vary from notes on matchbook covers to complicated computer printouts with voluminous information that is generally one week old. The matchbook cover notes are specific data on the hottest job in the factory, gathered by subordinate personnel, checked personally by a fabrication supervisor, then suspected as being 50% in error by the Fabrication Manager. If it's hot enough, the Manager will put "hands on" the job himself.

The computer data is generated by edge-punched fabrication orders or other automated systems and printed out on a batch basis on regular intervals. This interval can be one day or one week. In any event, most Fabrication Managers do not have real-time reporting of order location or job process. Most certainly they do not have an accurate exception to schedule alert through a computer system. So, Fabrication Managers spend most of their time with people-generated, handwritten exception to schedule reports requiring decisions for optional methods to guarantee schedule compliance.

At the Torrance facility of McDonnell Douglas we have solved part of the problems in the Numerical Control section with a manual Numerical Control Management Center that utilizes the best configured computers in the world. These are Fabrication Foreman computers, model 1924. These computers can furnish options, historical data, potential problems and solutions and an undying desire to satisfy the user. They are easy to program, subject to very little maintenance, inexpensive, and generally easy to justify. Their main faults are nonrapid retrieval of data, the common cold, and the inability to communicate without using 4-letter words.

Our system records, in near-real-time, the status of 70 N/C machine tools. Status here means operational status, up or down, machine hours negative to a schedule, machine hours ahead of schedule, individual job compliance to an operational machine schedule and to an end item completion schedule, and a queue of jobs preloaded or next in line for operational schedule. Also included are constraint items, priority items, and completion dates for elimination of constraints.

Our communication links to the Management Control Center are 18 telephones, located on the shop floor, to one receiver at the Management Center, an Electrowriter connected with service organizations and a broadcast system to the machine shop floor. Very simple, effective, and adaptable to immediate change.

We would like to automate this system. A computer (intelligent, of course), placed in a position to alert personnel to an exception to schedule, to demand response to a constraint, to store historical data, and to tell me upon demand (in a sentence or paragraph) what my problems or options might be, would be ideal.

I think that a computer might do this for me. However, the people who seem to control computers in this world always want the prerogative of saying what data I should receive, and when. Without knowledge of my basic responsibilities or goals, computer people always have a system that would satisfy me - *IF* I would learn to read and respond to computer outputs. I am not a scientist, I am not a mathematician, I give orders in shop-level language. I understand the American language. I wish to receive data in a quickly understood medium so that translation to shop-level instruction can be made.

Most computer-oriented people do not understand the rigors of a firm schedule or my responsibilities. Generally, I do not understand or communicate well with computer people.

To apply computer technology to a factory environment within the foreseeable future, I believe that computers will need to be shop programmable, and subject to the schedule compliance problems of todays and tomorrows nitty-gritty Fabrication Managers.

I believe in computers. They are the life line for my Numerical Control machines. They could be the answer to my sleepless nights. Let's put Computer Aided Manufacturing on a new priority for fabrication. Past Due, Forecast Shortage, Assembly Jig Down, Aircraft on Ground, and now for Computers,

HELP!

GLOSSARY OF MANUFACTURING TERMS

Assembly Start Date: The date in a manufacturing process when all components must be in a stockroom ready for assembly.

Constraint Item: A fabrication detail which cannot be completed on schedule because of a supporting function constraint.

End Item Completion Schedule: The date when a fabrication department must complete a fabrication detail.

Exception to Schedule: A fabrication detail process which is negative to a completion schedule.

Fabrication Detail: A manufactured part fabricated to meet Engineering design specification.

Fabrication Order: A set of specifications which detail the fabrication process and tools to be used in manufacturing a fabrication detail.

Operational Machine Schedule: A machine operation by machine operation schedule designed to meet an end item requirement.

Priority Item: Those fabrication detail processes which have violated the end item completion dates and require identification for management action. An exception to schedule.

Schedule Compliance: Not an exception to schedule.

METRICATION AND SYSTEMS DESIGN

by JOSEPH L. POKORNEY

Innovative Management Systems
Northbrook, IL

THE UNITED STATES METRIC TRANSITION

In 1790 the United States first considered and rejected a decimal measurement system. Thomas Jefferson presented a decimal system developed around a new "foot" based on a pendulum of such length that a swing from one end of its arc to the other and back would take two seconds. Subsequently, John Quincy Adams conducted an extensive study of the metric system and presented an eloquent discussion supporting its adoption in 1821. In 1866 Congress legalized the use of the metric measurements on an optional basis.

The U.S. signed the treaty of the Metre in 1889 even though we were not using the metric system. Four years later, by administrative order, the new International Metric Standards were declared to be the nation's "Fundamental Standards", and the foot, pound, etc., were defined as precise percentages of the metre, kilogram, etc. Thus the U.S. became *officially* metric but not *practically* metric. Various legislative proposals were made in later years, but none could muster enough support to ensure passage. During this time most of the world's nations moved to accept the metric system, until at the present time the U.S. and a handful of small African and Asian nations represent a nonmetric island in a metric world.

The preceding events led Congress to pass the Metric Study Act in 1968 August, directing the Secretary of Commerce to arrange for a broad inquiry and evaluation of metrication in America. In his report to Congress on 1971 July 29 entitled, *A Metric America - A Decision Whose Time Has Come,*[1] the Secretary of Commerce recommended that:

- The United States change to the International Metric System deliberately and carefully.
- This be done through a coordinated national program.
- The Congress assign the responsibility for guiding the change, and anticipating the kinds of special problems described in the report, to a central coordinating body responsive to all sectors of our society.
- Within this guiding framework, detailed plans and timetables be worked out by these sectors themselves.
- Early priority be given to educating every American schoolchild and the public at large to think in metric terms.
- Immediate steps be taken by Congress to foster U. S. participation in international standards activities.
- In order to encourage efficiency and minimize the overall costs to society, the general rule should be that any changeover costs should "lie where they fall".
- The Congress, after deciding on a plan for the nation, should establish a target date ten years ahead, by which time the U.S. will have become predominantly, though not exclusively, metric.
- There be a firm Government commitment to this goal.

