AUTOMATIC PROGRAMMING
Internal Memo 6

October 17, 1972

MAPL
A Language for Describing Models of the World

by

William A. Martin
and
Rand Krumland

Introduction

Automatic Programming Internal Memo 4 introduced a relational model called the World of Business. In Protosystem I the user describes his problem to the system by what amounts to instantiation of selected relations in this World of Business model. This model has been refined, modified, and elaborated. The new version is described here. After some remarks as to purpose, we describe the language in which the model is written. Next, we give the more universal parts of the model itself. In a following memo we will give details of the model pertaining to business systems and describe the A&T Supermarket case by instantiating our model.

Three Cultures - Three Languages

Before giving the details of the language, it might be well to say a few words about what it is intended to accomplish. More thoughts on this also appear in Memo 2. A planned future memo will hopefully provide further elaboration.

Protosystem I encompasses the knowledge of three related, yet often separate groups of individuals.

1) Practicing managers
2) Computer systems specialists
3) Management scientists.

Each of these groups focuses on rather different aspects of a problem such as sales forecasting. Their view-point, coupled with their diverse backgrounds, rather sharply alters the terms in which each group would describe the same situation. For example, in the A&T Supermarket case we find the following management oriented description.

1) "Sales vary considerably by day, with peak sales occurring during the weekend specials."

In the OS/360 Inventory Control Program Description Manual we find:

2) "Annual usage = $\frac{\text{MPUQD x PUNOPDYR}}{\text{MPUPD}}$

where:

MPUPD = Number of periods for which demand history exists;

MPUQD = Total demand quantity for number of periods (MPUPD);

PUNOPDYR = Number of periods per year for parts usage fields.

PUNOPDYR -- parts usage number of periods per year -- is defined as a constant in the IC$OPIN macro."

Finally, in Multiproduct Production Scheduling for Style Goods with Limited Capacity, Forecast Revisions and Terminal Delivery, a Sloan School of Management Working Paper, we find:

3) "Let X_j represent the forecast of total seasonal demand for a product with the forecast made at the beginning of period j , then the ratios of successive forecasts are (X_{j+1}/X_j). Also let X_{N+1} represent actual total demand for the product. Define

(1) $Z_j = (X_{j+1}/X_j)$, $j = 1,\ldots,N$ (time periods).

Then it is assumed that the variable Z_j has the following distribution:

$$Z_j \sim f_{LN}(Z_j/\mu_j, \sigma_j) = \frac{1}{\sqrt{2\pi}\sigma_j Z_j} e^{-(\log Z_j - \mu_j)^2/2\sigma_j^2} \qquad j=1,\ldots,N$$

with Z_j independent of Z_k for $j \neq k$."

It is interesting to contrast these three fragments. A most important difference is that the management description is both less complete and less precise than the other two. In the A&T case we are not told how sales are computed, or even what is being sold to whom. The implication is that any reasonable method of computing sales will yield a quantity with considerable daily variation. We are also told that sales vary "considerably." Again, the implication is that any precise method of computing variability will yield a which will cause us to employ the types of business practices useful when sales vary "considerably". Next, note that we are assumed to be familiar with the concept of a "weekend special." In fact the A&T case description assumes that we understand on the order of 1,000 concepts and 2,000 facts about them.

It is probably reasonable to regard the language of practicing managers as suited to the description of heuristics which guide more precise procedures in structuring highly complex situations and organizations. If Protosystem I is going to automate the solution of the A&T case, it must be able to utilize the information in the A&T case description. We must then ask how this information is to be communicated to the machine, and how it is to be represented there. For several reasons many people believe that the mode of communication must be English. Typed English can of course be read into the machine but the machine cannot currently do anything with information represented in that way. Efforts are under way to translate English into some representation which the machine can understand. Since the languages of computer systems and applied mathematics are already understood by computers to a useful degree, we must consider them.

The language of computer systems specialists is intended for more precise description than that of practicing managers, but less precise than that of mathematical management scientists. This is not to say that a precise statement cannot be made in a programming language, but rather that a very precise statement such as that in the third fragment is made more naturally in the applied mathematics language. Another difference is that the computer systems language does not have very many concepts "built in" as does the language of practicing managers. Computer systems language is also intended for making complete statements which tell computers exactly what to do.

If we attempt to state the information in the A&T case description in computer systems language, the following problems will arise.

a) The additional precision of computer systems language will require us to make a more accurate statement than we can justify, or else construct a very elaborate structure to avoid making an accurate statement. For example, how shall we state that sales vary "considerably" using only the information contained in that phrase. The important point is that nothing is to be gained by making a more accurate or complex statement! (Actually, quite a bit is lost!) For the fact that sales vary "considerably" is precisely the summary type of information needed when trying to decide what business practices to use. (Such information can't be used when more accuracy is really required, since the additional accuracy is only apparent.)

b) Without the concept of "weekend special" in the computer systems language, it will be very difficult to express this concept without loss of information which will leave the system vulnerable to error through overly rigid interpretation. For example, what about "three day weekends." To "explain" every concept in the input description will increase its

length to an unmanageable size, and seriously burden the user with a task he is not skilled in.

c) The problem of man/machine communication will be greatly simplified if the machine has a representation of the problem which parallels that of the man. A study of a book like Human Problem Solving (A. Newell and H. Simon) doesn't lead one to feel that managers "think like system programs" and are then forced by English to make their statements as they do. Thus any attempt to go directly from a manager to a systems or management science language is going to be very difficult.

