Luger, G.F. & Stubblefield *Paradigm Dependent Human Factors in Expert Systems Design.* (Invited Keynote Paper) Expert Systems in Telecommunications, A Symposium sponsered by the Metropolitan Chapter, Human Factors Society, NY, NY, 1987.

The Communications Technical Group

and the

Metropolitan Chapter of the Human Factors Society

Expert System in Telecommunications

A Symposium

March 9, 1987

Time: 9:00 - 5:00

New York City, NY

Theatre in the Gallery of the
IBM-Building,
590 Madison Avenue at 57th Street,
New York City

Symposium Co-Chairs:

Doug Antonelli
IBM-E04/664
P.O. Box 12195
Research Triangle Park,
NC 27709
(919) 254-0147

Derek Schultz
Media Design Associates
151 Route 206, B24-5
Flanders,
NJ 07836
(201) 584-4991

Paradigm Dependent Human Factors Issues in Expert System Design

by
George F. Luger & William A Stubblefield
Department of Computer Science
University of New Mexico
Albuquerque NM, 87131

Abstract

There are a number of different "interaction styles" that have evolved with the development of Expert System applications. These include a *batch* style, a *direct access to data* approach, as well as the more usual *question and answer* and *menu or form driven* approaches. In addition, many expert systems are beginning to implement graphics based *direct manipulation* interfaces. The goal of this paper is to briefly describe these formats as well as the various Expert System "architectures" that support them. We also show that the choice of an Expert System interface cannot be made independently of these software "architectures." This has important implications for the selection of software "packages," "shells" or languages for designing Expert Systems.

1. Introduction

A number of important human factor issues in Expert System design are very much "paradigm dependent," in that the selection of a software "architecture" can commit the program designer to some crucial (and oftentimes unacceptable) interface designs. This paper, in section 2, goes over some common "styles" for Expert System interactions. Section 3 reviews the common software architectures for Expert System applications. Section 4 discusses the commitments that each of these architectures can impose on the interface and query designs in Expert Systems.

2. Interaction Styles for Expert Systems

The five interaction styles that we present are 1) *batch* [8], 2) *question and answer* [1, 10], 3) *direct access of instruments or data*, [1] 4) *menu or form* driven, and 5) *graphics* or *direct manipulation* approaches. [9] We now present samples of each style along with examples of important Expert Systems applications or software packages where each of these approaches is taken.

XCON, or eXpert CONfiguration program designed at Digital Equipment Corp, is a classic example of the *batch* style of Expert System interaction. [8] Digital developed this software product, with John McDermott of Carnegie Mellon University as external consultant, during the 1979-1982 time period. Currently this program configures all VAX class, as well as the full range of PDP-11 computers delivered by Digital. The configuration program takes the specifications for the computer from the purchaser's order, and, constrained by the hardware limitations (bus, disk size and controllers, main memory, etc) builds the particular computer to meet the customer's requirements. This configuration includes the size of the case, location of the boards, appropriate cabling and all necessary components for a running system.

The data for this program is prepared before a program run, usually in file format and then is used to "initialize" appropriate parameters prior to the run. There is no interactive question and answer dialogue, simply the building of the appropriate computer from its specifications. The

current version of XCON is running using the OPS5 production system language. OPS is primarily a "data driven" package, in that it takes the facts (specifications) of the situation, and using the appropriate rules (referred to as "productions") produces the required result. XCON is a commercially successful program, having on the order of 5000 rules and having configured well over one hundred thousand computers.

MYCIN is a program designed at Stanford University as a joint project of the Medical School and Computer Science Department. [1] The program, developed in the middle and late 1970s is the archetype of the "rule based" Expert System. The program has just over 500 rules, one of which in "English" form is presented along with part of a MYCIN "dialog" as figure 1. The goal of the MYCIN program is to diagnose bacterial infections in patients, especially spinal meningitis.

Once preliminary data is taken from the patient (name, age, sex, symptoms), if appropriate data is present (the patient has headaches, dizziness, etc.), MYCIN tries to establish whether the patient has Meningitis. This program is goal driven, and in trying to establish a goal, the questioning takes on a reasonable and coherent form. This is simply because the questions are focused on trying to establish some particular goal of the problem (Is this a staph infection?).

