MAC TR-118

AN ABSTRACT MODEL OF A RESEARCH INSTITUTE: SIMPLE AUTOMATIC PROGRAMMING APPROACH

Victor Briabrin

March 1974

Work reported herein was conducted at Project MAC, Massachusetts Institute of Technology as a part of research supported by the scientific exchange program between the National Academy of Sciences of the U.S.A. and the Academy of Sciences of the U.S.S.R.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

CAMBRIDGE MASSACHUSETTS 02139

An Abstract Model of a Research Institute: Simple Automatic Programming Approach

ABSTRACT

A problem of knowledge representation is considered in terms of designing a model for a simple sociological structure. A version of the access language is proposed which is based on three kinds of expressions acceptable by the system - constructors, specificators and requests. In addition, some topics concerned with model implementation and extension are discussed.

Acknowledgments

Papers and discussions produced by the Automatic Programming Group contributed to the main ideas expressed in this paper. Some problems directly or indirectly related to the subject of the paper were discussed with Bill Martin and Ed Fredkin to whom I am also most grateful for the general encouragement and assistance in my work at Project MAC. Carl Hewitt and Peter Bishop were my immediate colleagues and advisors in studying PLANNER and other relative material. With Sudhindra Umarji I discussed many different aspects of Automatic Programming.
Dottie Scanlon, Barbara Kohl and all members of Project MAC headquarters gave me permanent and invaluable assistance in the use of literature, equipment, computer time and in organizing everything which makes life easier. My wife Galya did a tremendous amount of work typing and editing the text, debugging sample programs and exhorting me to finish this memo.

1. Motivation

A model has been created with two purposes in mind: a) an attempt to simulate a simple sociological structure can show the underlying reasons for building an Automatic Programming System based on the scientific model; b) a sample implementation in PLANNER or CONNIVER can give a taste of the applicability of these programming systems to model construction and utilization.

During the development of the system an attempt has been made to analyze the author's own actions and decisions in order to draw some conclusions on:

a) which definitions are more universal than others;
b) what type of metalanguage is the most convenient for describing the syntax of the external language;
c) how to restrict the model so that it could be implemented in a short period of time and still contain enough knowledge to be extended without big changes in the basic structures;
d) what implementation language is the most appropriate for this kind of system.

Some results of this analysis are reflected directly or indirectly in the following text.

2. Contents of the Model

A model of an abstract research institute is considered from the following points of view:

a) administrative (structure of divisions);

b) scientific (association of divisions with research projects);

c) personnel (characteristics of the people and their assignments to divisions and research projects);

d) household (building maintenance, equipment, energy supply);

e) financial (sponsoring, money distribution, salaries etc.).

The goal of constructing a model is to create some formal scheme for describing a scientific establishment and to give the user (scientific or administrative manager, a guest, a student, or some snoopy person) an instrument which can probably help to analyze the structure of the institute, predict possible results of rearrangements, plan future developments, etc.

Three stages can be outlined in the process of creating and utilizing the system.

First stage - developing the frame which is supposed to be filled up with specific information concerning an actual establishment. In terms of LISP-like languages the frame is in fact a group of predefined functions manipulating relations which are to be inserted into the appropriate Data Base.

Second stage - filling the system with the necessary information so that the Data Base will represent the model of the specific establishment. Both the first and the second stages serve to create a model, but in the first stage information provided to the system is mostly procedural whereas in the second stage it is mostly relational.

Third stage - utilization of the system, i.e. posing questions, giving commands, changing the model, etc. In fact, pure utilization is impossible without permanent model readjustment: the whole cycle of model creation and utilization can, therefore, be described as "define the model, fill it with information, use it, repeat".

3. Language of Access

According to these three stages in model specification and utilization, there are three major types of statements acceptable by the system. They are:

-constructors

-specificators

-requests.

Statements of type "constructor" are used to define the internal structure of the model. Syntax of constructor depends on the programming system in which the model is implemented. In LISP-like language it has a common form of function definition. Function name becomes a name of releation to be used later in specificator or request. Function arguments become relation arguments to be substituted by the names of specific objects.

Statements of type "specificator" in LISP-like language environment correspond to function calls. Normally they have the following syntax:

(relation-name arg1 arg2 ...) (1)

where each argument can be either an atomic-object name or a list of atomic-object-names. Atomic-object-name can represent a set of homo-

geneous objects (in this case it can be thought of as a common name for typical members of a set) or a particular object (in this case it is usually the proper name of an object). Syntax of an atomic-object-name is as follows:

<atomic-object-name>::= <compound-name> [.<compound-name>]...