Although computer system's language is bad for our purposes, the language of applied mathematics would be even worse.

It is our opinion that in fact an appropriate language can be found for representing the A&T Supermarket case. A study of recent work, such as that of T. Winograd for example, leads one to feel that, as long as a global purpose is known, information can be represented in a form much more amenable to computation than English, yet without the disadvantages listed above. What we would like to show is that this can be done in a practical applications area like distribution systems using only thousands of concepts and relations rather than tens of thousands. Several points lead us to feel that this may be the case

a) We have designed our language so that the user may add concepts of his own. Thus it is not necessary for the system to know "everything," but only enough so that the user won't mind telling it the rest and will be able to succeed in doing so.

b) "Applications Questionnaires" are now on the market which purport to construct a tailored small business information system with the yes-no answers to less than 2,000 questions.

c) The A&T Supermarket case is 3,000-4,000 words long. It seems to be a typical length.

d) The specification of a data structure is generally smaller than the specification of the programs which manipulate it. We are not trying to write all business programs; just a data structure for them to use. Once we have this and a skeleton set of programs we are under way

It should be noted that a present day computer can handle thousands of concepts and relations in main memory fully indexed (giving the associative retrieval effect) with no problem. The CONNIVER programming language will help us do this.

We have represented the A&T Supermarket in our new language as a first step

In a following memo we will try to show how this representation is adequate to guide routines solving the A&T case. Next, we must show how computers can construct the representation given here through a dialogue with managers. (Once this can be done in English, our true objective will be reached.) Finally, we must hope that the size of our model will approach a limit as new cases are added.

The Modeling Language

We now describe the language in which the World of Business (W-O-B) model is written. (This amounts to a theory of "fill in the blanks.") We call it MAPL because it is about mappings and their properties. The data types in the language are:

a) Objects,
b) Concepts,
c) Relations, and
d) Sequences.

The relationships which can be expressed in this language are:

a) Set inclusion,
b) Functional dependence, and
c) Naming.

Each of these is now described in turn. There is nothing unique or unusual here, but we must say what constructs we choose to use and our notation for them.

Objects:

An object is a particular thing, such as a particular person, or a particular apple. An object can have a proper name such as %W-A-MARTIN or %GOOD-APPLE-1. We will start the names of objects (which are not concepts) in the W-O-B with a % so we can distinguish them from names not known to the W-O-B.

If %GOOD-APPLE-1 and %BAD-APPLE-2 are two objects, then the set of %GOOD-APPLE-1 and %BAD-APPLE-2 is also an object. This set might be called %MIXED-APPLE-SET. It is important to distinguish between an object which is a set, and the objects in it. A typical member of a set will be referred to by the set name with % replaced with $. $MIXED-APPLE-SET would be a typical element of %MIXED-APPLE-SET.

Concepts:

A concept is a special kind of object. It is a predicate which is true or false for any ordered tuple of objects. The name of a concept will be spelled

starting with a #. For example, we might have the concept #APPLE, which we would define to yield true for every object which is an apple and false for all other objects. #APPLE would yield true for %GOOD-APPLE-1 and %BAD-APPLE-2, but false for %MIXED-APPLE-SET, since %MIXED-APPLE-SET is a set of apples, not an apple. We say that #APPLE specifies the set of objects, %APPLE, for which it is true. We call this set %APPLE by convention, replacing the # with %. %GOOD-APPLE-1 and %BAD-APPLE-2 are two of the elements of the set %APPLE; %MIXED-APPLE-SET is a subset of the set %APPLE. We will use %APPLE to mean the set of all apples, and $APPLE to mean a typical member of this set.

We express the fact that a concept is true for a specific tuple of objects by adding the concept to the left of the tuple. For example, we might express the fact that a specific apple, %GOOD-APPLE-1 is tasty with the tuple

(#TASTY %GOOD-APPLE-1).

How would we express the fact that all apples are tasty? We don't want to write a tuple for each apple. We do this by writing the typical element, $APPLE, of the set %APPLE, in the ordered tuple, then the ordered tuple is understood to hold when each object which the typical element stands for is substituted for it. All we need, then, to say that all apples are tasty is,

(#TASTY $APPLE).

This expression can be modified by indexing, as will be explained later.

Relations:

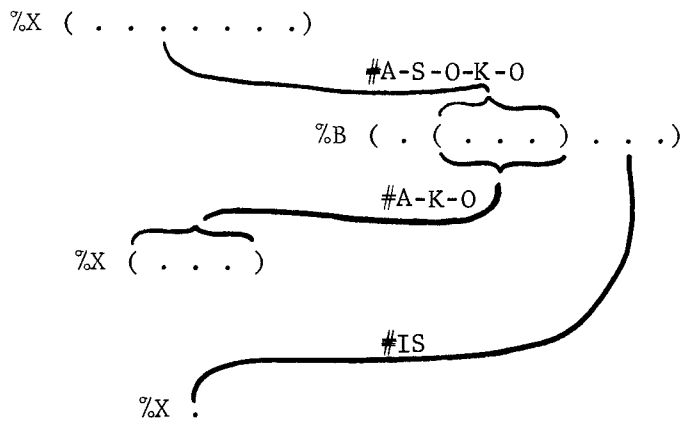
A relation is a set of ordered-tuples for which a specific concept is true. Given sets $S_1, S_2, \ldots, S_n$ (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples each of which has its first element from S_1, its second element from S_2, and so on. We shall refer to S_j as the j-th domain of R. A relation is an object. This means that relations can participate in tuples. In the W-O-B relations have a number of properties to be described later. These properties are given by using the relations in tuples.

