BY A. TERRY BAHILL, PAT N. HARRIS, AND ERICH SENN

Sometimes what you learn building an expert system is more important than whether or not it succeeds

LESSONS LEARNED *Building Expert Systems*

We have made many expert systems. Two of our best are Cogito, which gives installation advice for bringing up the UNIX (Bell Laboratories) 4.2BSD operating system on a VAX (Digitial Equipment Corp.), and Data Communication Diagnosis (DCD), which helps a person connect a terminal to a computer. This article will compare and contrast the development, tool selection, and testing of these two systems.

BACKGROUND

Before Cogito was completed, two detailed reference manuals were used for UNIX 4.2BSD installation instructions.[1,2] These voluminous manuals were difficult to use. In contrast, Cogito is easy to use because it filters the information and presents only relevant advice about the user's computer system. Cogito remembers data the user has previously entered and uses it to customize its response. Cogito is written in M.1 (Teknowledge Inc.), runs on a personal computer, and uses *if-then* production rules to encode the task knowledge.

DCD is a knowledge-based system designed to help engineers connect a peripheral device (terminal or printer) to a computer, modem, or local area network. Most of DCD was written in OPS5, although parts were written in C and M.1.

Instructions for connecting peripheral devices to computers are contained in the reference and user manuals of the devices. Unfortunately, their usage is fraught with difficulty. It is not easy to collect all the relevant information for the two devices in question since this information is spread over several chapters (for example, pin connections are discussed in the hardware section and transfer parameters in the set-up section) in several manuals written by different companies. DCD had three specific requirements:

- Provide enough advice to people who have some previous computing experience but little experience in data communica-

tions so they can design a cable with a 25-pin connector on each end that can be used to connect a peripheral device to a host device

- Read in a description (name) of the devices and ask the user for values of parameters not stored in the expert system's data base, provide a help feature to support the process of answering questions, and produce a sequence of instructions to help the user successfully interconnect the devices
- Derive an approximate solution even if some information is missing or the answers to questions are unknown.

Unfortunately, modern computer users are often overwhelmed with information. Expert systems can help reverse this trend. Voluminous data with many interrelationships presented over a short time span can overload the user. Data become useful information when the relationships between items are cohesive. Both Cogito and DCD eliminate extraneous information and relate relevant information, reducing the chances of user overload.

CREATING THE KNOWLEDGE BASE

Our first expert system,[3] made in 1984, had a flat tree: every premise was tied directly to the conclusion. However, to emulate human input-output behavior, we believed that an expert system should have a bushy tree. We discovered that we had to invent intermediary conclusions if we were to mimic a consultation with a human. Furthermore, humans have a limited short-term memory, thus a solution going from premises to conclusions should follow a path of simple knowledge intermediates.

Listing 1 shows the hierarchy we used to make a bushy tree for DCD. Of course, even with this hierarchy, the rules do not write themselves; the expert still had to add knowledge to get the rules. For example, the expert knew that pins 2 and 3 are used to transmit and receive data. So if pin 2 transmits data, pin 3 receives data.

The *if-then* production rules shown in Figure 1 were derived from the hierarchy of Listing 1. The hierarchy shown in Listing 1 helped make sure rules were not overlooked. For example, it was easy to check that each pin was contained in at least one rule.

This hierarchy was not very structured, but it still helped in writing the rules. Rules are easier to write in problem domains that can be structured more systematically with a technique that divides the problem space into a set of objects where every object has attributes and every attribute has values. This object-attribute-value ordered set is called an O-A-V triplet.

Typical usage of O-A-V triplets is: the object *has* an attribute and a value *is* an attribute. Figure 2 shows a knowledge base represented in the O-A-V schema. The lower left triplet in this knowledge base can be written as "coat *has* markings" and "striped *is* a marking."