The administration introduced a resolution (H.J. Res. 1092) on 1972 March 6, to the House of Representatives that would establish a 21-member National Metric Conversion Board to oversee the 10-year U.S. conversion program. On August 18 the Senate passed an earlier version of this Bill.

While the passage of this legislation will do much to accelerate the U.S. transition, in practice many metric units are already in common use in the U.S. We have purchased 35 millimetre film, 50 millimetre lenses, 100 millimetre cigarettes, and milligrams or millilitres of pharmaceuticals for years. Metric segments of the U.S. economy at present include pharmaceuticals, antifriction bearings, skis, photography, Olympic sports, NASA, and even automobile parts.

THE ADVANTAGES OF SI (SYSTEM INTERNATIONAL)

For the past 180 years the advantages and disadvantages of the metric measurement system have been bitterly debated in the United States. While many of the arguments against were based on emotional conservatism, there is no denial that conversion will cost the American public in both dollars and inconvenience. However, the cost of continuing to be a foot/pound island in the midst of a metric world is much greater.

Virtually every major industrial country and every trading partner of the United States is using a common metric measurement language. The current nonmetric islands in addition to the U.S. are:

Barbados	Jamaica	Sierra Leone
Burma	Liberia	Southern Yeman
Gambia	Muscat and Oman	Tonga
Ghana	Nauru	Trinidad

The modernized metric system has many benefits, but they can generally be summarized into four major points:

- SI is a unique measurement system in that each quantity has only one unit associated with it, e.g., length - metre; mass - kilogram; power - watt. This advantage is shown in Table I.
- SI is an absolute system which is independent of gravity and can be easily replicated anyplace. Most of the base units are defined in terms of absolute physical phenomena.
- SI is a coherent system; the product or quotient of two unit quantities produces a unit quantity. For example, a mass of 1 kilogram accelerated a distance of 1 metre per second produces a force of 1 newton (1 newton = 1 kg · 1 m/s^2). A coherent measurement system is not hampered by the many conversion factors intrinsic to the United States customary measurement system.
- SI is a simplified system because of the decimal structure and coherence. The use of fractions is reduced considerably in performing common calculations. The steps required for most common calculations are simpler, quicker and less error-prone.

Quantity	*SI Unit*	*U.S. Customary Units*
Length	metre	inch, foot, yard, rod, mile
Mass	kilogram	ounce, pound, ton
Temperature	kelvin	fahrenheit
Force	newton	dyne, ounce, pound, poundal
Energy	joule	foot pounds, British Thermal Units, calorie
Power	watt	horsepower, tons (cooling)
Pressure	pascal	pounds per square inch inches of H_2O or mercury, bar

Table I. Uniqueness of Metric Units

METRICATION AND DATA PROCESSING

Unfortunately, the National Bureau of Standards Metric Study totally overlooked the impact of metrication on data processing. While the major equipment manufacturers participated in the study, they chose to address only manufacturing problems while ignoring the systems and software implications. Thus the challenge of metrication goes unrecognized within the data processing industry at the moment. However, as we examine the impact of metrication it will become obvious that if we consider ourselves data processing professionals, then the challenge must be met immediately.

The transition to the modern metric measurement system will impact data processing systems in the following areas: the definition of data field sizes, numeric precision or accuracy, conversion of historical data, and the logic of mathematical calculations.

Data Field Sizes

Each metric unit is intrinsically more or less precise than the customary unit that it replaces. Thus centimetres are much more precise than inches, kilometres are more precise than miles, but metres are much less precise than feet, and kilograms are much less precise than pounds. This difference in accuracy dictates that metric units require more or less digits than do customary units to represent the same range of values. In an overly simple example, to represent 0 to 99 miles requires only two digits, while the equivalent range (in metric units) of 0 to 159 kilometres requires a data field of three digits. Only 62 miles, i.e., 99 kilometres, can be represented by two digits. Similarly, the representation of mass in kilograms will require fewer digits than pounds for various ranges of values. Thus 100 to 218 pounds requires three digits, while the metric equivalent of 45 to 99 kilograms uses only two. Obviously, as we begin to process metric measurement data, the selection of appropriate field sizes will become quite significant.

Accuracy

The inherent difference in precision also has a major impact on numeric accuracy. If using data in cubic inches (in^3) accurate to one decimal place or $\pm$ 0.05 inches, the same one decimal place in cubic centimetres (cm^3) would provide $\pm$ 0.05 cm, or $\pm$ 0.00305 inches, much more accurate than needed. However, if I am using data in pounds accurate to one decimal place $\pm$ 0.05 lb., then the equivalent one decimal place in metric kilograms provides accuracy to $\pm$ 0.05 kg or 0.110 lbs., which may not be adequate. The net effect of this difference in precision of each measurement system will be increased system sensitivity to field sizes, both to the right and left of the decimal point.

Historical Data

Systems that use measurement-sensitive data for forecasting, statistical analysis, or other analysis will be faced with a major discontinuity in data. It will be difficult to compare the last 5 years' automobile performance data in gallons/mile with next year's data in litres/kilometre. Cost-accounting systems will suddenly generate unit costs per kilogram or cubic metre, while all the previous data is in cost per pound or cubic yard.