Sequences:

A sequence is a set whose elements are ordered.

Set inclusion:

There are three concepts used for expressing set inclusion which occur constantly in the W-O-B. These are "is" (#IS), "a kind of" (#A-K-O) and a "a set of kinds of" (#A-S-O-K-O). These are shown diagrammatically in Figure 1. We express the fact that an object is in the set specified by a concept by using #IS. For example, we could write,



(. . .) denotes a set.

#IS ≡ %X is an element of %B.

#A-K-O ≡ %X is a subset of %B.

#A-S-O-K-O ≡ %X is a set of subsets of %B.

Figure 1.

(#IS %GOOD-APPLE-1 #APPLE),

which says that %GOOD-APPLE-1 is an element of %APPLE.

The use of "a set of kinds of" is slightly more complex. This is used to indicate that one concept specifies sets whose elements are selected from the set of objects specified by a second concept. For example, to indicate that a gaggle is a set of geese we would write

(#A-S-O-K-O #GAGGLE #GOOSE),

which says that every element of %GAGGLE is a subset of %GOOSE.

The final concept, #A-K-O, is used to indicate that the objects specified by one concept are a subset of those specified by another. For example, to indicate that all apples are fruit we could write

(#A-K-O #APPLE #FRUIT),

which says that %APPLE is a subset of %FRUIT.

Figure 2 shows a hierarchy of concepts suggested by Raphael which is similar to structures that can be formed using our set inclusion relationships. Raphael contends that a complete model of this type "is a representation for the class of all possible events. ...it represents the computer's knowledge of the world." [1, p.51] However, Raphael rejects this organization with the following statement:

> "Unfortunately the model described has several drawbacks which prevent its use in a general semantic information retrieval system. It is extremely difficult to construct a useful model, of the form described, for a significant amount of information; writing a program which automatically would add information to the model is out of the question. The "$\in$" and "$\subset$" relations are not sufficient to describe many useful groupings of nouns, but the introduction of a few additional relations would confuse the structural organization of the model and force the crosslink statements to be much more complicated. The verb groupings, in order to be useful, must be carefully selected according to the ill-defined restriction that the resulting configuration allow simple and useful crosslink statements. This may not always be possible and certainly becomes more difficult as the number of relations considered increases." [1, p. 53]

