

Project No. 007 001 01
Document Number
TRACOR 68-1360-U

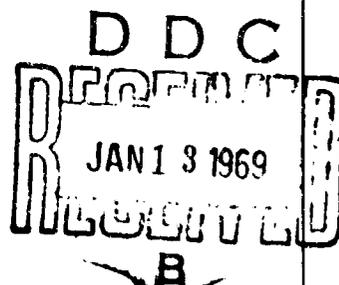
AD 680249

AN ITEM STORE: ITS DESIGN AND IMPLEMENTATION

by

T. W. Ziehe

December 1968



TRACOR

1500 ...

This document has been approved
for public release and sale; its
distribution is unlimited

Approved by the
CLEARINGHOUSE
for Federal Scientific & Technical
Information (NSA/CSS)



6500 TRACOR LANE. AUSTIN, TEXAS 78721

Project No. 007 001 01
Document Number
TRACOR 68-1360-U

AN ITEM STORE: ITS DESIGN AND IMPLEMENTATION

by

T. W. Ziehe

December 1968



6500 TRACOR LANE, AUSTIN, TEXAS 78721

ABSTRACT

Three types of information -- data, capabilities in symbolic form, and knowledge -- are distinguished in an informal manner. The role of each within an information system is sketched as the basis for a discussion of the item store. The item store is a general-purpose formatted store which will serve as a repository for files of inter-related items. The tree serves as the organizing principle for files within the store. The operational modes for the store are described as are the techniques being used to implement these operational modes.



6500 TRACOR LANE, AUSTIN, TEXAS 78721

ACKNOWLEDGMENTS

This report and the work it represents are being sponsored by the Information Systems Branch of the Office of Naval Research under Contract N00014-67-C-0396.



6500 TRACOR LANE, AUSTIN, TEXAS 78721

TABLE OF CONTENTS

<u>Section</u>	<u>Page No.</u>
Abstract	ii
Acknowledgments	iii
Preface	v
1. Introduction	1
2. Overview of the Approach	6
3. The Complex of Stores	9
4. Item Files - Contents of the Item Store	15
5. Item Store Operations	23
6. Implementation	28
7. Conclusion	39
References	41



6500 TRACOR LANE, AUSTIN, TEXAS 78721

PREFACE

This paper represents the material presented by the author at the colloquium, "Integration of Computerized Bibliographies and Bibliographical Systems in the Arts and Humanities," held as part of the Thirty-first Annual Meeting of the American Society for Information Science in Columbus, Ohio on October 20, 1968. Although the material covered in Section 4 was published earlier in the report An Organizational Form for Item Management (TRACOR Report 67-1111-U, February 1968), it is included here in summary form for the sake of completeness.



6500 TRACOR LANE, AUSTIN, TEXAS 78721

AN ITEM STORE: ITS DESIGN AND IMPLEMENTATION

1. Introduction

The computer's potential as an information handler is being explored and exploited at an ever increasing rate. Interest is being spurred by advances such as the following:

1. Impressive increases in storage capacity, speed, and density.
2. The confluence of the computer and communications technologies.
3. The development of improved techniques for data management.
4. The prospect of logical designs tailored to requirements in individual applications.

But in spite of these advances, application of the computer as an information handler is often hampered by the haziness of our concept of what "information" is and, therefore, how to "handle" it. This is not to deny that computers have been successfully applied to many information handling tasks where they are currently providing useful service. However, difficulties and failures of many kinds have been experienced as well. A considerable portion of some current efforts are, in fact, attempts to apply lessons learned from past failures. But the pursuit of such historical matters is not our main interest here.



The interest which is the focus of our attention is that of using the computer to handle what we often call language data and, in particular, bibliographic material. In such applications "handling" certainly includes the rote movement and arrangement of the data. In some it also includes (1) responding to stimuli, both the stimuli and the responses couched in natural language, and (2) decision-making which is based on analyses of stimuli and stored data, the analyses being of sufficient depth to produce non-trivial responses.

The work described in this report has not yet reached the operational stage. The reason we have not proceeded more directly toward implementation is part of the message to be conveyed. That reason has its roots in two experiences frequently encountered by those who apply the computer as an information handler. For example, consider the case of a librarian who wants to offer a computer-based bibliographic service. The two experiences are these:

1. The individual interested in offering the service encounters questions and problems which are discouraging because they are outside his immediate interest and perhaps competence. Questions such as a) what techniques should be used to encode the necessary data, b) how should the logical relationships of the data be represented, c) what type of storage devices will be most effective, d) how should the capabilities of the



system be implemented? These and similar questions must be answered but they draw attention away from the proper center of the librarian's interest: the bibliography and its use.

2. The files of data generated for a particular application, once in computer-usable form, suggest uses other than those considered when the files were designed. Again the individual responsible for the application, as owner of the file, finds himself drawn into other activities -- either by his own interests or by pressure which others are able to apply.

These experiences bear witness to the fact that the boundaries of particular applications are not at all clear and distinct. The capabilities and devices required are not unique to the particular application. Neither are the data without value in other applications. These realizations have been a prime mover in the development of large-scale, generalized systems such as the Remote File Management System [1] being developed at the University of Texas at Austin and the Time Shared Data Management System [2] being developed at the System Development Corporation. Although these and several other efforts [3], [4] are achieving an exciting level of generality with respect to the data they handle, many difficulties remain on the capability side of these systems: generalized capabilities are lacking in efficiency; systemic capabilities, once



developed, are difficult to shape to the wishes and requirements of particular users; most capabilities, but especially efficient and reliable ones, are expensive to implement and require considerable lead time.

The cure for these problems lies somewhere in that vast area known as "computer software design and development." Realizing the extent of the commitment which software implementation carries with it, we decided to invest some time and thought in efforts to avoid at least some of the difficulties which are taking their toll in current efforts. What we see after making the investment is not a cure-all, but it is encouraging. However, before continuing that aspect of the discussion, allow me to briefly describe the setting in which the work is being done, mention the background upon which it is drawing, and give credit to our present sponsors

This work is being done in the Semiotic Systems Group of the Life Sciences Research Department of TRACOR Incorporated, Austin, Texas. Semiotics, the theory of signs and how they are used, is finding application in the development of machines able to use signs for communication and control. Informally, a mechanical/electronic device with capabilities that enable it to perceive, recognize, and understand signs within a particular frame of reference is a "semiotic system." Such a system may be specialized to the signs of natural language or to signs within some other environment, for example, that of an electronic sensing device.



6500 TRACOR LANE, AUSTIN, TEXAS 78721

Within this context our general objective is the development of theories and techniques adequate for the design and implementation of a semiotic system. In this work we are drawing in various ways upon earlier work done at the Linguistics Research Center of the University of Texas at Austin and the Linguistic Research Project of The RAND Corporation. At present the work is sponsored by the Office of Naval Research, Information Systems Branch and by TRACOR, Incorporated. It has also received support from the U. S. Army Electronics Command, Ft. Monmouth.