Mathematical Calculations

The typical calculations that any data processing system perform are affected by the inherent change in units and also by the elimination of many customary conversion factors. Because the SI system is coherent, most of the traditional conversion factors are no longer needed. For example, if we are solving the following fuel consumption problem, the customary and metric calculations would be quite different.

Example A - Using Customary Units

A generator supplies 300 KW at 84% efficiency. What horsepower is required to drive it? If its driving engine efficiency is 30% and its fuel has a caloric value of 18,000 BTU/pound, what is its fuel consumption in pounds/hour?

$$\text{Power to generator} = \frac{300}{.84} = 357 \text{ kW}$$

$$\text{Horsepower required} = \frac{357 \times 10^3}{746} = 480 \text{ hp}$$

$$\text{Power input at 30\%} = \frac{480}{.3} = 1600 \text{ hp}$$

$$\text{Heat content of fuel} = \frac{1600 \times 550}{778} \text{ BTU/s}$$

$$\text{Fuel consumption} = \frac{1600 \times 550}{778 \times 18000} \text{ lb/s}$$

$$= \frac{1600 \times 550 \times 3600}{778 \times 18000} \text{ lb/hr}$$

$$= 227 \text{ lb/hr}$$

Conversion Factors Used

1 hp = 746 watts = 550 Ft-lb/s
1 BTU = 778 Ft-lb
1 hour = 3600 seconds

Example B - Using SI Units

A generator supplies 300 KW at an 84% efficiency. What power is required to drive it? If its driving engine is 30% efficient and its fuel has a calorific value of 42 megajoules/-kilogram, what is its fuel consumption in kilograms/hour?

$$\text{Power to generator (= Power required)} = \frac{300}{.84} = 357 \text{ kW}$$

$$\text{Power input at 30\%} = \frac{357}{.3} = 1190 \text{ kW}$$

$$\text{Heat content of fuel} = 1190 \text{ kJ/s (as 1 W = 1 J/s)}$$

$$\text{Fuel consumption} = \frac{1190 \times 10^3 \times 3600}{42 \times 10^6}$$

$$= 102 \text{ kg/hr}$$

Due to SI coherency, only the 3600 s/hour constant was needed. Conversion to metric units will impact all systems that perform routine calculations using customary measurement units. Computer-assisted design packages and other engineering/scientific data systems will be impacted most severely.

THE METRIC IMPACT

The degree of impact from metric conversion will vary depending upon the nature of the particular data processing system. Some systems will not be affected at all, or in such minor ways that they can readily accommodate the change. Other systems will have to be converted to accept and process both metric and customary data. It can be expected that some systems will be so difficult to convert that it will be more cost-effective to discard them and design replacements.

The types of decisions facing data processing analysts can be shown by looking at a simple problem:

An inventory system records data on gasoline consumption for various vehicles by processing transactions and maintaining totals. Some of the data is as follows:

Type	*Data Picture*	*Range in Gallons*
Transactions	99V9	0 - 99.9 ± .05
Monthly total	99999V	0 - 99,999 ± 50*
Yearly total	999999V	0 - 999,999 ± 500*

To process this data in metric units, the effect of processing litres must be examined. 1 litre = 0.2642 gallons or, conversely, 1 gallon = 3.785 litres. If the transactions are changed to record litres, the existing data fields limit the range of value as follows:

Data Picture	*Range in Litres*	*Range in Gallons*
99V9	0 - 99.9 ± .05	0 - 26 ± .013
99999V	0 - 99999 ± 50*	0 - 26,420 ± 13*
999999V	0 - 999999 ± 50*	0 - 264,200 ± 13*

In effect, the range of values that can be processed has been limited to approximately 1/4 of the previous range. To improve this situation, the transaction data picture could be changed by adding a digit or by using the three digits more effectively in this way:

Alternative A

Data Picture	*Range in Litres*	*Range in Gallons*
999V9	0 - 999.9 ± .05	0 - 264 ± .013
999999V	0 - 999999 ± 50*	0 - 264.200 ± 13*
9999999V	0 - 9,999,999 ± 500*	0 - 2,642,000 ± 130*

Alternative B

Data Picture	*Range in Litres*	*Range in Gallons*
999	0 - 999 ± .5	0 - 264 ± .13
999,999	0 - 999999 ± 500*	0 - 264,200 ± 130*
9,999,999	0 - 9999999 ± 5000*	0 - 2,642,000 ± 1300*

*Accuracy is limited by input data accuracy.

However, the decision of which method to use is a tradeoff availability of storage, actual range of the data, and the desired accuracy. The second alternative provides much less accuracy with no increase in digits, while the first alternative provides much improved and possibly unneeded accuracy at the cost of one more digit. This analysis must be performed for each measurement-sensitive data item, although in the example the monthly and annual totals were modified by adding a digit for simplicity.

Since the metric transition will progress in an orderly fashion over approximately 10 years, most systems will have to process both customary and metric units during the overlapping years. Typically, an inventory system or bill of materials processor would be required to handle both customary and metric-sized items. The result is a possible 10% to 30% increase in inventories or materials processed by such systems. A similar requirement for dual capabilities will exist in the generation of reports and in performing design calculations.

Clearly, the United States metric transition presents a unique challenge to the data processing industry, in that it will impact the total industry, the impact will proceed in an orderly fashion, and we are aware of it. Thus we can and must act to meet it. The long lead time for systems conversion and redesign, combined with the rapid acceleration of the metric transition, dictate that the true data processing professional take action now to meet the metrication challenge.