<compound-name>::= <common-name> | <proper-name> |
<common-name> / <proper-name>

<common-name>::= <chain-of-words>

<proper-name>::= <chain-of-words>

<chain-of-words>::= <word> [-{<word> | <number>}]...

Examples:

1) COMPUTER-SCIENCE-DEPARTMENT.PROJECT/ALPHA.RESEARCH-GROUP/A1

2) EMPLOYEE/JOHN-SMITH-JR

3) SALARY-1ØØØØ-DOLLARS-PER-YEAR

Compound names constituting an atomic-object-name have to be ordered in accordance with the semantic subordination imposed by the subordinate-type relation (see next paragraph). Thus in the first example it is supposed that some relations had asserted already that "Research Group A1 is a part of Project ALPHA which in its turn is a part of Computer Science Department".

The combination of common-name and proper-name is useful if any ambiguity arises from utilizing only one or the other. Chain-of-words is the basic syntactic element to create atomic-object-names (as well as relation names). It is defined syntactically almost the same way as an identifier in most programming languages but looks a little more natural when defined through the utilization of the notions "word" and "number".

The effect of a specificator accepted by the system is that some additional substructure appears in the Data Base or some changes take place in the existing structure.

Statements of the type "request" are similar to specificators in syntax but the result of their evaluation is mostly informative rather than creative. Request functions normally analyze the existing Data Base structure and create temporary substructures which are used to develop appropriate answers to given requests.

4. Specificators

Specificator is the major type of expression used to insert certain knowledge into the Data Base. It is useful to classify possible specificators by introducing the notion of "specificator type". Actually, specificator type defines a manner of connecting objects within the Data Base structure. For the simple model of a Research Institute the following specificator types seem to be adequate:

a) subordination

b) correlation

c) condition

Some examples are considered below. The first group of specificators is of type "subordination". The syntax of each specificator is represented by a relation name and list of formal parameters (formal parameters are given in lower case). Comments and simple examples show the manner of their utilization. The following specificators were found to be most useful:

1) (CONSIST-OF container constituent constituent-specification)
where

<constituent-specification>::=<list-of-specific-constituents> |
<expectable-number-of-constituents> | <empty>

CONSIST-OF specificator introduces two objects called "container" and "constituent" and an optional "constituent-specification". "Constituent-specification" can be either in the form of "list-of-specific-constituents", or it can give the "expectable-number-of-constituents", or it is empty.

Example:

If we want to express the fact "A laboratory consists of three sectors, namely A1, A2 and A3", then a CONSIST-OF specificator can be used to represent this fact to the system in the form:

(CONSIST-OF LABORATORY SECTOR (A1 A2 A3))

or in the form:

(CONSIST-OF LABORATORY SECTOR 3)

or in the form:

(CONSIST-OF LABORATORY (SECTOR/A1 SECTOR/A2 SECTOR/A3))

In the last example constituent is represented by the list of specific objects rather than by a common name.

2) (WORK-ON actor activity activity-specification)

WORK-ON specificator describes an object, called "actor", which is supposed to work on some "activity". Similar to CONSIST-OF specificator this type of relation also can have an optional "list-of-specific-activities" or "expectable-number-of-activities" as a form of "activity-specification".

Example:

The fact "Sector A1 is working on Project STAR" can be expressed in the form:

(WORK-ON SECTOR/A1 PROJECT (STAR))

or in the form:

(WORK-ON SECTOR/A1 PROJECT/STAR)

3) (HAS-ATTRIBUTE item attribute attribute-specification)

This type of specificator permits representing a fact that an object called "item" has some "attributes", which can be represented as in previous examples by an optional "list-of-specific-attributes" or an "expectable-number-of-attributes".

Example:

Expression "A person is characterized by his experience, educational degree, salary and creativeness" can be represented in the form of relation:

(HAS-ATTRIBUTE PERSON CHARACTERISTIC
(EXPERIENCE EDUCATIONAL-DEGREE SALARY CREATIVENESS))

4) (MEASURED-in property unit-of-measure unit-specification)

This type of specificator introduces the way of measuring for some property. Unit-specification can represent either the range of units-of-measure or a list of specific names for units-of-measure.