This discussion revolves around one component of the proposed system, the component we call the item store. However, to clarify the role and function of the item store, Section 2 briefly describes the point of view we are taking in the design of the system and Section 3 describes the complex of stores within which the item store operates. Sections 4 and 5 deal specifically with the item store, its contents and the capabilities being designed for interacting with it. In Section 6 we turn from design to implementation, describing the techniques we propose to use in implementing the system, including the operations which activate the item store.

2. Overview of the Approach

Picture, if you will, a particular information handling system as an entity separate and apart from its environment, distinguished by boundaries which are expressed in terms of capabilities and data. The system, though distinct from its environment, is able to act upon it, to act in response to it, and to act without stimulus from or effect within its environment.

The diagram in Fig. 1 distinguishes three kinds of information in such a system. Symbolized capabilities are the ground from which systemic activity springs. Data are the objects which enter into and emerge from that activity. The capabilities of the system are event-like in character, whereas data have thing-like qualities.

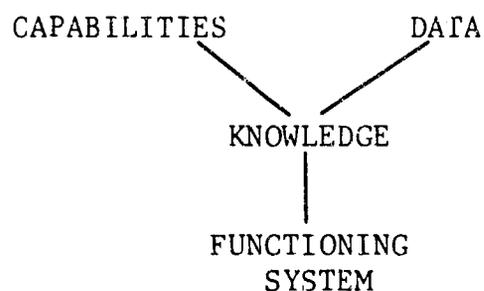


Fig. 1 Systemic Information



However, the system can do only what it knows how to do. It can act or react only when it knows which of its capabilities to use and to what data it should apply the selected capabilities. Therefore, the knowledge of the system, the third type of information, is viewed as an appropriate combination of capabilities and data. Capabilities, in and of themselves, are not committed to any specific data. Similarly, data are susceptible to use with more than one capability. Each apart from the other is incomplete. Only in combination do they form a basis for systemic activity.

The knowledge of the system -- that is, capabilities in combination with data -- is symbolized apart from both types of components. This separation will make it possible to alter the knowledge of the system without altering capabilities or data. This is an important advantage since much of the shaping and adjusting required to tailor a system to a particular environment requires little or no change in capabilities or data, only in the relationship in which they stand.

Capabilities, data, and knowledge then are the three distinct types of information symbolized. "Item" is simply a term used to refer to a unit of information -- particularly in storage -- when it is not necessary to indicate what type of information it is.

Units of each type of information are stored within the system. Therefore, in the next section we consider the



6500 TRACOR LANE, AUSTIN, TEXAS 78721

complex of stores which serves as a repository for the information. During periods of system activity, units of each type of information enter into the activity, each fulfilling its particular role. In Section 6 we return to the subject of systemic activity.

3. The Complex of Stores

The five storage facilities shown in Fig. 2 form the system's complex of stores. Each of the five components, connected as shown, is able to receive items into the store, retain items as long as they are useful, retrieve items for use within system activities, and remove items from the store when their usefulness is at an end.

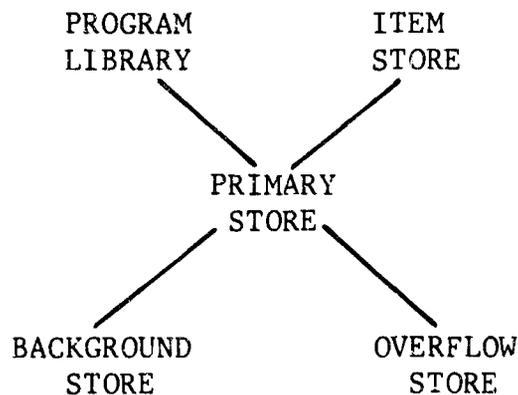


Fig. 2 Components of the Complex of Stores

Although each of the five stores carries out the same basic operations, two important operational differences categorize them. A formal store places highest priority upon retaining information for a relatively long period of time. Therefore, a formal store, like a library or archive, uses a detailed format into which all incoming information is put; items retrieved from the store are, of course, available in this same format. The



format used formalizes the concept of a unit of information, dictates how each unit is identified, and expresses relationships among the units in the store.

In contrast to the formal stores, operational stores place greater importance upon receiving, retrieving, and removing items. That is to say, an operational store is oriented to the use of its contents. The format of its contents varies, each variation reflecting the requirements of a particular use. Generally, the format is quite simple, although elaborately formatted tables and lists are also used. Fig. 3 distinguishes the formal from the operational stores in the complex of stores.

FORMAL STORES:

PROGRAM	ITEM
LIBRARY	STORE

OPERATIONAL STORES:

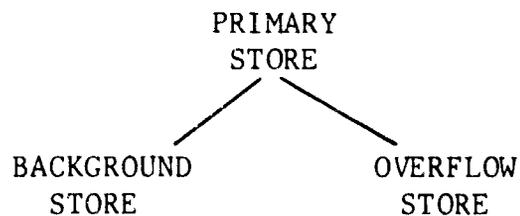


Fig. 3 Formal/Operational Stores

A second important distinction among stores is reflected in the terms primary and secondary. The primary store is directly connected to the control/processing unit of the computer. Therefore, its contents -- units of capability, data, and knowledge -- enter into both sides of system activity -- the instructions side as well as the data side.

The four non-primary stores of the complex are secondary stores. A secondary store is connected only to the processing unit and its contents are therefore restricted to entering system activity on the data side. The principal activity into which the contents of secondary stores enter is that of moving items between the particular secondary store and the primary store.

With these distinctions in mind, we can now describe in more detail the role of each store in the complex, the connections among the components, and the connections between the complex and its environment. Fig. 4 presents some of this detail graphically.

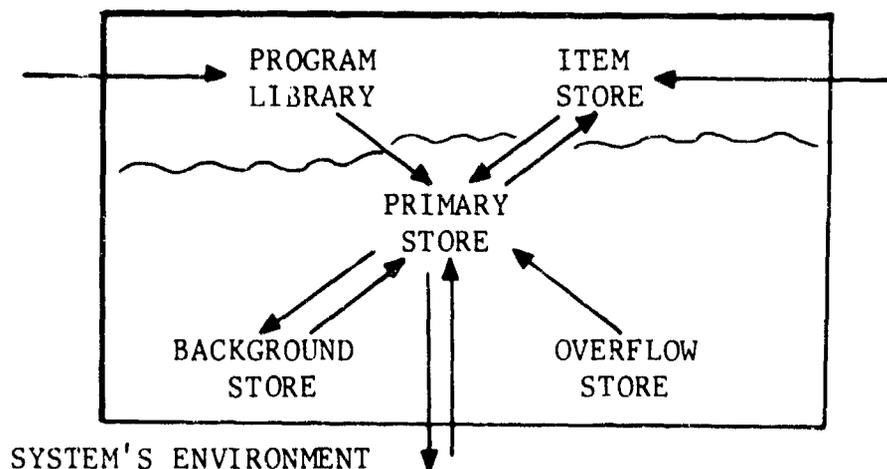


Fig. 4 The Complex Stores



6500 TRACOR LANE, AUSTIN, TEXAS 78721

The primary store is the operational hub of the complex. Within the primary store capabilities and data are bound together, forming units of operational knowledge. These units of knowledge enter the computer's processing and control units, extending system activity in appropriate ways. Since the primary store cannot retain the whole of the system's knowledge, items must be secured as required and retained until the space they occupy is required for items of higher priority.