MEETING THE METRIC CHALLENGE

To minimize the impact of metrication on an organization's data processing system, the data processing manager must lead the way to a structured solution. He obviously will have an uphill battle since many people are either not aware, not interested, or nonbelievers as far as metric conversion is concerned. While major countries have converted to the metric system recently (i.e., Great Britain and Japan), none have been so dependent upon computers as is the United States, and thus we have no reservoir of experience from which to draw. In analyzing the metric challenge, the following major tasks become evident.

Metric Awareness Program

The data processing manager should initiate a metric awareness program at the top level of the organization. This program could include informal talks, seminars or workshops as appropriate. Essentially, everyone must be made to understand the inevitability of metrication and the degree to which it will or will not impact their operations.

Impact Analysis

An impact analysis of every data processing system application in operation or being designed should be conducted. This analysis should determine the degree to which the system's input, processing or output is dependent upon measurement-sensitive data. The result of the impact analysis should be a classification of all systems

systems in terms of the degree of impact from metrication. In conducting this analysis the life cycle of each system must be considered, since the decision to convert or redesign a system should be based on the total cost/benefit of each system decision. The expected life of a particular system will have a significant effect on the cost/benefit analysis.

Metric Conversion Plan

Each data processing manager should develop a metric conversion plan. The plan should show the time phasing of the metric capability for each system. Specific resources required for the change should be identified. All system users should be involved in developing this plan since they will bear the brunt of any metric transition problems.

Metric Design Efforts

All new systems being designed should reflect the results of the impact analysis study. All measurement-sensitive systems should be designed with dual capabilities, i.e., both customary units and metric units. The mathematical processing should be clearly separated from all logical operations to facilitate the eventual conversion to metric units. Obviously, field sizes should be selected with eventual conversion to metric as a primary factor.

THE METRIC OPPORTUNITY

While the U. S. metric transition is a challenge, it is also an opportunity for the data processing industry to demonstrate its professionalism by acting to reduce the effect of an inevitable change. Hopefully we can stop debating the relative worth of new hardware or software technology, stop engaging in the self criticism of the past few years, and have the foresight to anticipate a major change that will affect many of our users. If not, the EDP profession will again stand accused of not meeting the true needs of users. If each data processing manager would only analyze the impact of metrication on his organization, it would be possible at some future retrospective and historical meeting to point proudly at how easily we converted to processing centimetres instead of inches, litres instead of quarts, kilograms instead of pounds, and degrees kelvin instead of fahrenheit.

REFERENCES

1. "A Metric America - A Decision Whose Time Has Come", U. S. Department of Commerce, National Bureau of Standards Special Publication 345, 1971 July, p. III.
2. Daniel V. De Simone, "Moving to Metric Makes Dollars and Sense", *Harvard Business Review,* **50,** 100-111, 1972 January - February.
3. "Soon It May Be Give A Centimetre and Take A Kilometre", *Nations Business,* **59,** 88-89, 1971 April.
4. "Will Management Go Metric?", *Business Automation* **17,** 6-7, 1970 December 15.

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

AFIPS OFFICERS and BOARD OF DIRECTORS

President

Mr. Walter L. Anderson
General Kinetics, Inc.
12300 Parklawn Drive
Rockville, Maryland 20852

Vice President

Dr. Robert A. Kudlich
Raytheon Co., Equipment Division
Wayland Laboratory
Boston Post Road
Wayland, Massachusetts 01778

Secretary

Mr. Richard B. Blue, Sr.
TRW Systems Group
Scientific Data Processing Lab.
One Space Park—R3/1098
Redondo Beach, California 90278

Treasurer

Mr. George Glaser
McKinsey and Company, Inc.
3000 Sand Hill Road
Menlo Park, California 94025

Executive Director

Dr. Bruce Gilchrist
AFIPS
210 Summit Avenue
Montvale, New Jersey 07645

ACM Directors

Dr. Anthony Ralston
SUNY at Buffalo
Computer Science Department
4226 Ridge Lea Road
Amherst, New York 14226

Mr. Donn B. Parker
Stanford Research Institute
333 Ravenswood Avenue
Menlo Park, California 94025

Mr. Herbert S. Bright
Computation Planning, Inc.
5401 Westbard Avenue, Suite 520
Washington, D. C.

IEEE Directors

Dr. A. S. Hoagland
IBM Corporation
Dept. 29A—Building 021
P.O. Box 1900
Boulder, Colorado 80302

Professor Edward J. McCluskey
Stanford University
Department of Electrical Engineering
Palo Alto, California 94305

Dr. S. S. Yau
Department of Electrical Engineering
Stanford University
Palo Alto, California 94305

Simulations Council Director

Mr. Frank C. Rieman
Electronic Associates, Inc.
P. O. Box 7242
Hampton, Virginia 23366

Association for Computation Linguistics Director

Dr. A. Hood Roberts
Center for Applied Linguistics
1717 Massachusetts Avenue, N. W.
Washington, D. C. 20036

American Institute of Aeronautics and Astronautics Director

Mr. Frank Riley, Jr.
Auerbach Corporation
1501 Wilson Boulevard
Arlington, Virginia 22209

American Institute of Certified Public Accounts Director

Mr. Noel Zakin
AICPA
666 Fifth Avenue
New York, New York 10019

American Statistical Association Director

Dr. Mervin E. Muller
5303 Mohican Road
Mohican Hills
Washington, D. C. 20016

American Society for Information Science Director

Mr. Robert J. Kyle
Emery University
Upper Gate House
Atlanta, Georgia 30322

Instrument Society of America Director

Mr. Theodore J. Williams
Purdue Laboratory for Applied Industrial Control
Purdue University
Lafayette, Indiana 47907