Example:

(MEASURED-IN EXPERIENCE DEGREE-OF-EXPERIENCE
(NOVICE DILETTANTE PROFESSIONAL EXPERT))

5) (SUPERVISE chief minor minor-specification)

SUPERVISE specificator is the most clear and direct representative of subordination-type relation.

Example: (SUPERVISE BOSS EMPLOYEE 12)

6) (DURATION activity time time-specification)

DURATION specificator allows us to represent an activity which is dependent on a specific amount of time.

Example: (DURATION SPONSORING-PERIOD MONTHS 24)

It can be noticed now that all relations described so far have similar formats. General syntax of this format can be defined as follows:

(subordination-name superior subordinate subordinate-specification) (2)

where

<subordination-name>::= <chain-of-words>

<superior>::= <atomic-object-name>

<subordinate>::= <atomic-object-name> | (<atomic-object-name>...)

<subordinate-specification>::= <list-of-specific-subordinates> |
<expectable-number-of-subordinates> | <empty>

Semantics of relation between superior and subordinates can be represented in Data Base by the simple structure of Fig.1. We shall call this type of structure a "relation-branch".

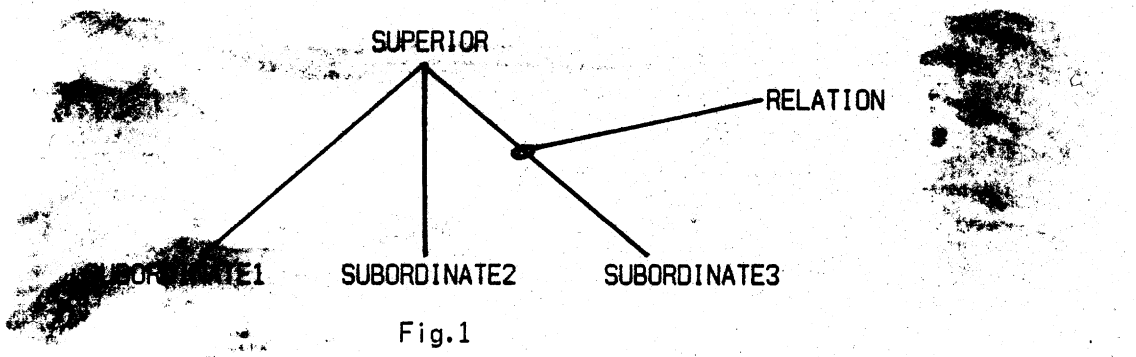

Fig.1

It is clear that the same object can participate as superior or subordinate in different relations (probably of the same type but on different levels). Therefore the main Data Base structure looks like a multilevel and multilayer tree, constructed from relation-branches.

It was found that subordination-type specificator (which is somewhat similar to set inclusion) is not sufficient for some elements of model representation. "Correlation" is another type of specificator used to complement the subordination.

General syntax of correlation is the following:

(correlation-name heading list-of-tuples-of-correlating-objects) (3)

Example:

If we want to express the fact: "Group B1 is occupying rooms 5Ø1, 5Ø2, 517, whereas Group B2 is occupying rooms 2Ø7, 213, 217" then the following expression can be used:

(OCCUPATION GROUP ROOM (B1 (5Ø1 5Ø2 517))
(B2 (2Ø7 213 217)))

It is clear from this example that one-to-one correspondence is supported between elements of the heading (in this case "GROUP" and "ROOM") and elements of each tuple; or, in other words, each element

of the heading corresponds to a vertical crossection of the matrix, represented by list-of-tuples. On the other hand each element of tuple can be either a single item (like "B1" or "B2") or a list (like "(5Ø1 5Ø2 517)"). This type of specificator can be illustrated by the following structure (Fig. 2):