A disadvantage of the MYCIN question and answer approach is that it requires large amounts of typing, with a single interview taking 30 to 40 minutes. This is slow going and can be very error prone. Shortliffe [1] one of the designers of MYCIN, says the rigidity and awkwardness of this dialog process has to a large extent accounted for MYCIN's lack of acceptance among doctors. Indeed by objective (double blind) evaluations of MYCIN's patient diagnoses, it outperformed the human doctors it was compared with, including several specialists at Stanford. Yet this program, largely because of its cumbersome interface, is not used!!

Other more recent programs at Stanford, especially PUFF and ONCOCIN, were designed to overcome the cumbersome interface of MYCIN. PUFF, a program for the diagnosis of pulmonary problems, takes information directly from the patient's testing devices such as breath capacity and heart monitoring during exercise. The direct taking of data from instruments for the analysis by the rules of the program makes the program seem more "intelligent" in that it does not ask for information that it can either infer or access directly. This also removes the medical technician or doctor from the "data input" mode and lets them oversee more important aspects of the analysis.

ONCOCIN [1], another program from Stanford, is designed for recommendation of cancer therapies. ONCOCIN is able to directly access the records of the patient for relevant information, from age, weight, to the results of various tests. The general style of both PUFF and ONCOCIN is the same as MYCIN: they are goal driven rule based programs with certainty factors giving the "confidences" the program has in its various responses. The programs also support the *why and how* queries that the user can interject at any time in the problem solving "dialog."

The fourth interface model for Expert Systems is the *menu or form driven* approach. This interface model seems to be the approach taken by many "PC" based shells such as EXSYS, INSIGHT, GURU, FIRST CLASS and others. This interaction style is similar to question and answer in that the system asks the user for new information as it is needed, however the response is a menu selection rather than a typed answer.

When each new rule is added to the shell program a "menu" is designed of English like queries related to the particular information required for the rule. These menus can improve the interaction of the original *question and answer* approach taken by early systems like MYCIN. They are also much less error prone in that the rule designer can specify exactly what answers are expected from each query. Still, a good bit of rigidity can appear in the interactive session, and the problem solving still is rather slow.

The final interactive style for Expert System design is the *graphic* or *direct manipulation* approach. The roots of this paradigm are located in the SmallTalk language, or the "object oriented" model for problem solving, [3,7] developed at XEROX in the early 1970s. Later work at MIT, with the development of Flavors, augmented and extended the original SmallTalk approach. Without getting into too much technical detail, the "object oriented" approach provides a simulation of the relationships within the problem solving area under consideration. This simulation

includes a graphic interaction whereby the user can directly manipulate the system. This usually happens by the user moving the "mouse" to a component of the simulation and then "clicking" the mouse to change one of the values of that component. For instance, one component of a steam cycle simulation might be a valve that can be turned on or off by clicking over the image of the valve. The color of the valve often changes when the clicking mouse changes its state.

One of the most important examples of the object oriented approach is a program called STEAMER [10] developed by the US Navy to train their cadets in understanding and using the propulsion systems for large vessels. STEAMER has many screens and "scenarios" each of which capture some aspect of the propulsion system. For example, the steam cycle could be represented graphically on the screen, including pumps, condensers, water tanks and various valves and pipes connecting this system together. The user can then change the various parameters of the system (turning a valve on or off, changing the steam pressure, etc) to see how these changes effect the entire system.

The *graphic or direct manipulation* model [9] can offer an excellent interface for appropriate problems, for teaching neophytes about steam engines, for example. Other applications, such as medical diagnosis, may not be at all appropriate for this approach, however. There is also a fairly high "start up" cost on such systems. Steamer requires about $100k hardware investment. This, of course, will not be appropriate for simpler applications. There is also a high "learning" cost for the programmer to get acquainted with the Object Oriented tools.

3. Architectures for Expert System Design

A shell for an Expert System is a "framework" for creating an Expert System. One of the prime design features of Expert Systems is the separation of the program modules for rules and inferencing. A schematic for such a division is presented in figure 2.

Carefully observing this inference engine / knowledge base distinction gives the program an important modularity for adding new rules or altering the explanation query routines, without causing serious disruption of the other aspects of the program. In particular a shell for an Expert System is the inferencing module of figure 2 without the knowledge base. The creator of the Expert System then designs rules appropriate to the particular application at hand, without having to redesign the entire inferencing package. Many of these shells come from successful expert system applications, probably the most famous shell being EMYCIN, from the original MYCIN research [1]. The EMYCIN shell allowed the designers of PUFF create that application in about five person years, whereas the original MYCIN program took about forty person years.