In addition to connections with each of the secondary stores, the primary store is also the principal interface with the system's environment. Information from the environment enters the system through the processing unit and the primary store. Information passing to the environment follows the same path in reverse.

The background and overflow stores augment the capacity of the primary store. The background store contains units of capability and data, both in operational form. Items in the background store are those which, for the moment, are not required in the primary store; however when a need arises they can be returned to the primary store with little delay.

The overflow store serves a similar, yet distinct, function. Units of data in the primary store reside in compartments which have a fixed capacity. When a particular operational unit of data exceeds the capacity of its compartment,

part of the unit is transferred to the overflow store. A partial unit can be returned to the primary store when needed by exchanging the locations of the two partial units. This exchange or swap is represented by the two-headed arrow in Fig. 4.

From these descriptions it should be clear that both the background and the overflow store are extensions of the primary store made necessary only by limited capacity in the primary store. However, from a user's point-of-view, the three form a logical unity and should be regarded as a single store with the characteristics of the primary store.

The role of the program library in the complex of stores is quite like that of the program library in the typical computer executive system. A program to be entered into the library is written in a source language -- Fortran, Algol, assembly language -- and associated with a name that distinguishes it from other programs in the library. From this source-language form of the program, a load-for-execution form is produced. Typically, both forms are retained in the library.

Each program in the program library is the symbolic form of a system capability, that, under proper circumstances, can be translated into system activity. Adding new programs to the library is operationally outside the system. This is represented in Fig. 4 by the arrow from the environment into the program library.



In principle, the item store is the repository for all formally stored information in the system. However, as just discussed, capabilities symbolized as programs are retained in a separate program library, primarily to take advantage of a vast body of existing capabilities for managing program libraries. However, all other formally stored information resides in the item store. This includes extensive files of systemic information, such as systemic knowledge as defined earlier. Since system and user files are stored in precisely the same format, capabilities and knowledge developed for system files can also be used with user files and vice versa.

In most instances information enters the item store from the primary store. However, as indicated in Fig. 4, the store can receive information directly from its environment. This implies that the information was collected and formatted by an entity external to the system and incorporated into the store in much the same way a new version of, or supplements to, the program library will be made.

Various types of devices can host the item store -- drums, disks, magnetic tapes. While the operational efficiency will vary from type to type, the operational characteristics of the store are independent of device type.

4. Item Files -- Contents of the Item Store

Given the role which the item store must fill, we now turn to consider the design of this storage facility. That design is based upon a body of conventions that we call the organizational form. The organizational form formats the contents of the item store and shapes all system capabilities related to that store. These capabilities, based as they are on the organizational form, are independent of the information stored and its logical structure. The organizational form, although flexible enough to handle the variety of information that must be stored, also allows operational efficiency. In this section we briefly sketch the organizational form and illustrate its flexibility; this material was covered in more detail in an earlier report [5]. Section 5 continues this discussion of the organizational form with a survey of the operations based upon it.

Item files form the contents of the item store. Each file is a set of component classes; these classes determine the contents of the file. Inter-relationships among the classes define each file's structure. Members of the file's classes comprise its records. Each record of the file has a structural pattern that is constrained by but not identical to the structural pattern for the file itself.

Each file in the item store carries a name which distinguishes it from every other file in the store. All trans-

actions with the contents of a file require the use of the file's name and are constrained by the file's boundaries. That is to say, no transaction within a particular file can automatically extend into a second file. The end-of-file of the first will terminate the transaction.

We have chosen the tree as the structural basis for item files and their records. The tree, as an organizing principle, offers a middle ground between lists and more general graphs. Fig. 5 is a representation of a typical tree and will serve as the basis for a brief review of terminology associated with trees.

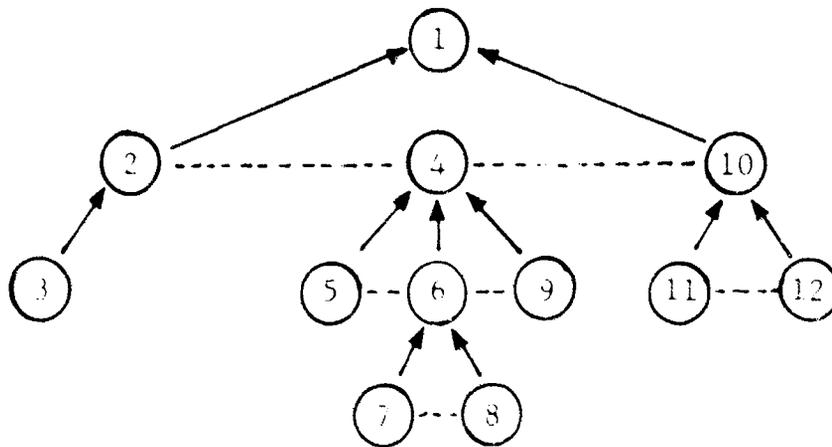


Fig. 5 A Tree

A tree's components are nodes and connections between pairs of nodes. Each node, except one, has a primary connection to one other node, its parent. In Fig. 5, encircled numbers



represent nodes, and arrows represent connections between the nodes. The only node without a parent is the tree's root node. Level is a measure of distance from the root node to another of the tree's nodes. One unit is counted for each node encountered, including the nodes at each terminus. Therefore, the root node is on level one, nodes for which it is parent are on level two, etc.

Although a node has only one parent, any number of nodes can have a particular parent node. Such nodes are its offspring and are always on the next lower (higher numbered) level. For example, node 4 is the parent of 6, and nodes 7 and 8 are the latter's offspring. Nodes with offspring are non-terminal nodes; those with none are terminal. Nodes which are offspring of a single parent are related to each other as sibling nodes.

Each non-root node is connected to the root of its tree by a single set of connections. That set of connections is the path between that node and the root. All nodes along a path are ancestors of the node at its lower terminus, thereby including a node's parent among its ancestors. For example, both 1 and 2 are ancestors of node 3. Similarly, each node for which a particular node is an ancestor, is a descendant of that node; a node's offspring are included among its descendants. To illustrate, nodes 5 through 9 are the descendants of node 4 and of these 5, 6, and 9 are its offspring.

Each node of a tree occupies a position distinct from the position of every other node. Node position can be described in any of several ways, but should not be confused with inter-node relationships. Each node of a tree stands in a definite relationship with each other node of the same tree. Parent, offspring, sibling are examples of such relationships but many other more distant relationships also exist.

Based on this concept of a tree, we define the structure of a file as a single tree of item classes. One class corresponds to each node of the tree and a particular file can, in principle, have any number of classes. The inter-node connections represent the pattern of class relationships as they structure the file. Fig. 6 is an example of a file with nine classes.