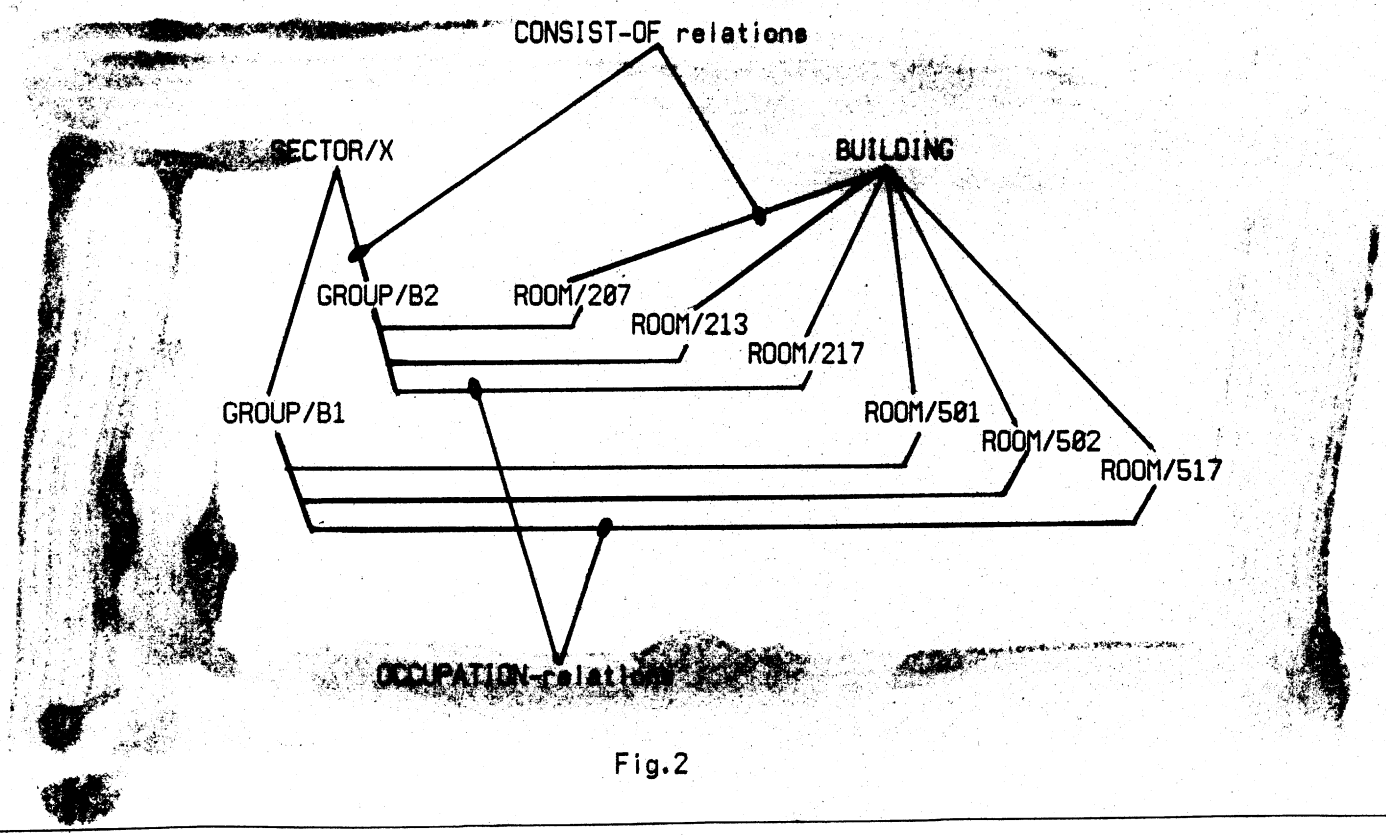

Fig.2

Sometimes it is ambiguous what type of relation should be used to express certain facts. For instance, the correspondence between "Groups" and "Projects" in the Research Institute can be expressed either in the form of subordinations (WORK-ON) or in the form of correlations. Fig.3 represents the structure using only subordination specificators, whereas Fig.4 represents almost the same knowledge with the aid of correlation specificator (RESPONSIBILITY).