Another way of writing the O-A-V triplet is as a sentence ("attribute of object is value"); for example, "marking of coat is striped." This usage makes the construction of facts easy:

ARTWORK: PAUL AMBROSE/FPG INTL. CORP.

```
description-parameters
name                [A..Z], [a..z], [-]
sytek-unit-number   XXXX,X --> [0..9]


general-parameters
baud rate           110 to 19200, unknown
standard            RS-232C, RS-422, CENTRONICS, unknown
synchronization     asynchronous, synchronous, unknown
duplexity           full-duplex, half-duplex, simplex, unknown
hardware-protocol   not-required, required, unknown
software-protocol   none, Xon/Xoff, ACK/NAK, unknown
echo                on, off, unknown

function
mapping       2-2, 2-3, unknown
pin1          frame-ground, not-available, unknown
pin2          transmit-data, receive-data, unknown
pin3          receive-data, transmit-data, unknown
pin4          request-to-send, not-available, unknown
pin5          clear-to-send, not-available, unknown
pin6          data-set-ready, not-available, unknown
pin7          signal-ground, not-available, unknown
pin8          data-carrier-detect, not-available, unknown
pin20         data-terminal-ready, not-available, unknown

character-format
data-bits     7, 8, unknown
stop-bits     1, 2, unknown
parity        none, even, odd, unknown
```

LISTING 1. Parameter hierarchy.

```
type of animal is mammal
category of mammal is ungulate
extremities of ungulate is hooves.
```

With such facts, it is easy to write *if-then* production rules, for example:

```
if composition of coat is hair
then type of animal is mammal.

if type of animal is mammal
and extremities of animal is hooves
then category of animal is ungulate.

if category of animal is ungulate
and marking of coat is stripes
then identity of animal is zebra.
```

A tree is seldom built all at one time; instead, branches are added incrementally. When a branch is added, sometimes something that is a value at one level becomes an object at a lower level ("mammal" is a value in "type of animal is mammal," but becomes an object in "category of mammal is ungulate.") This O-A-V=O-A-V linkage is shown in the right column of Figure 2.

In contrast, sometimes O-A-V triplets are linked together without an intervening attribute:

```
type of animal is mammal

marking of coat is stripes.
```

This O-A-V-O-A-V linkage is shown in the left column of Figure 2.

Teknowledge distributes a tutorial expert system, SACON, with their PC-based shell, M.1. The company has been refining SACON for 10 years. It is composed exclusively of O-A-V=O-A-V linkages like those in the right column of Figure 2. This implies that perhaps these are the best linkages for an expert system.

We have subsequently tried several other methods for constructing *if-then* production rules without forcing human experts to distort their reasoning processes to produce *if-then*-type rules. We have found the analytic hierarchy process[4,5] helpful in constructing O-A-V triplets.

FIGURE 1. *If-then* production rules derived from the hierarchy in Listing 1.

```
goal = advice.
question(computer-pin2) = 'What is the function of pin 2
   on the computer side?'.
legalvals(computer-pin2) =
[transmit-data, receive-data, unknown].

if computer-pin2 = transmit-data
then computer-pin3 = receive-data.

if computer-pin3 = receive-data
and terminal-pin3 = receive-data
then sketch = diagram1.

if sketch = diagram1
and display(['Interconnect the data signals like this

  terminal                computer
    2 +                     + 2
      \                   /
       \                 /
    3 +  \               /  + 3
    7 +   \_____________/   + 7'.nl .nl])
then advice = ok.
```

SELECTING THE PROPER TOOL

Consider the following analogy. A sculptor wishes to create an image of a dove, and has both marble and ice in which to carve. Although the mediums of ice and marble require different tools and speed of construction, the marble and ice doves are the same as the image within the sculptor's mind. The difference in mediums does not affect the result. Now suppose the sculptor wants to sculpt with iron rebar and a torch. This dove will not appear the same. In this case, the medium does affect the result.