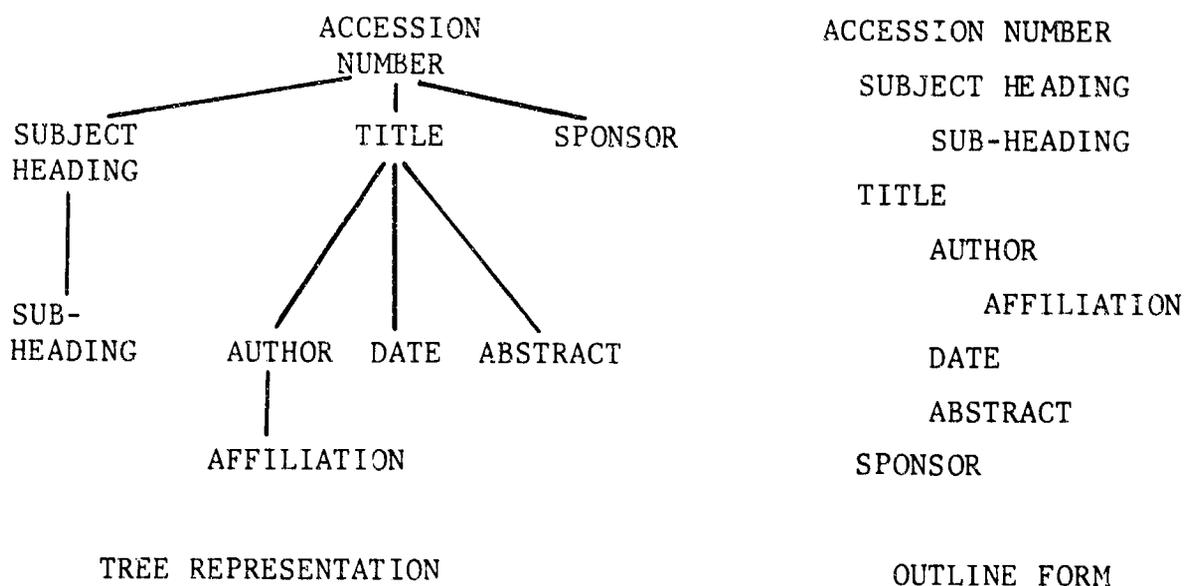


Fig. 6 - A Bibliographic File

This file of bibliographic material is structured with accession numbers at the root. All other information is positioned relative to that. Fig. 6 also shows the same file in the more familiar "outline" or "indented" format. The two forms of representation are equivalent. Fig. 7 is a second example of a file. This file contains text, perhaps the contents of a book on your library shelf. It has but six classes and its structure is quite different from that of the bibliography.

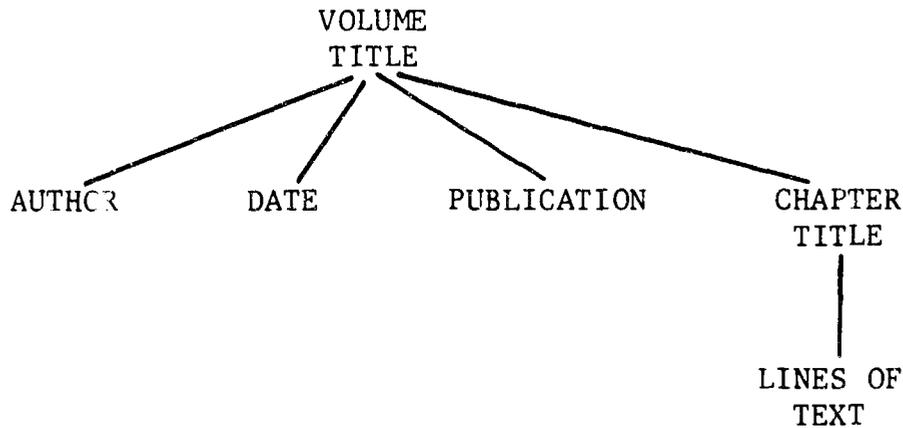


Fig. 7 - A File of Textual Material

With these two examples pointing the way, Fig. 8 further illustrates structural variations which the organizational form allows. In these three separate files the representation of the classes has been simplified to focus attention on the structure. Just as there is, in principle, no limit to the number of classes in a file, so there is also no limit to the number of levels or to the number of sibling classes in a particular cluster.

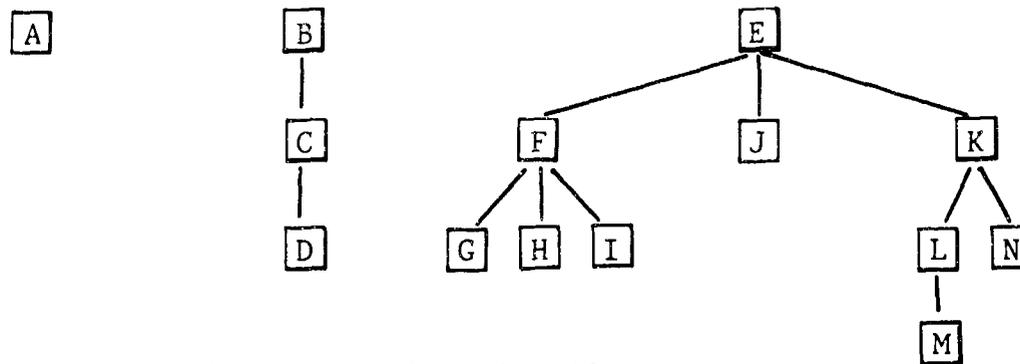


Fig. 8 -- Variations in File Structure

Representing a file as a tree of classes gives a general picture of its contents and its structure, a picture that is useful both as a mental image and in documenting the file. However, an item file, in reality, expresses relationships among individual items and, therefore, involves considerably more detail than is recorded in schematic diagrams of the type we have considered. These details revolve around the nature of an item, the composition of item classes, and the constraints which a file's inter-class relationships place upon the item components of its records.

The item is the basic unit of a file's contents. Each item has one value, a single passage of encoded information. A value is a string of binary bits, unrestricted in length, format, encoding conventions, and meaning. Some examples of item values are. an integer in base two representation; an integer in base ten; a passage of natural language text, encoded as a sequence of fixed-length characters or bytes; a set of independent two-position switches; an instruction, that is, a description of one element of capability. Each value is accompanied by a measure of

its length; a zero-length value, called a "null" value, is therefore quite acceptable.

Every item belongs to an "item class." An item class is a cluster of item "attributes" and a set of items -- "elements" of the class -- to which the attributes apply. Class attributes include such information as the position of the class within the file, the encoding type for values of class elements, and the name of an algorithm by which the relative order of any two elements of the class can be determined.

The attribute clusters for the classes of a file form the file's "map." A file's map contains all of the information required by the system as it manages and processes the file's records. Fig. 9 shows that a map can be represented as a tree, the shape of which exactly matches the tree representing the file itself. Each cluster of attributes corresponds to one node of the map tree. In Fig. 9 capital letters without boxes represent the clusters of attributes at the nodes.