We will cover the three main software "architectures" for Expert system design: a) the **decision tree** model [5], b) the **rule based** model [1,2,6], and c) the **frame or object oriented** approach [3,7]. The fundamental assumption for good program design in each of these areas, as mentioned above, is the separation of the knowledge rules from the control of the program. In the first two architectures mentioned above the decision trees and the rules capture the knowledge of the program and these structures are manipulated by their respective "engines." In the object oriented approach the entities of the problem are each represented by an appropriate object and the relationship between these objects are captured by the "methods" (procedures that make up part of the object's specification) and "messages" (information that the objects send and receive from the objects to which they are related). Thus, new entities and rule relationships are added by creating new objects with appropriate methods and designed to send and receive messages appropriate to the other objects already in the system.

The decision tree Expert System model (exemplified by Expert System shells such as Expert Ease, Expert Edge, and First Class) takes a number of examples of a particular situation and "induces" decision tables from these examples. Figure 3 gives a table of choices for electronic toys that might be available in a game's shop for groups of people, depending on the cost and age of the potential recipient. The user gives the program concrete examples and from these the program "induces" general rules for future decisions.

The variables for this decision tree may be "integer" as well as the "logical" values shown in figure 3. For example, the age could be entered as 12 instead of child, or 25 instead of adult. Before running a new evaluation (taking another gift enquiry), the program creates decision tables. These decision tables optimize the process of classifying problem instances. The tables are constructed so that questions which make the greatest distinction between problem instances are asked first. This leads to an optimal decision tree in the sense that classifications can be made with the fewest possible questions. In the example of figure 3, the decision tree then takes new instances asking the cost and age of the intended gift recipient, and produces an answer depending on the rule "induced." An age of 20, for example would be classified with the previous instance of the 25 year old adult rather than the 12 year old child for the gift recommendation. Confidence measures are possible in such a model, as well as the ability to search all possibilities that had any confidence at all of being interesting. The usual "why and how" queries are not available in this model.

The basic paradigm for the "rule based" Expert System model is the production system. [2] The production system is a construct that organizes the inference rules in the system and applies them to the solution of a particular problem instance. Each rule is stated as a "condition - action" pair within the production model, and corresponds to a single "chunk" of problem solving knowledge: the condition is a pattern that determines when that knowledge should be applied; the action is the conclusion that is drawn by the production. A description of a problem being solved is kept in the "working memory" of the system. Working memory is initialized to the starting problem description; at each cycle of the production system loop, an appropriate production is selected and used to modify working memory. Productions may also generate user queries for information. A simple schematic for the production system is presented in figure 4.

The production system may be run in either "data driven"' or "goal driven" mode. The data driven mode takes the facts of a problem and presents them to the conditions of the rules in the production system. When the conditions of a particular rule are matched the action of that rule is asserted as a new true description of the problem situation. This new description is again passed to the conditions of the productions. When another rule is matched another action is made part of the description. This simple cycle continues until either the goal is found or there are no more productions that match the problem description.

In "goal driven" problem solving the goal to be found is presented to the actions of the productions. When an action matches, the conditions of that action are created as new (sub)goals to be established. These new goals are presented to the actions of the productions and another match indicates further conditions that must be established to prove these (sub)goals. Thus the search proceeds until actions are matched that are always true (the given information of the problem) and "back chaining" links these facts with the original goals of the problem, or until no matches occur and the search "fails". In the cases of either "data" or "goal" driven reasoning, the production system offers a model for the problem solving, and the resultant search process can be characterized by a graph.

The explicit use of rules within the production system model make possible the "why - how" queries of the user. "why did you ask me this?" is answered by presentation of the particular production rule under consideration when the query was made. The answer to "why" is the presentation of the production rule under consideration. "how did you get that information?" is answered by presentation of that part of the search space used to determine that the fact was true. Thus both "why" and "how" queries are artifacts of the production rule based search of a graph. Examples of "why" and "how" queries may be found in the MYCIN segment presented in figure 2.