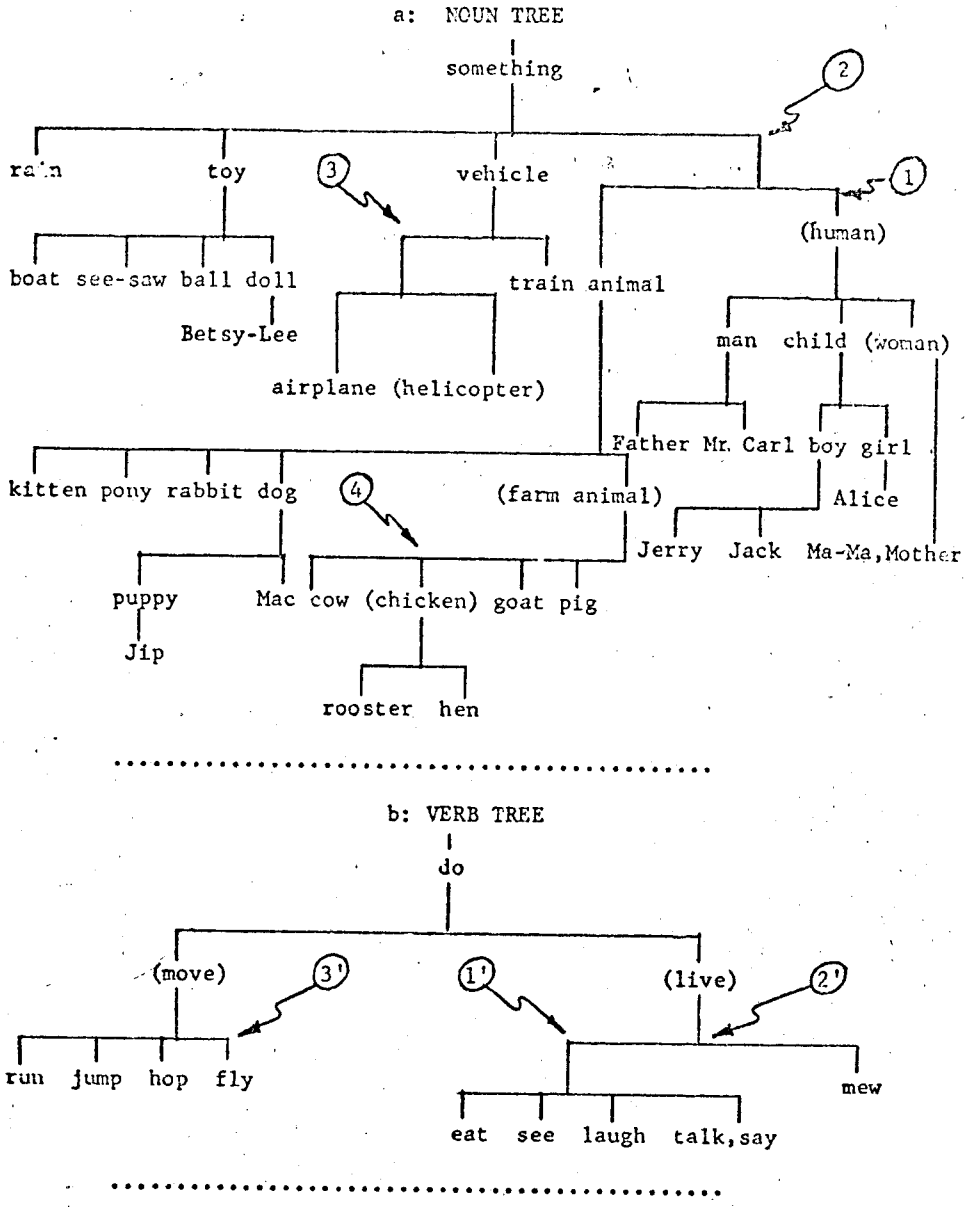


Figure 2 A Word Association Model

It seems to us that as a representation of knowledge this structure is not fundamentally wrong, and, if properly extended, would prove to be very useful in certain situations. The problems Raphael noted stemmed from the particular application he had for the structure and the particular form of it that he used. There are several reasons why we will have more success with it than he believed possible.

First, Raphael implies that such a structure should be used as an absolute test for allowable noun-verb, or more generally object-action, relationships. It should instead serve as more of an index to possible relationships or sets of relationships. Once candidate relationships are obtained, procedures can then be applied to rule out uninteresting cases or to somehow order the relationships on a scale of degree of interest, desirability, or applicability. (In CONNIVER we can implement mechanisms for accessing relation candidates as special fetch functions. Thus, "DT-FETCH" might operate on relation patterns containing objects, concepts, typical elements, and unknown -- ?X -- variables. The typical elements will match any object which #IS in the set containing the typical element or any typical element of a set which is #A-K-O the set containing the typical element to be matched.)

Second, in Raphael's structure a node is restricted to be #A-K-O one other node. Instead of this strict tree structure, we will allow a node to be #A-K-O many nodes. This creates a lattice under set inclusion and will allow us to make any useful grouping of objects. It is not clear whether any particular grouping will correspond to a noun or a verb, but the grouping should make sense in the discipline (or "world") for which it is formed. Thus, in an area like management, it is very common to form concepts from concatenated nouns -- e.g., product group or responsibility center -- and these need to be easily described and related in our structure.

Third, by regulating our model to a domain that is in some ways limitable we will be able to construct a useful model for that domain that incorporates a significant amount of information that is not unduly complicated. It appears that the W-O-B lattice will be quite flat, so that the number of nodes under any concept won't be very large. Based on this expectation, we will in fact introduce a new node to represent additional information whenever we know more about a set of objects than

a) properties of a typical element;

b) properties of the entire set of objects.

For example, suppose we have the concepts #ENTERPRISE, #SELLS, and #FRUIT, and we find out further that

(#IS %A&T #ENTERPRISE)

Then if we want to indicate that %A&T sells fruit, but not necessarily all kind of fruit, the set %FRUIT must be subseted to that sold by %A&T. We declare

(#A-K-O #A&T-FRUIT #FRUIT)
(#SELLS %A&T #A&T-FRUIT)

The advantage of this scheme is that it makes every set mentioned by the user directly fetchable from the global data base and provides a place to put additional properties learned about the set. A look at the A&T case description indicates that not all that many sets will be formed.

Functional dependence:

The concept "a characteristic of", #A-C-O, is used to declare the existence of a relation for which the left-most element of an ordered tuple is a characteristic of the remaining ones. For example, to declare that any object which is a #FRUIT can have the characteristic #COLOR, we would write,

(#A-C-O #COLOR $FRUIT),

which we define to mean that there exists a <u>mapping</u> from each object specified by #FRUIT to the set of objects specified by #COLOR. Another way to say this is that given any fruit, this mapping exists and may assign the fruit <u>one or more</u> colors. That is, (#A-C-O #COLOR $FRUIT) declares that for each fruit there may be one or more tuples containing it and a color. The tuples are contained in the above mapping. By convention, such a tuple will be written

(W-OF X Y)

where Y is the fruit, X is its color, and W is the concept, #COLOR, which specifies the set into which the mapping is done. For example, given

(#A-C-O #COLOR $FRUIT)
(#IS #RED #COLOR)
(#IS %GOOD-APPLE-1 #APPLE)
(#A-K-O #APPLE #FRUIT)

we could state that %GOOD-APPLE-1 is red with the tuple

(#COLOR-OF #RED %GOOD-APPLE-1)

or that all apples are red with the tuple

(#COLOR-OF #RED $APPLE).

Note that this last tuple declares a relation which is a subrelation of (#A-C-O #COLOR $FRUIT); the subrelation is then #A-K-O the initial relation.

This example gives us an opportunity to review the notions of concept and object. Note that #RED is an object for which the predicate #COLOR is true. #RED is also a concept (concepts are a kind of object) which is true for all red objects. If we state (#A-K-O #MAGENTA #RED) we are stating that the concept #MAGENTA is true for a subset of the objects for which #RED is true. #MAGENTA #IS a #COLOR as well.