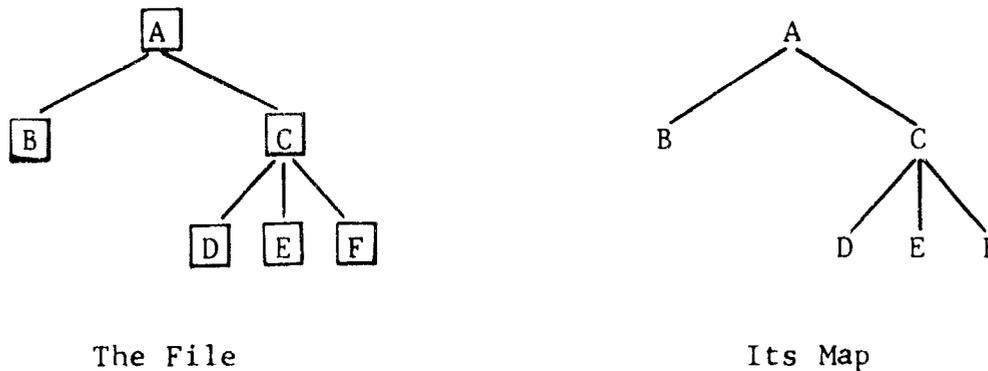


Fig. 9 - A File and Its Map

Arrays of class elements form the records of a file. Fig. 10 shows the map of a file and four of its records. Each record forms a tree with a class element at each node. The structural pattern of the first record exactly matches that of the map. Although the pattern of a record tree is constrained by the file's map tree, the patterns do not necessarily match. Two kinds of difference are allowed: (1) an element of a class may be omitted from a record, in which case all elements of descendent classes are also absent; (2) two or more class-sibling elements can occur as the offspring of a single parent. An example of the first type occurs in the second record; it contains no member of class D and none of class F. The third record, a single element of class A, is another example of this first difference. Examples of the second type occur in record two -- it contains two b and two e elements -- and in record four -- it has two c and two f elements.

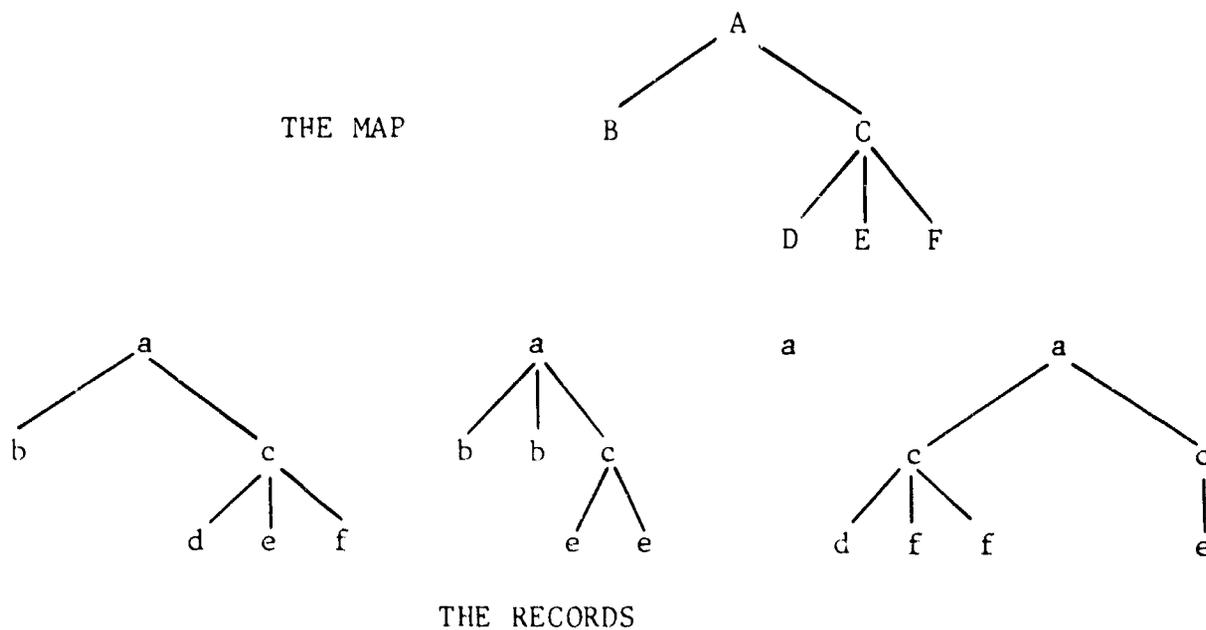


Fig. 10 - The Map and Records of a File



5. Item Store Operations

The operations associated with any storage facility put information into it, get information previously stored there, and remove information from it. The fact that the item store's principal function, as a formal store, is to retain information does not render these operations unimportant. For unless a store -- even a formal store -- can be used efficiently, it is of little value.

The item store will contain system files with which users are not directly concerned, as well as information deposited there by various users. The pattern of interactions, therefore, will typically consist of several independent, concurrent streams of transactions, each arising within a particular activity centered in the primary store. Each stream will be managed by a single control mechanism called a portal. A portal contains information which gives continuity to a series of transactions.

Transactions within a particular stream will always be restricted to a stated portion of the store's contents. That portion to which the transactions have access is the item store context for the activity. A context is either a complete file or a stated sub-set of the records of a file. (Provisions are also being made for user-access restrictions and user-priority rights, but these matters are not included in this discussion.)



The services rendered by a portal differ markedly for some transactions -- for example, the GET as compared with the PUT command. For other operations the required portal services are quite similar -- for example, the GET and GET NEXT ITEM commands. Therefore, we have defined a set of operational modes, each of which encompasses a set of operations requiring similar portal services.

Item store operations form the connection between the item store and the primary store. They move items between the two stores and make preparations for such movements. Preparations include setting a portal for a particular mode of interaction, positioning a storage device to give access to the relevant area of the item store, and reformatting the items moved or to be moved.

The purpose of this report dictates against a review of the individual operations for each mode of interaction. Therefore, the following paragraphs simply describe each of the eight modes of operation for the item store.

Reading. Reading is the basic operational mode. Every portal, regardless of its mode setting, includes the facilities required to read. Each other mode makes appropriate additions to these facilities.

The operations of the reading mode locate items previously recorded in the item store and transcribe them into the primary store, depositing them at a specified location. Item

retrieval proceeds most efficiently when items are retrieved in the order stored. However, each item has a positional address, making it possible to retrieve items in any order whatsoever.

In the read mode, items are always retrieved under control of the map according to which the context being read was written. Since reading an item does not alter the occurrence of it in the item store, a particular item can be re-read any number of times. Reading an item, in effect, produces a second copy of it in the primary store.

Writing. Operations of the writing mode put items from the primary store into a context of the item store. At the outset, that context contains no items. Each item written becomes the next item of the context. That is, items are stored in the same order in which they are received for storage. The map of the file being written is always used to check the stream of items entering the store. These checks block the entry of illegal item sequences. When the context being written is a new file, the map can of course have whatever structure is required. When the context is an extension of a file, the controlling map must also be the map of the file being extended.

Rewriting. Operations of the rewriting mode are combinations of reading and writing operations. Under the control of a single map, items within a specified context are read, while, concurrently, items are written within the same context.

Separate position registers are associated with the reading and writing activities so that items read can be rejected and not rewritten. Similarly, items not read, but rather supplied by the user, can be written into the context. A parameter of the writing activity specifies whether the records being read are preserved or whether they are replaced by the records written. If they are preserved, the records written are regarded as a new version of the context.