As mentioned at the end of the preceding section, the "object oriented" Expert System package is based on the concepts of the SmallTalk and Flavors languages. In this approach, each "object" in the problem domain is represented as a computational object in the system. These computational objects include named slots for values associated with the object and procedural information that implements the objects behavior. For example, figure 5 illustrates how objects might be used to describe a water pump in a (hypothetical) STEAMER like system. The "centrifugal-pump" object defines the class of centrifugal pumps; this definition includes value

slots for pumping capacity, its connections to other objects in the system, its membership in the more general class of pump objects etc. It also includes procedural information on how to display a centrifugal pump on the computer screen, how to compute its fuel consumption, how a pump behaves in case of an oil leak, etc. The instance object for "pump-17" is a member of this class, and binds values to these value slots, reflecting the particulars of that pump. In addition, it inherits the procedural information from the class object, enabling it to behave in a fashion consistent with the class object. Since this object "behaves" much like a real pump, it may be used by the system to reason about the behaviors of systems involving centrifugal pumps. For example, such a system could answer questions such as "what happens if pump-17 gets an oil leak."

While the "architecture" of this approach is a bit more complex (and beyond the scope of this paper) still the power of simulation with "active objects" and "messages" is a powerful tool for Expert Systems problem solving. We add comments on the human factors issues related to this, as well as the other approaches to Expert System design, in the next section.

4. Paradigm dependent Human Factors Issues

Although the preceding classification of Expert System paradigms was perhaps too superficial, it does provide a starting point for considering the human factors issues raised by the selection of a particular architectural approach. In the remainder of this section, we discuss not only the impact of such a commitment on user interface design but consider its effect on the broader range of human issues (ease of development, learnability, expense, etc.) involved in building and using Expert Systems. This discussion takes the form of a list of advantages and disadvantages of each of the major paradigms under discussion.

4.1 The decision tree

The advantages of the decision tree model include:

1. The implementation is very straight forward and concrete, requiring only the presentation of examples (see figure 2).

2. The program automatically generates an optimal inferencing path based on its "induction" algorithm. But see disadvantages 3 below.

3. Because the examples are enumerated there is a natural support for menus. Most induction/decision tree systems automatically produce menus from the set of examples. (e.g. Select one of each option: age, amount of money, etc, see figure 2).

4. The resulting decision tree (and most systems will show the program designer the tree produced) is a static and clean model of a state space search.

5. This methodology is available for both PCs (Expert Ease, First Class,..) as well as for larger systems.

The disadvantages of the decision tree model include:

1. In should be restricted to very small domains (six decision parameters, say), otherwise the "simple" model mentioned in the advantages becomes almost incomprehensible. Generating enough examples to cover all possibilities in a very rich domain may be practically impossible.

2. No allowance for variables eliminates the possibility of entering general rules. This is a great disadvantage for a domain of any complexity.

3. The inductive algorithm determines an optimal discrimination tree. The resulting

order of queries to the user may have no resemblance to intelligent questioning and may indeed seem illogical. The designer cannot change this "optimized" questioning order.

4. Because of disadvantage 3 above, any explanations available may not seem "natural," i.e., reflecting the user's concern over the current state of results.

5. There is very limited expressive power even within the rules. For example, $p \wedge q \rightarrow r \vee s$ is not allowed because as instances $p \wedge q \rightarrow r$ contradicts $p \wedge q \rightarrow s$. One or the other must go before the decision tree is induced, or else something else must be added to $p \wedge q$ to differentiate between r and s.

6. There is no "natural" way to generate graphics to accompany the problem solving.

4.2 The rule based Expert System

Advantages of the rule based model:

1. If - Then rules are a very natural way to capture both knowledge and heuristic information.

2. The designer has direct control over rule structure and a greater flexibility in building inferencing.

3. Both data and goal driven strategies support explanation facilities. See comments in section 3 above, and figure 1.

4. The designer may create general rule relationships, rather than being limited to the enumeration of examples, as in the decision tree approach.

5. By carefully structuring the rule base the program may be made to "reason" in a very realistic fashion. Unlike automatically induced decision trees, a rule base may be structured to ask questions of the user in an order specified by the designer.

6. There is excellent support for both question and answer as well as menu interaction styles.

7. The rule based model is supported on a large number of machines from PCs on up.

Disadvantages of the rule based model:

1. The rule based approach requires more competent programmers. They must understand issues of representation, search, heuristics, and so on.

2. It is difficult to predict the program's behavior from examination of the rule base (for a program of any interesting size!). There must be good editing, tracing, and debugging facilities provided.

3. It offers a very weak model for human performance.

4. Most rule based programs do not directly support graphics or direct object manipulation styles.

5. This is a poor vehicle for taxonomic information, such as that provided in "semantic networks," or "frame based"" reasoning packages.