Society for Industrial and Applied Mathematics Director

Dr. D. L. Thomsen
SIAM Institute for Mathematics and Society
97 Parrish Road, S.
New Caanan, Connecticut 06840

Society for Information Display Director

Mr. William Bethke
Rome Air Development Center
RADC (IS , W. Bethke)
Griffiss AFB, New York 13440

Special Libraries Association Director

Mr. Herbert S. White
Institute for Scientific Information
325 Chestnut Street
Philadelphia, Pennsylvania 19105

Association for Educational Data Systems Director

Dr. Sylvia Charp
Director of Instructional Systems
The School District of Philadelphia
Board of Education
5th and Luzerne Streets
Philadelphia, Pennsylvania

JOINT COMPUTER CONFERENCE BOARD

President

Mr. Walter L. Anderson
General Kinetics, Incorporated
12300 Parklawn Drive
Rockville, Maryland 20852

ACM Representative

Dr. Herbert R. J. Grosch
National Bureau of Standards
Center for Computer Science
Washington, D. C. 20234

Vice President

Dr. Robert A. Kudlich
Raytheon Company Equipment Division
Wayland Laboratory
Boston Post Road
Wayland, Massachusetts 01778

IEEE Representative

Dr. S. S. Yau
Department of Electrical Engineering
The Technological Institute
Northwestern University
Evanston, Illinois 60201

Treasurer

Mr. George Glaser
McKinsey and Company, Inc.
3000 Sand Hill Road
Menlo Park, California 94025

SCI Representative

Mr. Paul W. Berthiaume
Electronic Associates, Inc.
185 Monmouth Park Highway
West Long Branch, New Jersey 07764

JOINT COMPUTER CONFERENCE COMMITTEE

Mr. Al Hawkes, Chairman
Computer Horizons
53 W. Jackson Boulevard
Chicago, Illinois 60604

JOINT COMPUTER CONFERENCE TECHNICAL PROGRAM COMMITTEE

Mr.Henry S. MacDonald, Chairman
Bell Laboratories
Murrary Hill, New Jersey 07971

1974 NATIONAL COMPUTER CONFERENCE CHAIRMAN

Dr. Stephen S. Yau
Department of Computer Science
Northwestern University
Evanston, Illinois 60201

1973 NATIONAL COMPUTER CONFERENCE COMMITTEES

Technical Program Committee

Methods and Applications Committee

Robert W. Bemer—Chairman
Honeywell Information Systems

Gregory P. Williams—Consultant

Science and Technology Committee

Carl Hammer—Chairman
UNIVAC

William Bethke
Rome Air Force Development Center

Herbert S. Bright
Computation Planning, Inc.

Betty B. Brociner
MIT Lincoln Laboratory

Sylvia Charp
School District of Philadelphia

Bill Jameson
Spectra Associates

Philip Kiviat
Federal ADP Simulation Center

Ned R. Kornfield
Widener College

Special Program Committee

Charles V. Freiman—Chairman
IBM Corporation

Rhoda Grayson
IBM Corporation

Robert M. Landau
International Development Center

Arthur C. Lumb
Proctor & Gamble Company

Mervin E. Muller
World Bank

A. Hood Roberts
Association for Computational Linguistics

William Schmidt
COMSAT Laboratories

Noel Zakin
The American Institute of Certified Public Accountants

Exhibits Committee

Roy Gould—Chairman
Digital Equipment Co.

Ralph Show—Vice-Chairman
Tektronics, Inc.

Jerry Castanzo
Hewlett-Packard

John Sullivan
Data General Corporation

Steve Bowers
General Automation, Inc.

Warren Pugh
Omron Systems, Inc.

Curt Anderson
TEC, Incorporated

Local Arrangements Committee

Ruben Maldonado—Chairman
Brandon Applied Systems

Patrick Cunniff—Vice-Chairman
American Airlines

Lori Capodanno
Bell Labs

John J. Quirk
American Airlines

John Sullivan
Hewlett-Packard

Luke Ward
Omron Systems, Inc.