Version Reading. Typically, a context of the item store is a set of consecutive records. However, as we have just seen, the rewriting mode can be used to write an alternate version of a file or portion of it, producing thereby parallel sets of records. Operations of the version reading mode are equipped to handle files in which such alternate versions exist. They function within a context of records which need not be consecutive records of the file.

Revision Reading. In normal reading, the records of a file are read according to the map used when the file was written. The operations of the revision mode provide for reading according to a different map, the activity proceeding just as if the revised map had been in control when the file was written. Differences between the actual map of the file and the map governing the revision reading activity are restricted to those that require no reordering of the item stream.



6500 TRACOR LANE AUSTIN, TEXAS 78721

Summary Reading. During normal reading, each item is processed as a distinct entity, separate from all other items in the file. The operations of the summary mode treat sets of related items as a single item. When such a package of items is again written into a file of the store, the package returns to its normal status as a set of related items.

Merge Reading. Reading in the normal mode secures copies of items from within a single context. The operations of the merge mode make it possible to read items from two or more distinct contexts. The various streams of items are controlled with a single map as they are merged to form a single item stream. In this way two or more parts of a single file -- or two or more independent files -- can be read as if they were a single item array.

Sort Writing. Writing in the normal mode records items within one prescribed context in the store. Sort writing makes it possible to distribute items within a single stream into two or more contexts. The selection of a context for a particular item can, for example, minimize the number of sequence breaks within the contexts, sequence breaks being defined by a prescribed algorithm.



6. Implementation

We now turn from the item store -- the design of a thing-like storage facility and operations which manage the thing-like entities in it -- to the matter of implementing event-like capabilities. The question that must be answered is: how, in wisdom, do we proceed with implementing an operable system equipped to act in many -- in fact an unlimited number of -- different ways?

It must be clearly understood that the capabilities for managing the contents of the item store are but one part of the body of capabilities required. Other capabilities alluded to in the previous sections are the following:

- a. Move items between other pairs of stores in the complex.
- b. Manage the contents of the three operational stores.
- c. Bind capabilities to data, thereby forming operable knowledge.
- d. Acquire data from the environment of the system.
- e. Deliver data into the environment in a form and format appropriate to the circumstances.

Some other important capabilities that have not even been alluded to are the following:



6500 TRACOR LANE, AUSTIN, TEXAS 78721

- a. Capabilities for various kinds of analyses and decision making such as automatic classification, pattern recognition, and grammatical parsing.
- b. Capabilities in that area often called data management, especially such file operations as generating, updating, extracting, transforming (the structure), sorting, and merging.
- c. Capabilities for querying a file or data base, such as those offered in the Remote File Management System.

To understate the situation, the system will contain a vast accumulation of capabilities. Each must be symbolized in a form that facilitates clear and accurate documentation, offers a maximum amount of protection against obsolescence, and encourages continuing capability extension and evolution by means of new combinations of existing capabilities as well as by implementing new capabilities directly.

One implementation strategy we are finding useful is the one illustrated by the item store -- capabilities are based, to the extent possible, on a body of general conventions which cover a wide range of individual cases. Consequently, item store operations are based entirely on the organizational form, remaining independent of particular file structures.

Two other strategies, briefly mentioned earlier, are also influencing our approach to implementation. The first is

to preserve in the implementation the distinction between capabilities and knowledge, each of which is symbolized apart from the other and both apart from data. The second strategy is to implement capabilities and knowledge in modules or units, each unit distinct and detached from all others. The remaining paragraphs of this section describe how we are applying these two strategies.

The course we are following rests upon two basic concepts -- the concept of an elemental program and the concept of a program's environment. An elemental program is a unit of capability, symbolized in a form which the system can translate into action. Each elemental program is distinct from and independent of other elemental programs; neither does an elemental program contain commitments to any specific data.

An elemental program is a program in the ordinary sense in that it is written in a suitable programming language, has a name to distinguish it from other programs, and occupies a position in the program library from which it is retrieved when needed.

Some other conventions, not uncommon for computer programs in general, further characterize these units of capability.



- a. Each elemental program has a single entry point.
- b. Each execution of an elemental program eventually reaches a final termination point. Its execution in a particular instance can either be successful or a failure; but never is its execution unending nor can its execution be interrupted "temporarily" without, at some later time, continuing to completion.
- c. Each execution of an elemental program, when complete, leaves the program with the potential for action that it possessed when that execution began. This is a way of saying that a program's potential for action never changes from execution to execution. In addition, some elemental programs will be re-entrant in the sense that two or more parallel executions of the program can proceed without any one of them interfering with the others.

An elemental program is different from most other computer programs in that it contains no explicit references to external entities. This does not mean that each is completely self-sufficient. In fact, most elemental programs, in and of themselves, are incomplete. What we are doing, however, is documenting rather than filling the incompletions.

The incompletions of an elemental program represent the ways in which the program depends upon external entities. In other words, these incompletions define the program's requirements upon its environment. An appropriate environment of an elemental program is any set of entities which, collectively, satisfy these requirements. Fig. 11 is a representation of an elemental program and its environment.

In this representation we see that two types of incompletions, reflecting the two basic types of information, can occur: incompletions related to data and incompletions related to capabilities. Therefore, a typical environment consists of two types of components -- blocks of data in operational format and other programs. Actually, a distinction is made between operational blocks of data and parameters, which are in essence small data blocks. But the difference between the two is of little consequence for our present purposes.

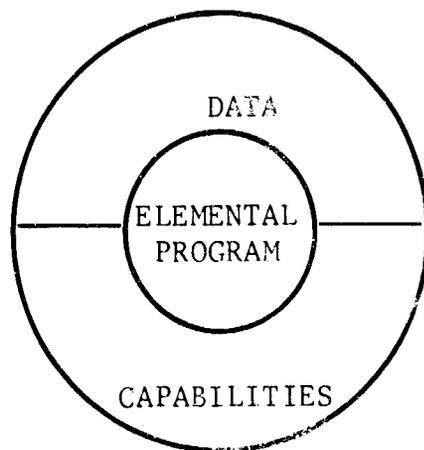


Fig. 11 - An Elemental Program and an Environment

An elemental program can be executed only when associated with an appropriate environment. Typically, more than one environment can be constructed that meets a particular program's requirements. This point is illustrated in Fig. 12 where a single elemental program appears in two different environments -- a and b.