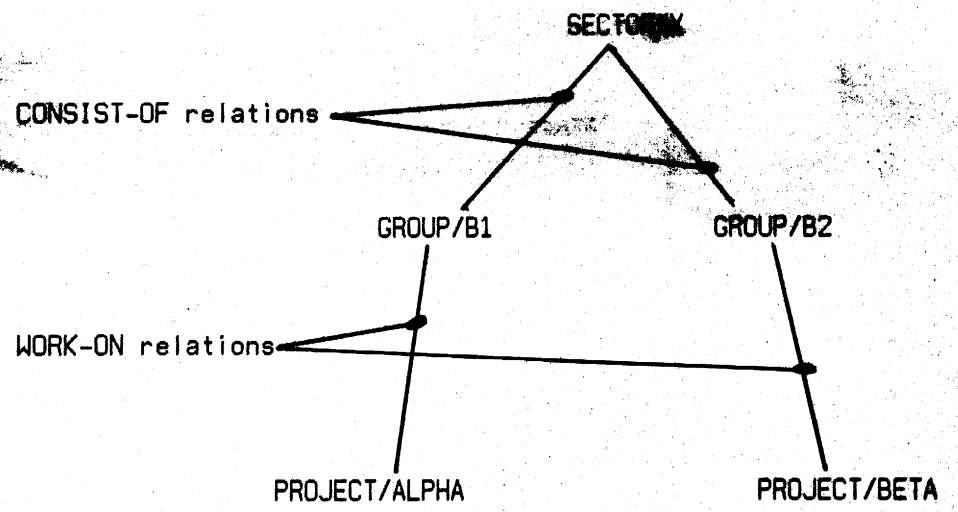

Fig.3

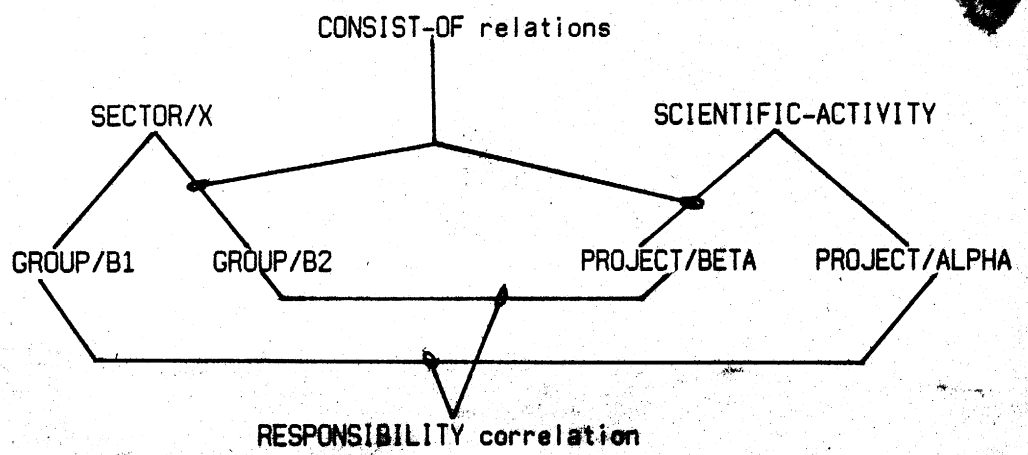

Fig.4

Subordination- and correlation-type specificators give the possibility of representing a static, affirmative knowledge about the domain. There is a need also to represent suppositions of the type:

"If a person with professional experience in symbol manipulation is employed in the Group B2 then Project BETA could be finished in six months". (*)

Condition is a type of specificator which helps to represent conditional knowledge. General syntax of condition:

(condition-name set-of-predicates list-of-expressions) (4)

where

<set-of-predicates>::= <predicate> |
(<predicate> [{AND | OR} <predicate>]...)

<list-of-expressions>::= <expression> [<expression>] ...

<predicate>::= <specificator> | <programming-system-predicate>

<expression>::= <specificator> | <programming-system-expression>

Evaluation of conditions is going on in the following way:
1) Set-of-predicates is evaluated as a logical expression with respect to AND and OR operators, where precedence is established either by brackets or by usual operator precedence. Each predicate can be either in the form of specificator or in the form of any eligible predicate for a given programming system, such as (EQUAL arg1 arg2) in LISP.

Specificator is evaluated to truth value T if an appropriate relation exists already in the Data Base; otherwise it is evaluated to the false value NIL. Programming-system-predicate is evaluated to truth or false value according to appropriate rules of the implementation language.

An additional feature is the possibility of using a specificator as an argument of programming-system-predicate. Also, specificator used within condition is allowed to contain variables, which can get their values by application of condition to the Data Base (see example below).

2) If the value of the set-of-predicates is T, then the list-of-expressions is evaluated, otherwise evaluation returns with NIL (meaning that given condition cannot be applied to the existing Data Base).

Each expression in the list-of-expressions can be either specificator or any expression eligible for the given programming system. Normally, during evaluation of the list-of-expressions, some temporary amendments are made in the Data Base and specificator variables are bound to particular objects thus permitting analysis of the results of new relation combinations.