Jerry Johns
Texas Instruments Incorporated

Thomas Johnson
Control Data

James Morris
Datamation

Burt Totaro
Datapro Research Corporation

Edward Lord
Burroughs Corporation

DISCUSSANTS, MODERATORS, PANELISTS, SPEAKERS

Abbas, Daniel (M&A)
Aines, Andrew (S&T)
Alexander, John (S&T)
Amara, Roy (M&A)
Armand-Pilon, Mario (M&A)
Ashley, George, (M&A)
Ashworth, A. W. Jr. (S&T)
Atchison, William F. (S&T)
Barlow, John S. (S&T)
Baskir, Lawrence, M. (M&A)
Baynard, Ernest C. (M&A)
Beaton, Albert, (S&T)
Bell, Tom (S&T)
Bence, Robert (M&A)
Bernard, Tom (M&A)
Benkovich, Andy (M&A)
Bernstein, Morton I. (S&T)
Bickford, Reginald (S&T)
Bigelow, Robert P. (M&A)
Bitz, Ira (S&T)
Boehm, Barry W. (M&A)
Boettinger, Henry (S&T)
Bond, P. (S&T)
Bond, Robert T. (M&A)
Brady, Ray (S&T)
Brady, Ron (M&A)
Braithwaite, Tim (S&T)
Branscomb, Lewis M. (M&A)
Brody, Edward (M&A)
Browne, Peter S. (S&T)
Bryant, Lynwood (S&T)
Burns, Don (S&T)
Burrows, James H. (M&A)
Cacciamani, Eugene R. (S&T)
Caravella, Robert T. (S&T)
Carlton, James (M&A)
Castruccio, Peter (S&T)
Chambers, John M. (S&T)
Chartrand, Robert L. (M&A)
Clapper, Roy (S&T)
Clayton, James M. Jr. (S&T)
Cook, Robert W. (M&A)
Corin, T. (S&T)
Couger, J. Daniel (S&T)
Crocker, Dean (S&T)
Culbertson, Donald (S&T)
Cumberpatch, James R. (S&T)
Cunningham, Joseph F. (S&T)
Danbury, Thomas (M&A)
Daniels, W. Riley (M&A)
Davis, Keagle (S&T)
Davis, Robert B. (M&A)
Davis, Ruth (S&T)
Day, Melvin, S. (M&A)
Doddridge, E. T. (M&A)
Dougherty, William (M&A)
Dreyfus, Philippe, (M&A)
Duggan, Michael A. (S&T)
Dunn, Robert (S&T)
Durand, G. C. (S&T)
Elashoff, R. M. (S&T)
Elias, Arthur (S&T)
Ephron, Henry D (S&T)
Evans, James (M&A)
Fano, Robert M. (S&T)
Farmer, James (M&A)
Faust, Hilda (S&T)
Field, William B. (S&T)
Fleischer, Richard (M&A)
Foster, C. L. (S&T)
Foster, Lawrence (S&T)
French, Hiram (S&T)
Fruchter, M. (S&T)
Fryer, Fred (M&A)
Gassman, Hans P. (M&A)
Galbi, Elmer W. (M&A)
Gatfield, Allen (S&T)
Gazis, Denos (M&A)
Gentile, John (M&A)
Giloi, Wolfgang (S&T)
Gimby, Robert A. (M&A)
Goldberg, Morton David (M&A)
Golding, Edwin, (S&T)
Goodman, Glenn (M&A)
Gorn, Saul (S&T)
Gorsline, George (S&T)
Grad, Burton (M&A)
Greene, Jerome (M&A)
Grosch, Herbert R. J. (S&T)
Guiltinan, Richard (S&T)
Harbison, Earl H. (S&T)
Heller, Jack (S&T)
Heller, Nelson B. (M&A)
Herzog, Bertram (S&T)
Herzog, Gerald B. (M&A)
Hillman, Donald J. (S&T)
Hinds, Walter (M&A)
Hobbs, L. C. (S&T)
Hoge, Robert R. (M&A)
Hopper, Grace M. (S&T)
Hoyt, Patrick (S&T)
Hunter, John M. (M&A)
Hunter, Kenneth (M&A)
Ivie, Evan L. (S&T)
Jackson, Channing (S&T)
Jacobs, Robert A. (S&T)
Jaumot, Frank E. (M&A)
Keefauver, William F. (M&A)
Kleinrock, Leonard (S&T)
Kimbleton, Stephen R. (M&A)
Kolence, Kenneth W. (M&A)
Koller, Herbert R. (S&T)
Koslow, Sidney, (M&A)
Kurita, Shobel (M&A)
La France, Jacque (S&T)
Larson, D. F. (M&A)
Larrowe, Vernon L. (M&A)
Latker, Alec C. (S&T)
Licklider, J. C. R. (S&T)
Lucas, Steve (M&A)
Macon, Nathaniel (S&T)
Madnick, Stuart E. (S&T)
Mancina, William P. (S&T)
Martini, G. S. (S&T)
Martino, Joe (S&T)
Matejka, D. (S&T)
Mayers, Harry R. (S&T)
Melkanoff, Michael A. (S&T)
Melody, William (M&A)
Milgrim, Roger, M. (M&A)
Mills, Harlan D. (S&T)
Mills, Richard D. (S&T)
Milstein, Jeffrey (M&A)
Minor, I. J. (S&T)
Moler, Cleve B. (S&T)
Morris, Frederick E., Jr. (S&T)
Mullineaux, R. (S&T)
Murdock, John (S&T)
McCoy, Charles R. (M&A)
McDonald, James E. (M&A)
McGurk, Dan L.(M&A)
McGurn, Thomas (M&A)
Nolle, Fred (M&A)
Oelkers, Robert F. (M&A)
O'Leksy, William (M&A)
Oettinger, Anthony (S&T)
Parncutt, Geoff (S&T)
Phillips, Thomas (M&A)
Pipberger, Hubert (S&T)
Pluto, Raymond (M&A)
Peohimann, Karl (M&A)
Pohl, D. (M&A)
Pokorney, Joseph L. (M&A)
Porter, James (M&A)
Power, William (S&T)
Prasad, H. R. (M&A)
Purcell, Chuck J. (S&T)
Quirk, Francis J. (S&T)
Randall, Donald A. (M&A)
Rasmussen, Norman L. (S&T)
Richardson, John M. (M&A)
Risso, George (S&T)
Roberts, A. Hood (S&T)
Robinson, Clifford (M&A)
Rogers, Robert H. (M&A)