As a second example, suppose we want to state that a distance is a characteristic of any two cities. We could state

(#A-C-O #DISTANCE #CITY #CITY)

Now suppose we want to express the fact that an arm can be in relation "part" to a body. This can be done most conveniently by declaring a relation with "a predicate on" (#A-P-O) rather than #A-C-O. (#A-P-O X A_1 ... A_n) states that a concept with values true and false and name X exists on the n-tuple (A_1 ... A_n). For example, to express that an arm can be a part of a body we would state

(#A-P-O #PART-OF $ARM $BODY).

The fact that a predicate is true for a particular n-tuple of objects is stated with (X O_1 ... O_n) where X is the name of the predicate. For example, to state that the predicate #PART-OF exists on an arm and a body we state

(#A-P-O #PART-OF $ARM $BODY).

To state that this predicate is true for both my arms I state

(#PART-OF $BILL-ARMS %BILL-BODY).

To state that it is true for my left arm I state

(#PART-OF %BILL-LEFT-ARM %BILL-BODY).

Note that the #A-P-O and #A-C-O concepts are used to declare relations. Using the naming conventions of the next section we can name the declaration statement; the name is then the name of the relation, and can be used in tuples to give properties to the relation.

For example, we might state (#A-C-O #SUPPLIER $A&T-ITEM) and name this relation %A. If each item is obtained from only one supplier we could state that %A is a one-to-many mapping. If some suppliers supply only one item, we can state (#A-K-O #B #A) and then state that %B is a one-to-one mapping. The reader should now review the difference between #A-C-O and #A-P-O. It should be clear that

(#A-C-O #COLOR $FRUIT)

Is just shorthand for

(#A-P-O #COLOR-OF $COLOR $FRUIT)

and in fact, the first will be represented in the machine as the second.

Naming:

It was already mentioned that any object can have a name. We assign names with the concept #A-N-O. For example, to name the relation (#A-C-O #COLOR $FRUIT), %R1, we would state

(#A-N-O %R1 (#A-C-O #COLOR $FRUIT)).

This will both state (#A-C-O #COLOR $FRUIT) and name it %R1.

Instantiation

The user describes his problem to the system by stating tuples to it. The system will not accept a tuple unless

a) the tuple declares a relation, %B, which is #A-K-O a relation %A which is already known to the system;

b) the tuple is a member of a relation %A which is already known to the system.

If neither a) nor b) holds the system rejects the new tuple as one which it cannot "understand." If a) or b) does hold, the system may still complain about the tuple if it conflicts with facts the system already knows, but the system will be said to feel it understands what was said and not want to accept it.

Remember that relations are themselves objects. Thus in cases a) and b) above, relation A can be thought of as a set specified by a concept which passes only objects which are tuples of a specified form. We will use relations, A, in this way and name them.

For example, if we state

(#A-N-O %A (#A-C-O #COLOR $FRUIT))

then if we state (#A-N-O %B (#COLOR-OF #RED $APPLE)), the system will take it and imply (#A-K-O #B #A). If we state (#A-N-O %B (#COLOR-OF #RED %GOOD-APPLE)) the system will take it and imply (#IS %B #A), since %B is fully instantiated. (The element replaces the one element subrelation.)

Properties of Relations

We indicated in the last section that relations themselves have characteristics. One of the most important characteristics of a relation is its mode (indicated by the concept #MODE). A relation can have any one of the following modes:

a) %CAN-HOLD,

b) %MUST-HOLD,

c) %MUST-NOT-HOLD,

d) %DOES-HOLD, and

e) %DOES-NOT-HOLD.

(Thus, (#A-K-O #MODE $RELATION), (#IS %CAN-HOLD #MODE), etc.) If the mode of a relation is not given it is assumed by default to be %DOES-HOLD.

Properties of Mappings

We have explained how an #A-C-O declaration sets up a mapping. A mapping is also set up by the true values of a predicate on 2-tuples declared with #A-P-O. Sometimes it is useful to know the type of the mapping. Therefore we introduce the concept #MAPPING-TYPE which specifies objects

a) %ONE-TO-ONE

b) %MANY-TO-ONE

c) %ONE-TO-MANY

d) %MANY-TO-MANY

As an example, consider specifying whether a store can belong to more than one division. Given (#A-N-O %R1 (#A-P-O #PART-OF $STORE $DIVISION)), if a store must belong to only one division we state

(#A-N-O %R2 (#MAPPING-TYPE-OF %MANY-TO-ONE %R1))
(#MODE-OF %MUST-HOLD %R2)

A second characteristic of a mapping is its scope, that is, the extent to which it applies to the members of the sets indicated by the elements of the 2-tuples. It will be indicated by the concept #MAPPING-SCOPE which specifies the objects

a) %ONTO-ONTO,
b) %INTO-ONTO,
c) %ONTO-INTO, and
d) %INTO-INTO.

ONTO indicates that the entire set -- that is, all elements of the set,-- participates in the mapping, while INTO indicates that only a subset of elements participate.

A third characteristic is the set of constraints that apply to the domains of a mapping, %R1, with respect to a superior relation which %R1 is #A-K-O. This characteristic will be indicated by the concept #CONSTRAINED which specifies the following objects:

a) %CONSTRAINED-CONSTRAINED,
b) %UNCONSTRAINED-CONSTRAINED,
c) %CONSTRAINED-UNCONSTRAINED, and
d) %UNCONSTRAINED-UNCONSTRAINED.