Current expert system shells force a structure on the knowledge base that may be different from the expert's structure; for example, requiring the knowledge to appear as production rules, semantic networks, or frames. When an expert has trouble developing a rule that covers a certain situation,

it is often said that the expert's knowledge is nonverbal and internalized. Alternatively, perhaps the expert cannot develop a rule that fits the situation because he or she does not think in terms of *if-then* production rules.

Cogito provided many examples of how humans are forced to fit their knowledge into the constraints of a tool. Cogito has many rules that might otherwise be simple lists. Furthermore, Cogito's problem domain is probably most suitable for a forward-chaining strategy. If the inference engine had been capable of extensive forward-chaining, the knowledge base could have been shortened. Thus, the tool used definitely affected the coding of the knowledge.

However, an engineer's job is to find and apply the best available tool to solve a problem. The tools available for Cogito were M.1 and OPS5. The decision to use M.1 was primarily based on the differences in rule implementations and ease of input and output. Cogito's rules are English-like phrases. OPS5 rules are reminiscent of LISP code; experts who are not familiar with OPS5 or LISP have trouble reading and understanding OPS5 rules. Rule readability is important so the knowledge engineer can verify with the expert that the intent of the rule is the same as the coded rule. With M.1, the knowledge engineer can concentrate on problem semantics and not on the syntax of OPS5 or LISP.

We selected OPS5 for DCD primarily because it used forward chaining, and this seemed the best type of inferencing when all the data are available at the beginning of the inferencing process. However, a subset of the problem (creating a cable plan) was also written in M.1, which gave us the opportunity to compare these two tools.

The quality of a system's user interface is crucial for a system's survival. M.1 has an advantage in this matter since it provides legal value checking, which is difficult and cumbersome to implement in OPS5. The human-computer interface of M.1 is better than that of OPS5, but this did not matter in our case since all the data were collected from the human expert before the inference process started using a module written in C.

The knowledge base contains the rules to solve the problem at hand. Rules written in M.1 are easier to read than those in OPS5. However, the easier rule syntax of M.1 is only an advantage for novice knowledge engineers. Based on our experiences, we think experienced programmers can write either M.1 or OPS5 rules with equal ease. On the other hand, M.1 is at a disadvantage because its conclusions cannot have values for several different expressions. This handicap is not shared by OPS5, where several different expressions can be in the conclusion of a