Roglieri, John (M&A)
Romberg, Bernhard W. (S&T)
Rosenthal, Charles (S&T)
Rossmassler, Stephen (M&A)
Roth, Sydney J., Sr. (M&A)
Ryan, Frank (M&A)
Rydell, C. Petter (M&A)
Sackman, Harold (M&A)
Saltzberg, Bernard (S&T)
Sammet, Jean (S&T)
Samson, Thomas (S&T)
Satterfield, Carroll D. (M&A)
Schaefer, Richard (M&A)
Schatz, Vernon (M&A)
Schmidt, William G. (S&T)
Schramm, George (M&A)
Schreiber, Robert (M&A)
Scott, Robert H. (S&T)
Sedelow, Sally Y. (S&T)
Sherrod, John (S&T)
Smith, J. I. (S&T)
Snyder, James N. (S&T)
Stallings, Rex (M&A)
Stefferud, Einer (S&T)
Stephans, Emery (M&A)
Stevens, Charles (S&T)
Stokes, Richard A. (S&T)
Stone, Robert (S&T)
Strable, Edward A. (S&T)
Strassburg, Bernard (M&A)
Stultz, Norman F. (M&A)
Suhler, William (S&T)
Swersey, Arthur J. (M&A)
Taggart, William (S&T)
Torrance, Anthony R. (S&T)
Tukey, John (S&T)
Turoff, Murray (S&T)
Tyndall, Gene R. (S&T)
Tyson, H. Blair (M&A)
Tyulina, Natalya (S&T)
van Atta, Richard (M&A)
van Dam, Andries (S&T)
Varnl, Robert (M&A)
Viguere, Richard (M&A)
Walden, David C. (S&T)
Walker, Robert K. (M&A)
Walters, William (M&A)
Weaver, Adrian G. (M&A)
Weil, John W. (S&T)
Weinberg, Gerald M. (M&A)
Weiss, Edward C. (S&T)
Welke, Larry A. (M&A)
Weitz, R. (S&T)
Wilkes, William E. (M&A)
Woods, William A. (S&T)
Young, Harry (M&A)
Yovits, Marshall C. (S&T)
Zakin, Noel (S&T)
Zimmerman, Robert M. (M&A)

REVIEWERS

SCIENCE AND TECHNOLOGY

Aagaard, J.
Abrial, J.
Adams, E. N.
Aiken, R. M.
Anderson, R. H.
Anderson, T. C.
Armer, P.
Arndt, F. R.
Arnovick, G. N.
Aron, J. D.
Augustin, D. C.
Aupperle, E. M.
Bailey, E.
Baker, F. T.
Ball, N. A.
Bareiss, E.
Barlow, A. E.
Barney, D. J.
Baskett, F.
Belady, A.
Bell, T. E.
Beretvas, T.
Berning, P. T.
Bernstein, R. L.
Bjorner, D.
Black, D. V.
Bloomfield, J. A.
Bodoia, M. J.
Bond, N.
Booth, G. M.
Bork, A.
Borko, H.
Bowdon, E. K.
Brandon, D. H.
Bratman, H.
Bredt, T. H.
Bremer, J. W.
Brennan, R. D.
Brown, R. R.
Browne, J.
Bryan, E.
Campbell, J. B.
Cardwell, D. W.
Carlson, C. R.
Carmichel, R. L.
Chernak, J.
Cheydleur, B. F.
Coffman, E. G.
Cohen, D.
Coles, L. S.
Coltin, A. H.
Condon, S. F.
Cook, W. F.
Corduan, A. E.
Cowan, R. J.
Crockett, D.
Daetwyler, D. W.
Daniels, W. E., Jr.
Davidson, D.
Davidson, D. R.
Daykin, D. R.
Derby, R.
Diss, C. E.
Dockery, H. L.
Dodd, G. G.
Downing, K. D.
Duggan, M. A.
Dumey, A. I.
Eccles , W. J.
Eckhouse, R. H., Jr.
Elfant, R. F.
Elliott, G. R.
Engel, G.
Enslow, P. H., Jr.
Erickson, R. F.
Farmer, N. A.
Ferrari, D.
Feurzeig, W.
Feustel, E. A.
Fife, D. W.
Firschein, O.
Fisher, D.
Fischler, M. A.
Foley, J.
Foster, C. L.
Foster, J. E.
Fox, M. R.
Gaines, R. S.
Gelenbe, E.
Gilliland, B. E.
Goldblatt, R. C.
Gordji, S.
Gorschboth, F. F.
Gosden, J.
Gotterer, M. H.
Grace, A. G., Jr.
Grace, D.
Grau, A.
Gruenberg, E. L.
Hall, D. M.
Hamblen, J. W.
Hamilton, D. E.
Hartwick, R. D.
Hastbacha, A. A.
Heafner, J. F.
Hedrick, G. E.
Henschen, L. J.
Hermann, P. J.
Hertlein, G.
Herzog, B.
Heterick, R. C., Jr.
Higgins, A. N.
Highland, H. J.
Holden, A. D. C.
Hollander, G. L.
Hollingworth, D.
Hopgood, F. R. A.
Hsiao, D. K.
Hutt, A. E.
Hyman, M. A.
Irwin, M.
Jessep, D. C.
Johnson, D. L.
Jones, W. J.
Jordan, B. W.
Jordan, S. R.
Joseph, A. F.
Kain, R. Y.
Katonak, P.
Kaufman, M. T.
Keller, R. E.
King, R. E.
King, W. F.
King, W. K.
Klerer, M.
Knight, D. W.
Koory, J.
Kopf, J. O.
Kosy, D. W.
Kovach, L. D.
Kreuder, N.
Kroeneberg, A. B.
Krulee, G. K.
Kuney, J. H.
Kurtzberg, J. M.
Langdon, G. G., Jr.
Lasser, D. J.
Lauro, J.
Leavitt, M. R.
Lennon, W.
Lindenmeyer, L. R.
Linger, R. C.
Liskov, B. H.
Lomet, D. B.
Lowe, T. C.
Luehrmann, A.
Lukas, G.
McKnight, R. S.
McMurran, M. W.
MacClary, D. R.
Machover, C.
Magness, F. M.