4.3 The frame or object based systems

Advantages of the "frame - object" paradigm:

1. It offers a natural fit to many domains where "simulation' is an appropriate problem solving tool.

2. It is excellent for capturing taxonomic knowledge with its classes and inheritance. For example the facts that Huey, Louie, and Dewey can swim can be represented by the fact that they all are ducks and that every element of the duck "class" can swim. This can be a very important advantage when are large numbers of elements in classes. The exceptions are noted with the individuals.

3. A number of "frame - object" models also support rules (KEE, ART and others) thus offering the strengths of both approaches.

4. There is excellent support for the "graphic object manipulation" style of data entry or program control. The graphic image is an essential aspect of the architecture of this approach.

5. There is the potential for "abstracting out" interface styles. For example, in KEE and other systems, standard (and powerful) interfaces may be used to interact with the system.

Disadvantages of the "frame - object" paradigm:

1. It is a very difficult paradigm for programmers to master.

2. It is expensive requiring lisp machine or comparable workstation. Not yet available on PCs.

3. Very slow development times.

4. Because these "environments" are very complex and interdependent, and still depend on new technology, many of the commercial systems available still have inconsistencies and are less robust than might be desired (they are often very "kludgey").

5. Conclusions

The advantages and disadvantages enumerated above are meant to be the specifics of our conclusions. None the less, it is important to restate the general philosophical issues behind the design of this paper. First of all, human factors issues must be considered when choosing an Expert System design paradigm. Secondly, each of the paradigms described above is limited, but can actually seem very clever in its approach to problem solving. (In fact, the user can dangerously overestimate the actual intelligence and "understanding" of these programs.) Finally, the authors feel very strongly that the human factors choices made in designing an Expert System are very important, if not the most important issues related to the success of the program. Quite simply, if the program doesn't meet the problem solver's expectations and needs, it won't be used.

6. References.

1. Buchanan, B. G., Shortliffe E. H. Rule-Based Expert Systems. Addison Wesley, Reading Mass. 1984.

2. Davis, R., Buchanan, B., and Shortliffe, E. Production Rules as a Representation for a Knowledge Based Consultation Program. In *Artificial Intelligence* 8(1), 1977, 15-45.

3. Fikes, R., Kehler, T. The Role of Frame-Based Representation in Reasoning. In *Communications of the ACM* 28(9) (Sept. 1985).

4. Harmon, P. and King, D. *Expert Systems: Artificial Intelligence in Business.* John Wiley & Sons, Inc. New York.

5. Hart A. The Role of Induction in Knowledge Elicitation. In *Expert Systems* 2(1) (January 1985).

6. Hayes-Roth, F. Rule Based Systems. In *Communications of the ACM* 28(9) (Sept. 1985).

7. Kunz, J. C., Kehler, T. P., Williams, M. D. Applications Development Using a Hybrid AI Development System. In *AI Magazine* 5(3) (Fall 1984).

8. McDermott, J. R1: The Formative Years. In *AI Magazine* 2(2) (Summer 1981).

9. Schneiderman, Ben *Designing the User Interface: Strategies for Effective Human Computer Interaction.* Addison Wesley, Reading Mass. 1987.

10. Waterman, D. A., *A Guide to Expert Systems.* Addison Wesley, Reading Mass. 1985.

Figure 1
A Transcript of a MYCIN Interaction

30) Has Pt358 had any injury or insult to, or defect in the central nervous system (excluding neurosurgery)?
** NO

.
.
.

35) Is meningitis a hospital acquired infection?
** WHY

The user asks "WHY" to see an explanation of the line of reasoning MYCIN is following. The program first rephrases the questions and then show the rule whose premise it is trying to establish.

[i.e. WHY is it important to determine whether the infection was acquired while the patient was hospitalized?]
[2.0] This will aid in determining the organisms (other than those seen on cultures or smears) which might be causing the infection
It has already been established that
[2.1] the infection which requires therapy is meningitis, and
[2.2] organisms were not seen on the stain of the pending csf culture, and
[2.3] the type of the infection is bacterial, and
[2.4] neurosurgery is not one of the relevant items from the history of pt538, and
[2.5] the age of pt538 is greater than 10 days
Therefore if
[2.6] the infection was acquired while the patient was hospitalized then
there is evidence that the organism (other than those seen on cultures or smears) which might be causing the infection is e.coli (.75) staphylococcus-coag-pos (.3) pseudomanas-aeruginosa (.3) klebsiella-pneumoniae (.5)
Author: Yu
[RULE545]

The numbers in parentheses following each conclusion indicate the measure of important of the conclusion. This single number combines frequency and utility considerations, as discussed in session 4.