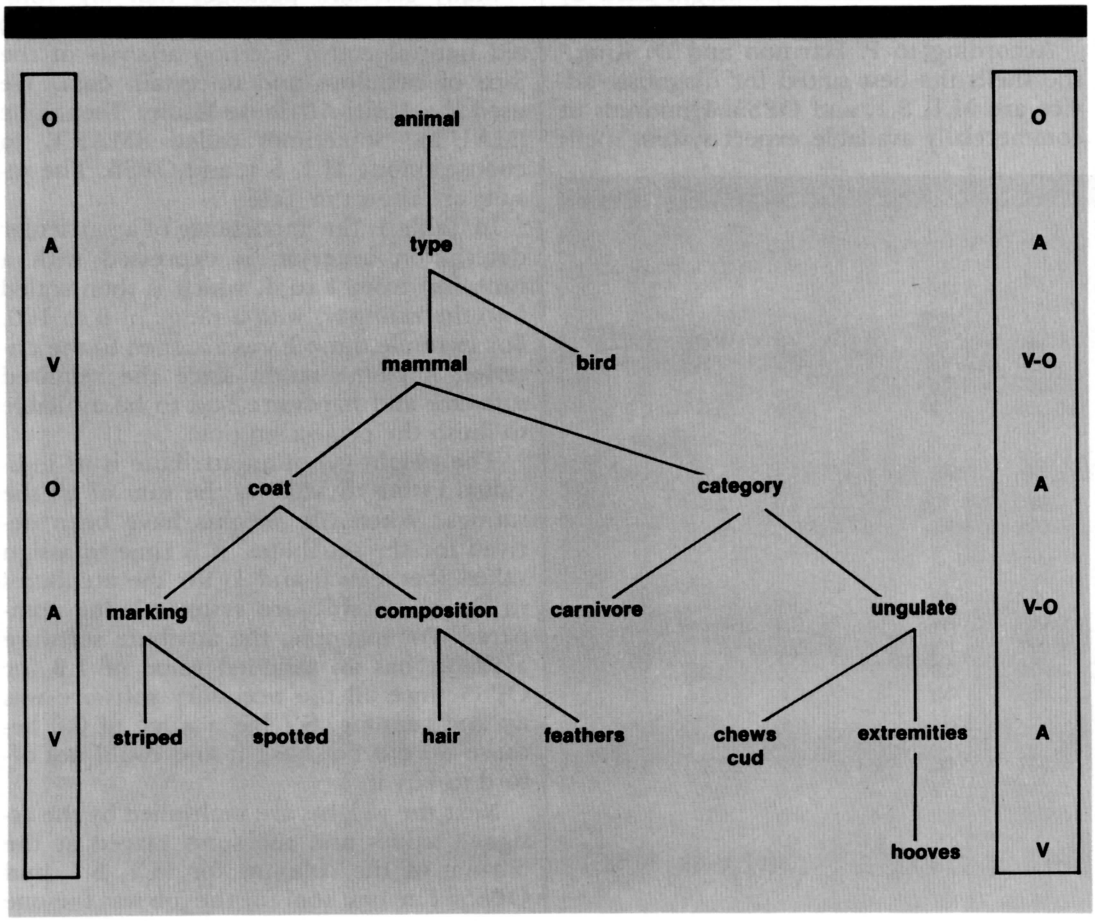


FIGURE 2.
Object-attribute-value hierarchy.

rule, reducing the number of rules.

M.1 automatically handles the answer "unknown." In OPS5, we had to specify in each rule how "unknown" should be handled. The inference engine of M.1 uses certainty factors. OPS5, on the other hand, does not provide a mechanism that deals with uncertainty. Probability calculations can be done in OPS5, but this is cumbersome. However, DCD, like most expert systems, did not need certainty factors. In fact, in one of our expert system classes at the University of Arizona, Tucson, we found that only six of the 25 student-generated systems used certainty factors.

QUANTITATIVE SELECTION

We have helped build over 80 expert systems for class projects, master's theses, and commercial ventures. We are often asked why we chose a particular expert system shell. Often our qualitative answers do not satisfy the questioner. Therefore, for the DCD project, we provided a quantitative answer to this question.

No one-to-one match exists between problems and software tools. However, conceptions of particular types of problems are the same for several different domains. These conceptions are commonly referred to as consultation paradigms. Examples of consultation paradigms are diagnosis/advice, planning, and design. Diagnosis/advice is the consultation paradigm used by DCD.

According to P. Harmon and D. King,[6] the shells the best suited for diagnosis/advice are M.1, S.1, and OPS5. Hundreds of commercially available expert system shells exist; we did not consider every one for DCD, but restricted ourselves to these three. The following discussion illustrates a technique and is not meant to advocate a particular shell.

To select the shell for our data-communication problem, we concentrated on the following criteria (Table 1):

- Representation: The knowledge that enables us to solve data-communication problems must be formulated with *if-then* production rules. The inference strategy used should be forward-chaining since all necessary data are given at the beginning of the inference process. The tool to be selected must provide an approximate solution when some information is missing or the answers to questions are unknown.
- Implementation: When this project began, available hardware included several personal computers, many PDP 11s, and a VAX. Available software included OPS5, M.1, and ROSIE.
- Interfacing: Questions asked by the system should be displayed on a screen and answers entered on a keyboard. Solutions to a problem must be written to a data file. The knowledge base must be built and modified with a tool-independent text editor.
- Support: Documentation should be readable with easy-to-understand examples. References to past tool applications and their level of success should be available.