If the j-th domain of %R is %CONSTRAINED, then the relation superior to %R1 contains no tuple with its j-th element in the j-th domain of %R1 and for some k, the k-th element of that tuple not in the k-th domain of %R1. For example, we might declare (#A-P-O #PERFORMS $JOB-TITLE $ACTIVITY). Later we might refine this by stating (#A-P-O #PERFORMS $BUYER $BUYING). If a buyer can only perform buying and no other activity we can state this by constraining the domain %BUYER.

It is somewhat inconvenient to have to use the naming mechanism in order to specify the mapping properties of relations. Therefore we give a shorthand which can be used on input. We will use the abbreviations

%ONE-TO-ONE	11
%MANY-TO-ONE	M1
%ONE-TO-MANY	1M
%MANY-TO-MANY	MM
%ONTO-ONTO	OO
%INTO-ONTO	IO
%ONTO-INTO	OI
%INTO-INTO	II
%CONSTRAINED-CONSTRAINED	CC
%UNCONSTRAINED-CONSTRAINED	UC
%CONSTRAINED-UNCONSTRAINED	CU
%UNCONSTRAINED-UNCONSTRAINED	UU

and include these abbreviations in the names #A-C-O and #A-P-O. For example, to state that every #FRUIT has a #COLOR, we state (#A-IO-C-O #COLOR $FRUIT). To state that every fruit has one and only one color we state (#A-1M-IO-C-O #COLOR $FRUIT). Inside the machine this will be expanded to

```
(#A-N-O %G1 (#A-P-O #COLOR-OF $COLOR $FRUIT))
(#MAPPING-TYPE-OF %ONE-TO-MANY %G1)
(#MAPPING-SCOPE-OF %INTO-ONTO %G1).
```

#A-IO-C-O and #A-1M-IO-C-O are fairly common and so we give them the alternate, more pneumonic names, #A-R-C-O, "a required characteristic of", and #A-U-R-C-O, "a uniquely valued required characteristic of", respectively.

Note that if some mapping assigns every objects in a set %B to exactly one object in set %A, then, under this mapping, %A can serve as an index to the objects in %B. Each object in %A specifies a set of objects in %B. Each object in %B specifies an object in %A. As a special case we say that %B can serve as an index to itself.

Suppose we declare

(#SELLS $A&T-STORE $A&T-STORE-SKU).

This says that all A&T stores sell all A&T store stock-keeping-units. What we would like to say is that each A&T store sells only the S-K-U's associated with it. We can do this by establishing a mapping between S-K-U's and A&T stores and then stating

(#A-N-O %R1 (#SELLS $A&T-STORE $A&T-STORE-SKU))
(#INDEX-OF %A&T-STORE %R1)

By definition of #INDEX-OF, this restricts the mapping %R1 to contain only tuples (#SELLS X Y) where X and Y both have the same A&T-STORE as index.

Suppose a relation, A, is declared to be indexed by a set, B, but the set has yet to be established as an index of some set,C, which is a domain of relation A. For example, suppose that %A&T-STORE had not been established as an index of %A&T-STORE-SKU when

(#A-N-O %R1 (#SELLS $A&T-STORE $A&T-STORE-SKU))
(#INDEX-OF %A&T-STORE %R1)

is stated. In this case, indexing the relation implies that an indexing of $A&T-STORE-SKU by $A&T-STORE exists. In order to accept this statement, the system must establish that such an indexing is permitted by existing relations.

We will need the notions of an evaluated tuple of objects and an array of such evaluated tuples. If one object names another, then the second is a value of the first. We will make the restriction that an object can have at most one value, but perhaps several names. Suppose we want to describe A&T store sales, we might declare,

(#A-U-R-C-O #SALES $STORE)
(#A-K-O #A&T-STORE-SALES #SALES)
(#A-K-O #A&T-STORE #STORE)
(#A-N-O %R1 (#SALES-OF $A&T-STORE-SALES $A&T-STORE))
(#INDEX-OF %A&T-STORE %R1)
(#A-N-O $A&T-STORE-SALES $DOLLAR-VALUE)
(#A-N-O $A&T-STORE $INTEGER)

R1 is then a relation between each A&T store and its sales. An <u>evaluated tuple</u> is a tuple with the predicate name at the left removed and each of the remaining objects replaced with its value. We could give the sales of A&T stores by giving the evaluated tuples of %R1. To do this we allow statements of the form

(#ARRAY relation-name list-of-evaluated-tuples-which-are-elements-of-the-relation).

For instance, if A&T has 3 stores with sales of $1,000, $1500, and $2,000 respectively, then one evaluated tuple of %R1 would be ($1500 2), and we can give the sales for all the stores by

(#ARRAY %R1 (($1000 1) ($1500 2) ($2000 3))).

General Features of the W-O-B

A complete list of all the relations in the W-O-B will be given in the next memo. However, this memo will make more sense if we introduce the central ideas here.