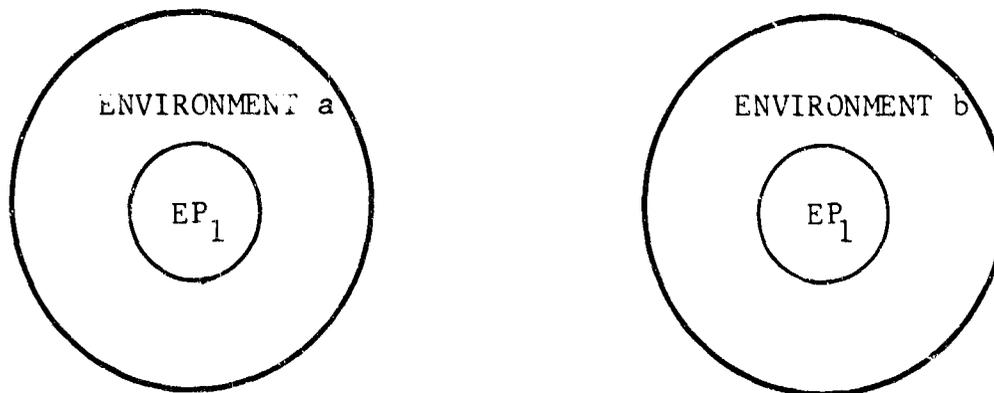


Fig. 12 - An Elemental Program and Multiple Environments

An elemental program cannot alter its own activity potential nor can that potential be altered by an entity external to the elemental program. However, the performances that a program delivers in successive executions need not be identical. Variations reflect changes in the program's environment. Therefore, a program can change its own performance -- or that of another program -- by altering the composition or contents of its environment. If a program's range of potential performances can be charted on a bad-to-good scale, the program's performance can "improve" through successive executions.

As stated earlier, the association between an elemental program and a particular environment is documented as a

separate entity. That entity, a process prescription, is a unit of systemic knowledge, for it binds together capabilities and data. The requirements of an elemental program upon its environment are documented in a two-level tree called the program's schema. The schemata of two elemental programs, EP₁ and EP₂, are shown in Fig. 13. Although we have not decided in detail how to document environmental requirements, the kind of information that documentation must contain is quite clear. For example EP₁'s capability requirement, cr₁, must allow or reject the use of any particular program in that position of EP₁'s environment. Similarly data requirements must allow or reject the use of particular blocks of data.

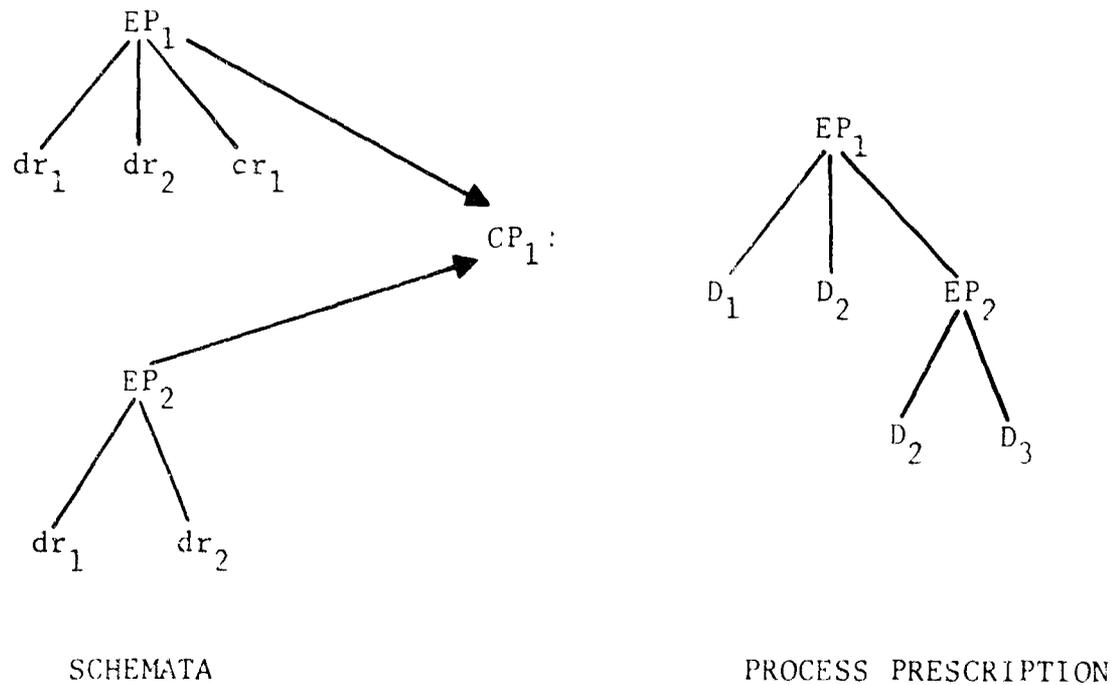
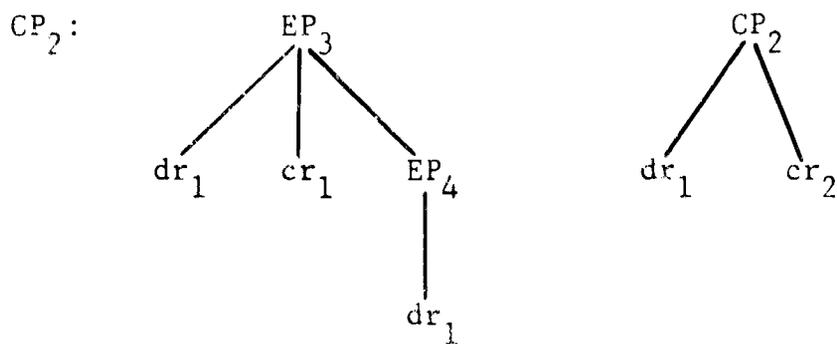


Fig. 13 - A Constructed Program

Assuming that EP_2 satisfies the requirements set forth by cr_1 and that $D_1, D_2,$ and $D_3,$ satisfy the various data requirements, the constructed program, $CP_1,$ is well formed. The components of a constructed program are not generally collected and assembled into a single unity. Instead, the components are simply named and their inter-relationships are expressed in the process prescription. A constructed program is executed by interpreting its process prescription.

In Fig. 13 the constructed program CP_1 is completely specified in that each program component has a completely specified environment. However, completeness is not required of constructed programs. The program $CP_2,$ shown in Fig. 14, has, for example, two incompletions. One is a data requirement, $dr_1,$ imposed by each of the two elemental programs. The other is a capability requirement, $cr_1.$



An Incomplete
Process Prescription

Schema of a
Constructed Program

Fig. 14 - A Constructed Program and Its Schema

The incompletenesses of constructed programs, just as in the case of elemental programs, are documented in program schemata. And so the schema of the constructed program, CP_2 , has two reference slots as shown.

Carrying this process one step farther we see how a constructed program is used as a component within a constructed program. Fig. 15 provides the example. We see the schema of CP_2 we were just discussing. Its form is in no way different from the schema of an elemental program. Therefore, the conditions which allow the formation of CP_3 are precisely the same as the conditions shown in Fig. 13 when both components were elemental programs. There is, in fact, no reason why a constructed program, which satisfies the requirement cr_1 , could not have been used in place of the elemental program EP_5 .