Many software packages that are commercially available (such as Expert Choice[4]) aid multiobjective decision analysis in the face of nebulous and uncertain data. We used the Multi-Attribute Utility Technique (MAUT),[7] sometimes called SMART, to choose among M.1, S.1, and OPS5. The results are shown in Table 1.

In Table 1, the importance of a particular description criterion is expressed with a rank (*ra*) from 1 to 4, which is then scaled into the rating (*r*) with a range of 0 to 100. For example, rank 1 was assigned to the criterion implementation since the required software and hardware had to be available to finish the project on time.

The weight (*w*) of an attribute is its individual rating divided by the sum of all the ratings. When the weights have been derived for the attributes, it is time to assign values (between 0 and 1) for the attributes to the three software systems being compared. For example, the attribute software available has an assigned value of 1.0 for OPS5 since all the necessary software was up and running. S.1 has a value of 0.0 because we did not have it and could not afford to buy it.

Next the weights are multiplied by the assigned values and the sums placed at the bottom of the columns for M.1, S.1, and OPS5. The best tool for the job was the one

TABLE 1. Criteria and attributes used by the Multi-Attribute Utility Technique to help select an expert system shell; w—weight, r—rating, ra—ranking.

CRITERIA *and* attributes	w	r	ra	M.1	S.1	OPS5
REPRESENTATION						
Facts	0.064	78	3	0.9	0.9	0.9
if-then rules	0.065	80	3	0.9	0.9	0.9
Inference strategy	0.062	76	3	0.1	0.1	1.0
Uncertainty	0.059	72	3	0.9	0.9	0.7
IMPLEMENTATION						
Software available	0.080	98	1	1.0	0.0	1.0
Hardware available	0.820	100	1	1.0	0.5	1.0
Data base access	0.078	96	1	0.5	0.9	0.9
INTERFACING						
User display	0.074	90	2	0.8	0.8	0.8
Explanation facilities	0.069	84	2	0.8	0.8	0.5
Help functions	0.070	86	2	0.5	0.5	0.5
Editor	0.072	88	2	0.5	1.0	1.0
Debugging aid	0.067	82	2	0.8	0.8	0.7
SUPPORT						
Documentation	0.057	70	4	0.7	0.6	0.8
Courses	0.050	62	4	1.0	1.0	1.0
Applications	0.049	60	4	0.8	0.8	0.8
SUMMATION	—	**1222**	—	**0.689**	**0.685**	**0.834**

```
4.4.1. Initializing /etc/fstab

Change into the directory /etc and copy the appropriate file from:
                    fstab.rm03
                    fstab.rm05
                    fstab.rm80
                    fstab.ra60
                    fstab.ra80
                    fstab.ra81
                    fstab.rp06
                    fstab.rp07
                    fstab.rk07
                    fstab.up160m (160MB up drives)
                    fstab.up300m (300MB up drives)
                    fstab.hp400m (400Mb hp drives)
                    fstab.up (other up drives)
                    fstab.hp (other hp drives)
to the file /etc/fstab, i.e.:
                    #cd /etc
                    #cp fstab.xxx fstab

   This will set up the initial information about the usage of disk
partitions.
```

LISTING 2. Instructions for installing fstab.[2]

with the largest sum, in this case OPS5. (Please note that the ratings, weights, and assigned values are subjective; they were chosen by Senn. The other authors would have assigned different values and therefore might have drawn different conclusions.)

USER SATISFACTION

We found it easier to install a UNIX operating system using Cogito than the UNIX reference manuals because with Cogito the information presented to the user was tailored to the specific application. For example, in the disk-definition state, the file '/etc/fstab' must be created. Assume that the boot disk is a 'AMPEX 300M' connected to uba0 at drive 1. This implies that the disk address is 1. Also assume the user's name is 'Pat' and that Pat is ready to install 'fstab'. Since the instructions given in the manual are generic, the installer must relate the general instructions to the specific application. Listing 2 is a copy of a relevant section of the manual. Listing 3 is a copy of Cogito's instructions.