Example:

The supposition expressed in the natural language statement (*) on page 14 can be represented to the system in a following way:

```
(PROJECT-ACCELERATION-CONDITION  ((CONSIST-OF  GROUP/B2  (?X(?Y)))
        AND  (HAS-ATTRIBUTE  ?X  (EXPERIENCE PROFESSIONAL))
        AND  (HAS-ATTRIBUTE  ?X  (SPECIALTY SYMBOL-MANIPULATION)))
        (DURATION PROJECT/BETA SIX-MONTHS))                          (**)
```

In this example some new possibilites are illustrated. Set-of-predicates consist of three specificators connected by AND operators. Free variable X (denoted by the question-mark) participates in all

these specificators, which helps to represent the following idea:

> "If any person could be found who is a member of Group B2 and has a professional experience in his specialty, namely symbol-manipulation, then PROJECT-ACCELERATION-CONDITION would be satisfied and as a result a relation (DURATION PROJECT/BETA SIX-MONTHS) would be true."

Free variable Y represents "the rest of GROUP B2" which can consist of one or more persons.

The lists (EXPERIENCE PROFESSIONAL) and (SPECIALTY SYMBOL-MANIPULATION) are different forms of X's attributes which can be attached to object X in the Data Base either directly as in Fig.5 or indirectly as in Fig.6.

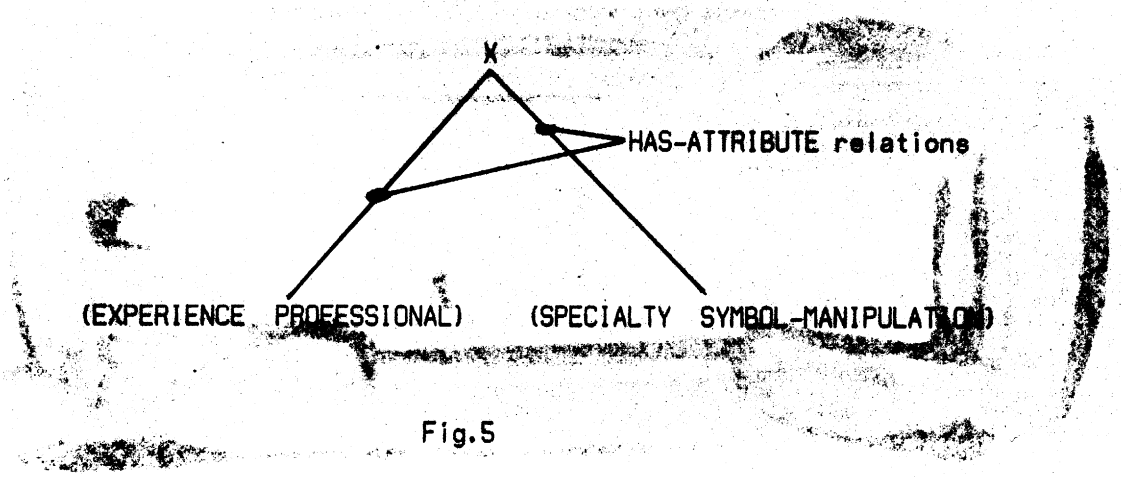

Fig.5

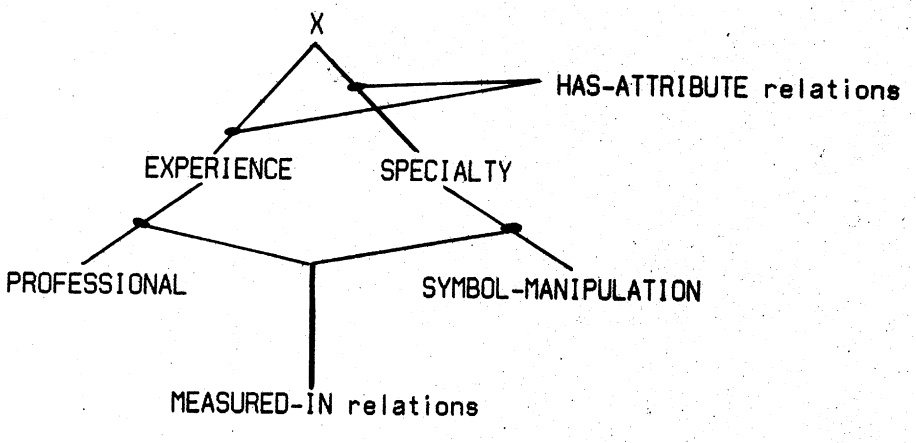

Fig.6

Conditions are not evaluated when they are introduced to the system and they do nothing with the existing model structure in the Data Base. Instead, they are collected (like theorems in PLANNER), together with associated lists of patterns which should permit their pattern-directed invocation. In the previous example the pattern associated with PROJECT-ACCELERATION-CONDITION is: (DURATION PROJECT/BETA SIX-MONTHS).