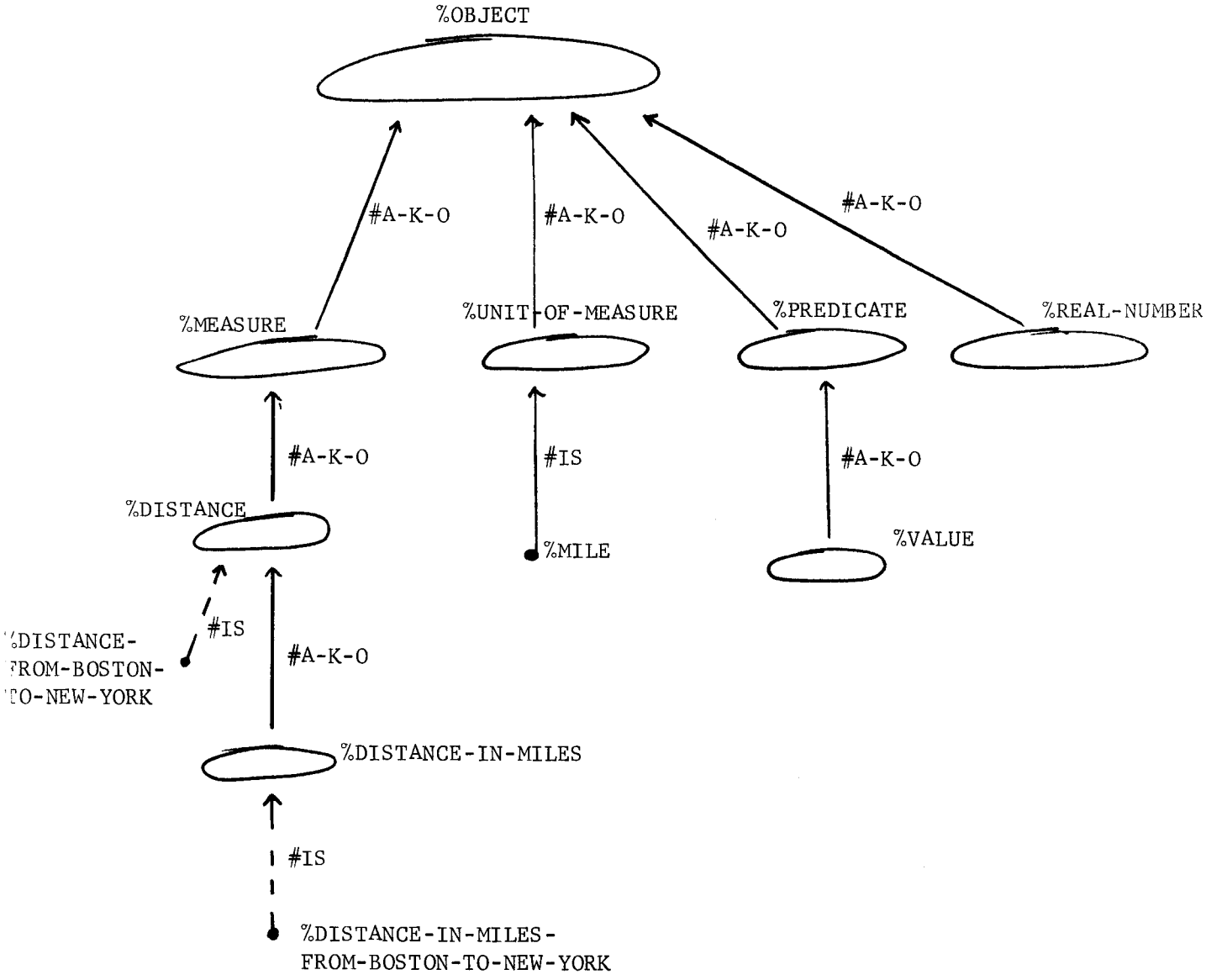
One interesting thing about the W-O-B is that every concept except #OBJECT is #A-K-O at least one other concept.

Thus the concepts form a lattice under set inclusion. As the W-O-B grows it will be interesting to watch the shape of this lattice. Also every object #IS some concept.

At the top of the lattice we find the concepts which have a broader meaning. In this section we will be discussing those which are not particular to business activities.

As a first example, it is instructive to see how we handle distance. Figure 2 shows this part of the W-O-B net. If the user declares that the distance-in-miles-from-Boston-to-New-York is a distance-in-miles, then the system will know that this distance can have a numerical value which is a real number and that the units of this value are miles. If the user declares that the distance-from-Boston-to-New-York is a distance then the system will still know that the distance can have a numerical value which is a real number, but it won't know the units. Note that we are forcing the user to name the distance, not just give its value.

As a second example, it is important to understand how we handle an activity and a data item (activities are explained in Memo 4). This is shown in Figure 3.



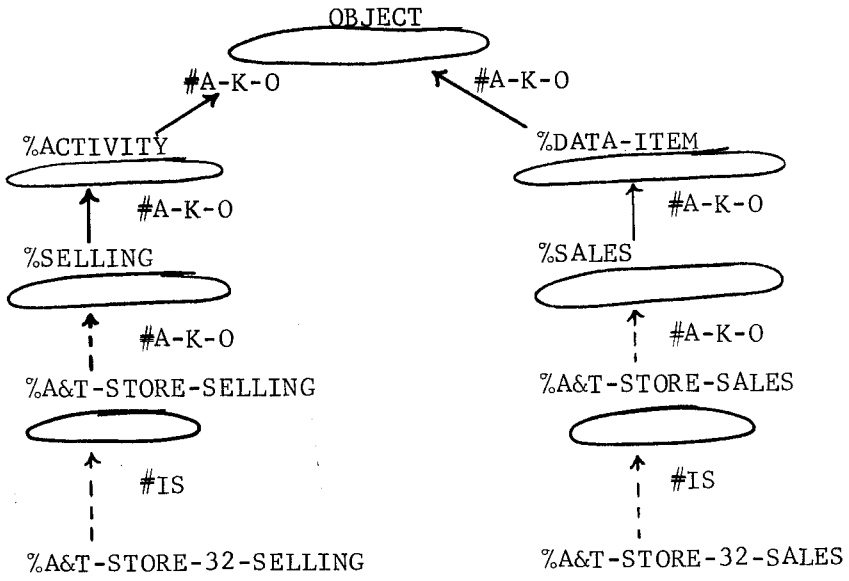
(#A-N-O $MEASURE $REAL-NUMBER)