Cogito has remembered information the user gave it a half-hour ago: that the user's name is Pat and the disk is an AMPEX 300M. Cogito has used this information to make its instructions specific. Cogito's instructions are personalized, relevant to the user's task, clear and complete.

LISTING 3. Cogito's instructions for installing fstab.

```
# cd /etc
# cp fstab.up300m junk
# vi junk
(Edit the file, Pat.
  a. Add the line '/dev/up0b::sw::'.
  b. Give the global substitute command
     ':g/up0/s//up1/'.
  c. Save the new contents and quit the editor.)
# cat junk >> fstab
```

This example illustrates that the manual's instructions are incomplete and rely on previous and implicit knowledge. The instructions are incomplete because the swap partition '/dev/up0b', must be added to the file. They rely on previous knowledge because the user must recall that the UNIX standalone disk name for a 'AMPEX 300M' is 'up'. They rely on implicit knowledge because the user must somehow know that the disk address of the partitions must be changed from '0' to '1'.

TESTING

Testing Cogito was difficult. It was impossible to present the system with every possible combination of inputs and evaluate its outputs. The best test we devised was to let intended users try Cogito in many hypothetical situations. If the knowledge base was incomplete, then in some situations the advice given to the users should be incorrect.

Cogito was tested by the computer systems administrator and a professor of the Dept. of Systems and Industrial Engineering and the computer systems administrator for the Dept. of Computer Science at the University. All found Cogito's advice to be complete and correct.

In an effort to bolster our testing, we asked several nonexpert computer users to try the system. Their evaluations emphasized the difficulties they had using Cogito and making sense of its queries or output, and the extent to which they were able to fool the system with plausible (but nonsensical) inputs.

Recently, we gave Cogito a real test. Because of inadequate glue on the heads of our RA81 disk, we had to rebuild our operating system from the distribution tapes. Cogito helped us. We found a few omissions in Cogito's advice, but no mistakes, and completed the task in about 12 hours. It took us three months the first time we built the system, but we were way down on the learning curve then.

This experience has reinforced our belief that all expert systems are inadequately tested. No quantitative procedures exist for testing expert systems. Most tests merely involve running a few case studies; they do not exhaust all possibilities. For example, we are confident that Cogito works well for small VAX 750 systems, but we cannot be sure it will work as well for 730s or 780s.

We carefully planned our tests and evaluations for DCD. Because the DCD system was designed for users with little previous experience in data communication, we allowed juniors from our microcomputer class to

test and evaluate it. Thirty-three student teams (two members per team) participated in testing and evaluation.

It is impossible to test all the combinations of problems likely to occur in data communication. Therefore, only two typical test cases were given. System users were asked to use DCD to:

- Design a cable that would connect a WYSE terminal to a VAX computer and to give advice about setting up the WYSE terminal to make the interconnection work
- Design a cable that would connect a WYSE terminal to the Sytek local area network and to give advice about setting up the WYSE terminal and the Sytek port.

We specified two major subtasks in the evaluation process. First, we focused on the accuracy of the system. Was the advice given by the system sufficient to make the interconnection work? This assessment was trivial since the advice given by the system resulted in go/no-go situations. Second, the students evaluated the quality of the human-computer interaction. Was the system easy to use? These evaluation criteria should be directly proportional to future frequency of use of the system.

All the undergraduate students accomplished the tasks with an average time of 20 minutes, suggesting that the accuracy of the system was 100%. The students rated the quality of human-computer interaction as "user friendly." As a further test of the DCD system, a selected graduate student tried to connect a WYSE terminal to the VAX computer and a WYSE terminal to the Sytek net using only the manufactures' user and reference manuals. He succeeded after 1.5 hours—about three times longer than the undergraduate students who established the same connections by consulting the DCD system.