Maple, C. G.
Marcotty, M.
Mathison, S. L.
Mattson, R.
Matyas, M.
Merrill, R. D.
Mezei, L.
Miles, E. P., Jr.
Miller, S. W.
Mills, H. D.
Minker, J.
Mitchell, B. A., Jr.
Mittman, B.
Moran, J. D.
Morton, K.
Muntz, R. R.
Nemeth, A. G.
Nielsen, W. C.
Noe, J. D.
Onyshkevych, L. S.
Owen, J. T.
Paden, D. R.
Patterson, J. W.
Pearson, K. M., Jr.
Penderghast, T. F.
Pettygrove, F. A.
Pinson, E.
Pizer, S. M.
Raben, J.
Ray, L. C.
Redmond, E. V.
Rigney, J. W.
Roberts, D. C.
Rosenfeld, J. L.
Samek, M. J.
Sashkin, L.
Savas, M. A.
Schey, H.
Schwartz, M. H.
Seals, E.
Sedelow, S. Y.
Sedelow, W. A.
Seymour, J.
Shahbender, R.
Shank, G. D.
Shelton, B.
Shemer, J.
Shohara, S.
Shuey, R. L.
Sibley, E. H.
Silvern, L. C.
Simmons, W. G.
Skelly, P. G.
Smith, P. H., Jr.
Sparrow, C. A.
Spier, M.
Springe, F. W.
Srodawa, R. J.
Stanley, W. I.
Stevens, S. S.
Stewart, R. M.
Stone, D.
Stone, G. C.
Sudborough, I.
Sutcliffe, A.
Tang, C. K.
Taylor, R. W.
Tennbaum, J. M.
Teplitz, A.
Tesler, L. G.
Thompson, M. D.
Timmreck, E. M.
Ullman, J. D.
Vemuri, V.
Venesky, R. L.
Vere, S.
Wachal, R. S.
Walden, D. C.
Walters, T. L., Jr.
Weiss, E. A.
Weiss, S.
Whelan, J. J.
White, S. A.
Whitney, D. E.
Wiederhold, G.
Willett, R. M.
Williams, T. G.
Wishner, R. P.
Wolfe, E. W.
Wolla, M. L.
Wright, K. R.
Wyllys, R. E.
Wyman, J. C.
Yau, S. S.
Young, J. W.
Zinn, K. L.

SESSION CHAIRMEN

Abrams, Marshall D. (S&T)
Aines, Andrew A. (M&A)
Allen, Rodney (S&T)
Anderson, Robert H. (M&A)
Andrus, William, E., Jr. (M&A)
Anzelmo, Frank (M&A)
Barnes, Robert F. (S&T)
Barr, William J. (S&T)
Baynard, Ernest C. (M&A)
Belzer, Jack (S&T)
Berra, Bruce (S&T)
Bigelow, Robert P. (M&A)
Bitz, Ira (S&T)
Blum, Edward (M&A)
Boehm, Barry W. (M&A)
Bright, Herbert S. (S&T)
Brociner, Betty B. (S&T)
Bryce, Milton (M&A)
Buerger, Gail (M&A)
Charp, Sylvia (S&T)
Clayton. Jim, Jr. (S&T)
Cotton, Ira W. (S&T)
Dodd, George (S&T)
Dorn, Philip H. (S&T)
Draper, George L. (M&A)
Dunning, Bonnie (S&T)
Engel, Gerald L. (S&T)
Ephron, Henry D (S&T)
Ernst, Herbert M. (S&T)
Fisher, Tom (M&A)
Foster, C. L. (S&T)
Gagliardi, Ugo O. (S&T)
Gentile, John (M&A)
Gilbert, Fenton L. (M&A)
Gilchrist, Bruce (M&A)
Goetz, Martin (M&A)
Gosden, John (M&A)
Groner, Gabriel F. (M&A)
Gulotta, Charles (M&A)
Halbrecht, Herbert Z. (S&T)
Hampson, Richard K. (M&A)
Highland, Harold, J. (S&T)
House, Peter W. (S&T)
Hsiao, Ben M. Y. (S&T)
Jameson, William J. Jr. (S&T)
Joshi, Aravind K. (S&T)
Kasarda, Andrew J. (S&T)
Kiviat, Philip J. (S&T)
Kornfield, Ned R. (S&T)
Krieger, Michael M. (S&T)
Kuschnerus, Hans J. (M&A)
Landau, Robert M. (S&T)
Ledley, Robert S. (S&T)
Lesk, Michael E. (S&T)
Luebbert, William F. (S&T)
Lundell, E. Drake, Jr. (M&A)
Lumb, Arthur C. (S&T)
Lynch, John T. (S&T)
Machover, Carl (S&T)
Mainwaring, Herbert J. (M&A)
McGraw, Dan (M&A)
McKay, Douglas B. (S&T)
Melkanoff, Michael A. (S&T)
Muller, Mervin E. (S&T)
Parker, Donn B. (M&A)
Porter, James (M&A)
Potts, Jackie (S&T)
Printz, Dan M. (M&A)
Roberts, Lawrence G. (S&T)
Rotolo, Louis S. (S&T)
Savides, Peter (M&A)
Schmidt, William G. (S&T)
Shipton, Harold W. (S&T)
Shuey, Richard L. (S&T)
Steel, Thomas B. (M&A)
Sweeney, Joseph P. (M&A)
Tucker, Dorothy (M&A)
Turoff, Murray (S&T)
Tyson, H. Blair (M&A)
Ward, James A. (S&T)
Williams, Gregory P. (M&A)
Zakin, Noel (S&T)

AUTHOR INDEX