(#A-C-O #UNIT-OF-MEASURE $MEASURE)

(#UNIT-OF-MEASURE-OF %MILE %DISTANCE-IN-MILES)

(#A-U-R-C-O #DISTANCE $CITY $CITY)

Figure 2.



(#A-P-O #HAS-CONTEXT $SELLING $RESPONSIBILITY-CENTER
$RESPONSIBILITY-CENTER-SET $COMMODITY-SET)
(#A-U-R-P-O #GENERATOR $ACTIVITY $DATA-ITEM)
(#A-U-R-P-O #GENERATOR-OF $SELLING $SALES)

Figure 3.

An activity is the W-O-B name for what is known in Memo 4 as a procedure family. An activity needs to be placed in context. This is done by using any instance of an activity in a #HAS-CONTEXT tuple which links the activity and the parameters which place it in context. For example, given the relations in Figure 3, one could declare #A&T-STORE-SELLING to be the concept for selling done by each A&T-STORE with the relations.

(#A-K-O #A&T-STORE-SELLING #SELLING)
(#A-N-O %R1 (#HAS-CONTEXT $A&T-STORE-SELLING $A&T-STORE %A&T-CUSTOMER
$A&T-STORE-ITEM-SET))
(#INDEX-OF %A&T-STORE %R1)

Note that the user has declared that all stores serve the same customers, but he has left open the possibility of different item sets for each store. If the user wants to define selling at A&T-STORE-32 he could add to the above,

(#IS %A&T-STORE-32-SELLING #A&T-STORE-SELLING)

(#HAS-CONTEXT %A&T-STORE-32-SELLING %A&T-STORE-32 %A&T-CUSTOMERS
%A&T-STORE-32-ITEM-SET)

A relation which does not hold for all time is called an event. Events are of two types, simple events and recurrent events. A simple event happens only once; it has a start, a duration, and an end. A recurrent event happens repeatedly; and each occurrence is essentially a simple event. Note that occurrences can't overlap one another in time. Obviously, a recurrent event is a concept which specifies a set of occurrences. As usual, we can give properties of the entire event, or of a typical occurrence. (Of course we can always subset the occurrences to give more detail.) The occurrences of a recurrent event form a sequence ordered on the time each occurrence starts. It will be common to establish a one-to-one mapping between the occurrences of an event and some other sequence already ordered by time. Objects in this second sequence can then be used to index the first one.

It is quite common for one event to be a generator of another, or in some other way responsible for its occurrence. If A is generated by B, then the set of objects used to index the occurrences of B can also be used to index A, since we assume that A can occur only if B occurs (A need not occur).

An activity is a common type of recurring event. An activity can generate unique data items which are therefore also recurring events. In fact we require that every data item be generated by some activity. The context and time index of the data item can then be determined from its generating activity.

An enterprise is made up of a heirarchy of responsibility centers. Every activity is the responsibility of some responsibility center. We may want to mention a data item without giving its associated activity. If so, we can associate it with a responsibility center which has responsibility for its activity. It is a data item of that responsibility center and all of those above it.

Every recurrent event must be either random or scheduled. If it is scheduled then its schedule must be given. A schedule is a sequence of objects which are moments in time or a rule for generating such a sequence. The typical member of this sequence can be used when referring to a scheduled occurrence. A random event must either have its occurrences numbered, or its occurrences must be indexed by those of an event it depends on. If a relation, X, is numbered, the

numbers form the objects of the set %X-EVENT-NUMBER.

Clearly the values of any data item form a sequence in time. Thus it is understood that when the values are numerical time series analysis can be done on them.

BIBLIOGRAPHY

1) M. Minsky, ed., Semantic Information Processing, MIT Press (1968).

2) D.V. McDermott and G.J. Sussman, The CONNIVER Reference Manual, MIT A/I Lab., Memo 259 (May, 1972).

3) P. Winston, Learning Structural Descriptions From Examples, MIT Project MAC TR-76, (Sept., 1970).

4) T. Winograd, Procedures as a Representation for Data in a Computer Program for Understanding Natural Language, MIT A/I Lab TR-17, (Feb., 1971).

5) J. Earley, Relational Level Data Structures for Programming Languages, Computer Science Department, University of California, Berkeley, (March, 1972).

6) W. Hausman and R. Peterson, Multiproduct Production Scheduling for Style Goods with Limited Capacity, Forecast Revisions and Terminal Delivery, MIT Sloan School Working Paper 522-71, (April, 1971).

7) O/S 360 Inventory Control Program Description Manual, IBM document SH20-0776-0, (May, 1970).

8) A. Newell and H. Simon, Human Problem Solving, Prentice Hall (1972).

9) R. Balzer, Automatic Programming, Institute Technical Memorandum, Univ. of Southern California, Information Sciences Institute (Sept. 1972).