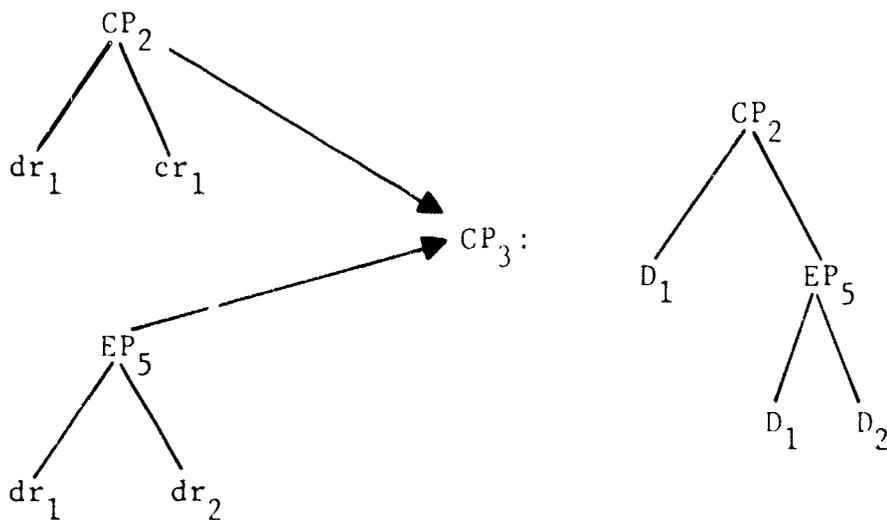


Fig. 15 - A Constructed Program as a Component of a Constructed Program

This approach to the construction of executable programs has several advantages. Many of the components comprising a constructed program can be used simultaneously in several different programs. Since a process prescription simply names the components of the constructed program it defines, each component is secured only when and if needed as the program is executed.

It is important to understand: there is nothing fixed (unchangeable) about the boundaries of elemental programs. An array of elemental programs can at any time be compiled into a single elemental program.

Of at least equal importance is the fact that each program can itself be used as a component of other programs. This will facilitate the formation of higher and higher level capabilities. For this, process prescriptions offer a brevity and an orderliness that promotes accuracy, a flexibility that allows adjustment to the circumstances and requirements of particular applications, and a rule of formation that will allow automating program production in particular situations.

To elaborate that last point, the system can, without difficulty, be equipped to compose new process prescriptions. Such prescriptions, if formed in response to needs which the system itself recognizes, would serve to guide the system's reaction to such needs. Of course, the key to this matter is equipping the system to sense significant needs and to develop appropriate responses to them.

Process prescriptions will be stored as a file in the item store. Fig. 16 suggests the contents and a structure for such a file. Handling process prescriptions in this way -- as data -- keeps them independent of the operational capabilities and characteristics of particular hardware/software systems. This independence is especially important for systems that require a long-term, evolutionary development, as well as for systems that have a long-term life-expectancy or wide-spread applicability.

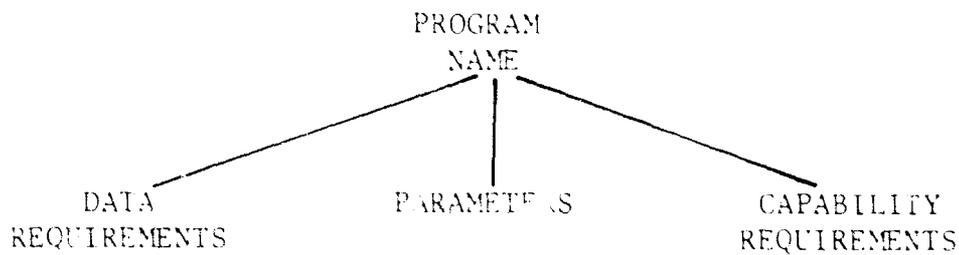


Fig. 16 - A File of Process Prescriptions

7. Conclusion

In Section 2 we distinguished three types of information -- data, capabilities, and knowledge. Each type of information makes a distinctly different contribution within the system; only in combination do we have operable systemic knowledge. Other sections described the conventions we are developing for symbolizing, storing, and using units of information of each type.

The item store, discussed at some length, provides a long-term repository for any information, but in particular for data and systemic knowledge. We believe that in the item store we are making adequate provision for permanently useful files of data in computer form. Such permanent files -- not necessarily unchanging, but permanent nevertheless -- are here to stay and will play a central role in many information handling applications. A facility devoted to their maintenance, development, and retention will be essential.

This does not deny the necessity for and importance of operational formats for data -- formats which reflect a particular use to which the data is put. The Remote File Management System serves as a good example of this point: data, as stored in the RFMS tables, reflect the purpose of the RFMS system which is use of the data in a particular way. In contrast, the purpose which the item store format reflects is preservation of the data.

Finally, if files of permanently useful data exist, it follows that bodies of permanently useful capabilities are also required. But with the techniques now used to implement capabilities, we find that all capabilities put into operable form are, after a longer or shorter period of time, discarded in favor of newly implemented -- but not necessarily new -- capabilities. Furthermore, systems with the capability range we are considering -- not unlike the range found in some of the systems under development and on the drawing boards today -- are proving to be extremely difficult to implement using conventional programming techniques. Elemental programs and process prescriptions will, we believe, make capabilities, once implemented, more permanent and more useful.



REFERENCES

1. Dale, Alfred G., The Remote File Management System: Some Academic Applications, Department of Computer Sciences, The University of Texas at Austin, October, 1968.
2. Vorhaus, A. H., and R. D. Wills, The Time-Shared Data Management System: A New Approach to Data Management, SP-2447, System Development Corporation, Santa Monica, California, February 1967.
3. Reliability Central Automatic Data Processing Subsystem, Vol. I and Vol. II, Auerbach Corporation, September 1966, AD-489-666 and AD-489-667.
4. The Galton Institute's Imprint, The Galton Institute, Beverly Hills, California, October 1968.
5. Ziehe, T. W., An Organizational Form for Item Management, TRACOR Report 67-1111-U, TRACOR, Incorporated, February 1968.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D		
<i>(Security classification of title, body of abstract, and indexing annotation must be entered when the overall report is classified)</i>		
1. ORIGINATING ACTIVITY (Corporate author) TRACOR, Inc. 6500 Tracor Lane Austin, Texas 78731		2a. REPORT SECURITY CLASSIFICATION Unclassified
2b. GROUP		
3. REPORT TITLE An Item Store: Its Design and Implementation		
4. DESCRIPTIVE NOTES (Type of report and inclusive dates)		
5. AUTHOR(S) (First name, middle initial, last name) Theodore W. Ziehe		
6. REPORT DATE December 1968	7a. TOTAL NO. OF PAGES 48	7b. NO. OF REFS 5
8a. CONTRACT OR GRANT NO. N00014-67-C-0396	8b. ORIGINATOR'S REPORT NUMBER(S) TRACOR 68-1360-U	
b. PROJECT NO NR 048-239	8c. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
c.	d.	
10. DISTRIBUTION STATEMENT Distribution of the document is unlimited.		
11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Office of Naval Research Washington, D. C. 20360	
13. ABSTRACT Three types of information -- data, capabilities in symbolic form, and knowledge -- are distinguished in an informal manner. The role of each within an information system is sketched as the basis for a discussion of the item store. The item store is a general-purpose formatted store which will serve as a repository for files of inter-related items. The tree serves as the organizing principle for files within the store. The operational modes for the store are described as are the techniques being used to implement these operational modes.		

DD FORM 1473
1 NOV 68

Unclassified

Security Classification

Unclassified

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Information Management Information Storage and Retrieval Item Management Data Management Trees Data Structures Semiotics						

Unclassified

Security Classification