[BACK TO QUESTION 35...]
** HOW 2.3

At this point the user is asking for additional clarification of low clause 2.3 of the above rule has already been determined. MYCIN shows the numbers of the rules used to make conclusions and will print any of them upon request

[i.e. HOW was it established that the type of the infection is bacterial?]
The follow rules concluded about:
the likelihood that the type of the infection is bacterial

	Rule	Yes	No
[2.7]	RULE148	(.19)	
[2.8]	RULE500	(.51)	
[2.9]	RULE501	(.95)	
[3.0]	RULE502	(.97)	
[3.1]	RULE526	(.98)	
[3.2]	RULE504	(.97)	
[3.3]	RULE524	(.98)	

Thus, it has been established that the type of the infection is bacterial.

MYCIN now prints intermediate conclusion to how the user what is is considering the likely causes of the problem to be. Further questions are motivated by these conclusions

I have found evidence (RULE545 RULE557) that treatment should cover for E.coli causing meningitis.
I have found evidence (RULE545 RULE557 RULE578) that treatment should cover for Pseudomonas-aeruginosa causing meningitis.
I have found evidence (RULE545 RULE557) that treatment should cover for Klebsiella-pneumoniae causing meningitis
I have found evidence (RULE507) that treatment should cover for Diplococcus-pneumoniae causing meningitis
40) Does Pt538 have local neurological signs?
** UNKNOWN

Figure 2
A Prototypical Expert System Architecture

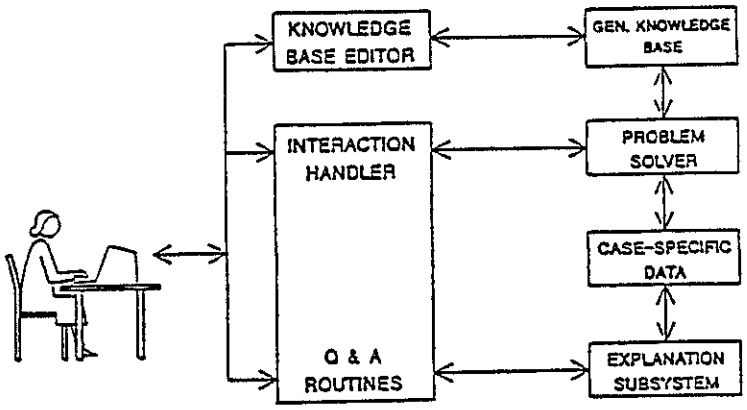


Figure 3
Decision Tables For Toy Selection System

Decision Table for Toy Selection Expert System

Money	Age	Gift
little	adult	calculator
much	adult	MindProber
little	child	Transformer
much	child	Computer

The Resulting Decision Tree

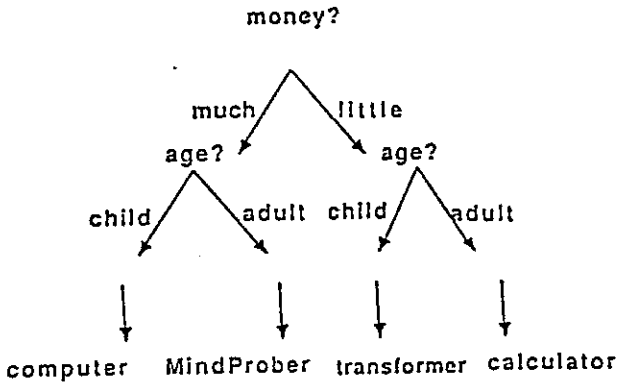
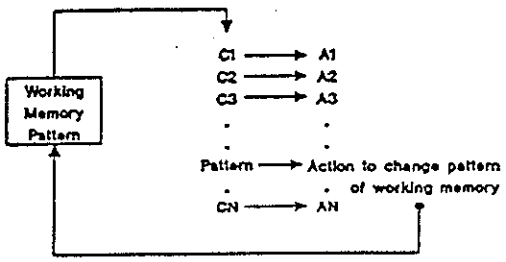


Figure 4
A Production System



CONTROL "LOOPS" UNTIL
WORKING MEMORY PATTERN NO LONGER MATCHES

Figure 5
A Hypothetical Object Oriented Program
Arrows point to other objects in system