In the case when several expressions constitute list-of-expressions in the condition, a complex pattern is created which sometimes looks similar to a small tree-like substructure kept temporarily apart from the main model structure. Possible utilization of this substructure can happen during a pattern-directed request (see below) when a small tree-like pattern is matched against different pieces of the gigantic Data Base.

5. Requests

Requests are used in the stage of model analyzing and getting information about different aspects of the simulated establishment. Underlying motives for different forms of requests are described informally below.

A). There is a need for information about some substructure representing a particular set of relations between immediately connected objects, their attributes, measures, etc. This type of information could be obtained through a "plane-pattern-request" (P-P-R).

Syntactically, plane-pattern-requests correspond to subordination- or correlation-type specificators with question-marked variables in the places of unknown elements.

Examples:

A-1) (P-P-R (?X LABORATORY SECTOR))

This request can have the meaning: "What is a direct relationship between objects, called LABORATORY and SECTOR?"

A-2) (P-P-R (OCCUPATION ?Z ROOM (5Ø1 517)))

This request can correspond to the question: "Who occupies rooms 5Ø1 and 517?"

A-3) (P-P-R (WORK-ON SECTOR/A1 ?ACTIVITY))

This expression can represent the question: "What is the name of activity carried out by SECTOR/A1?"

B). In addition to possibilities given by plane-pattern-requests, it is necessary to discover deep interrelations or "relation paths" between particular objects or groups of objects. A request for this kind of information can be named "cross-pattern-request" (C-P-R).

Syntactically, cross-pattern-request looks like a set-of-predicates in condition-type specificator.

Example:

(C-P-R (WORK-ON (?X(?YPP PROJECT/BETA) AND (HAS-ATTRIBUTE ?X
((DEGREE PHD-COMP-SC) (SPECIALTY COMPILER-CONSTRUCTION)))))

This expression can represent the following inquiry:
"Is there anything in the Data Base which is working on Project BETA and has as its part an item (?X), presumably a human being, who is lucky to have such attribute as PhD in Computer Science and is specializing in Compiler Construction?"

An end-user requiring this information would probably be satisfied if the system extracts from the Data Base and displays a structure of Fig.7 as an answer.

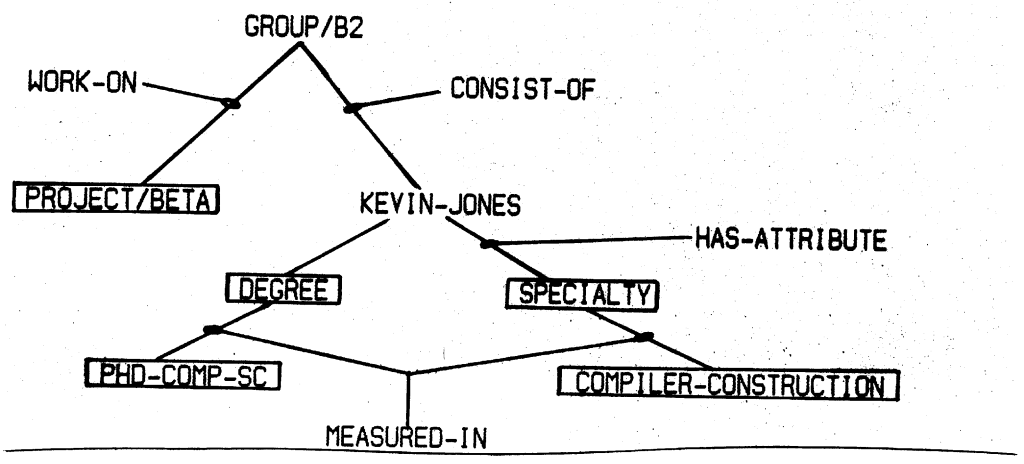

Fig.7

In fig.7 only the objects in rectangles are participating in the request, whereas all other objects and their relationships are extracted from the Data Base.

C). Finally, an end-user might have an understandable desire to look at possible results of some rearrangements in the model. Condition-type specificators can provide the necessary information if they are prepared to do that. "Call-by-name" (C-B-N) or "Pattern-directed-request" (P-D-R) are possible forms of condition invocation.

Examples:

C-1) (C-B-N PROJECT-ACCELERATION-CONDITION (X = JIM-JONES))

C-2) (P-D-R (DURATION PROJECT/BETA ?W))

Both these requests could invoke, for instance, specificator (**) given in the example on page 16. In the first type of invocation (example C-1) variable X should be bound to JIM-JONES value prior to evaluation of the set-of-predicates in (**). Possible system response could be the answer:

(HAS-ATTRIBUTE JIM-JONES (EXPERIENCE PROFESSIONAL)) IS FAULT

THEREFORE PROJECT-ACCELERATION-CONDITION CANNOT BE SATISFIED.

In example C-2 variable X has to be bound iteratively to different elements, members of GROUP/B2, until the whole set-of-predicates in (**) is evaluated to the truth value. If no one member of GROUP/B2 has the necessary attributes to satisfy the given set-of-predicates, then the answer to this request would be negative.

6. Implementation and Extension

A simple model designed on the basis of the above considerations can be implemented without serious problems if a programming system is available which allows flexible symbol manipulation and has the possibility of creating a structured Data Base. Even standard LISP can be used for this purpose, but advanced systems (like PLANNER[1], CONNIVER [2] and SETL[3]) permit much more convenience and flexibility in the stage of developing constructors (i.e. functions defining the internal structure of the model).

PLANNER and CONNIVER are also very advantageous because they have system-defined pattern search and pattern-directed procedure invocation. Both these features are in fact the major mechanisms for implementation of requests.

The proposed approach for developing a model of a Research Institute is strongly influenced by the ideas of MAPL language[4], particularly by the MAPL relational approach to knowledge representation. In this memo the author has tried to stress the idea that it is worthwhile to concentrate on the development of the following aspects:

- creating a frame of the model: definition of general model structure, introduction of relation types and development of an assortment of relation patterns which will be used in the later stage of model specification:
- design of a set of operators manipulating model elements: these operators can specify a search for particular node (object) or arc (relationship), or they can help to extract information about associatively connected sets of elements, etc.

Certain balance should be achieved between the amount of work performed by system-programmer, by domain-expert and by an end-user. All three categories should be permitted to participate to some extent in model creation and amendment as well as in knowledge specification and utilization.

Classifying possible statements into three main types (constructors, specificators and requests) is, of course, not the only possible classification for a given problem. However, this kind of classification helps to clarify what kind of work has to be done in order to achieve a simple and extensible solution for the problem.

In terms of a proposed approach, extension of the model can be performed by the development of additional types of specificators and requests. In some cases it requires only minor modifications of existing constructors, in other cases new constructors should be created with the purpose of complementing or replacing some of the existing constructors. It is essential that modification of the set of constructors should not require extensive modification of the existing Data Base, but instead just give another point of view of the heap of knowledge stored in the computer memory.

References.

1. C.Hewitt, Description and Theoretical Analysis of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot, Ph.D. Thesis, MIT, AI-TR-258, April 1972

2. D.McDermott, G. Sussman, The CONNIVER Reference Manual, MIT, AI-M-259, May 1972

3. SETL Users Manual, Courant Institute of Mathematical Sciences, October 1972

4. W.Martin, P.Krumland, MAPL - A Language for Describing Models of the World, MIT, PROJECT MAC, Automatic Programming Int. Memo 6, 1972

CS-TR Scanning Project
Document Control Form

Date : 2 / 29 / 96

Report # LCS-TR-118

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 23(29-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☒ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☒ Other: BIBLIOGRAPHIC DATA SHEET (2)

Page Data:

Blank Pages (by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:
(A) IMAGE MAP: (1-23) UNNUMBERED TITLE, ABSTRACT PAGES, 3-23
(24-29) SCANCONTROL, BIBLIO DATA SHEET(2), TARGETS(3)
(A) CUT&PASTE FIGS ON PAGES 11-13, 16-17, 19

Scanning Agent Signoff:
Date Received: 2/29/96 Date Scanned: 3/13/96 Date Returned: 3/14/96

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 3/11/1996

M.I.T. Libraries
Document Services