After this initial test, the DCD system was used by juniors at the University for five consecutive semesters. This continued usage proves its usefulness. However, DCD's usefulness is diminishing as our equipment changes, and none of our knowledge engineers have experience with OPS5 to update the knowledge base.

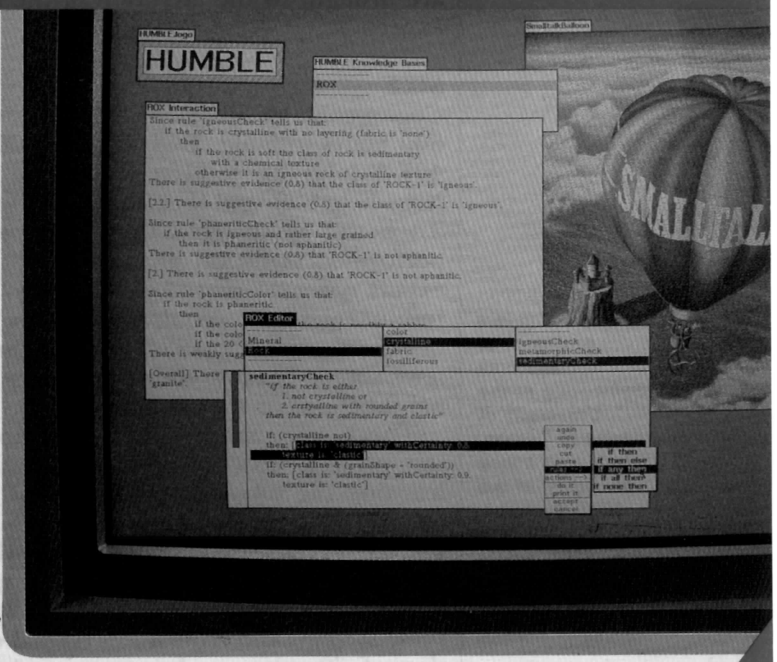
GENERAL-PURPOSE TESTING

The most difficult aspect of expert system design is testing. The traditional testing method is to have a human expert run many sample cases on the expert system. This consumes a lot of the expert's time and does not guarantee finding all mistakes. Conversely, brute-force enumeration of all inputs is impossible for most systems. Therefore we have developed a general-purpose tool to help debug knowledge bases without the intervention of human experts. We have

XEROX

ıle-Based Technology for nalltalk-80.

ly there's an expert system shell for the Smalltalk-80™ m. For those wanting a powerful way to combine rule- l technology with true object-oriented programming, IBLE™ provides the right tools for the task.
HUMBLE is an integrated expert system tool that runs tly in the Smalltalk-80 environment. It features both ward and forward chaining, a modular certainty m, an advanced user interface including a graphical ser, and a complete programmer interface.



e Bang for the Buck! HUMBLE costs far less than arable expert systems. Before buying, ask yourself: *I expand this tool to fit my particular needs? Are the es available if I need to change them? Will this tool rate with my own programs? Can I port the results of my to end user machines easily? at all?"* If you answered to any of these questions, then take a good look at IBLE.
HUMBLE is available for the Apple Macintosh, Sun-2 Sun-3 Series, Tektronix 4400 and 4310 Series, Apollo DOMAIN Series, and Xerox 1100 and 6085 Series work- ons.
HUMBLE runs in any License Version of the Smalltalk- stem. Knowledge bases can be transferred between any ltalk-80 version without modification.

For more information, contact the
Smalltalk-80 Marketing Manager
Xerox Special Information Systems
P.O. Box 5608
Pasadena, CA 91107
(818) 351-2351 **CIRCLE 17 ON READER SERVICE CARD**

XEROX®, Smalltalk-80, and HUMBLE are trademarks of Xerox Corporation.

* ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, September 25-30, 1988. San Diego, California

See us at OOPSLA '88*

tried to make this tool generic so it can work on any knowledge base, no matter which expert system shell is used.

The first component of the tool is a simple spelling checker with a preprocessor that replaces hyphens, underbars, and parentheses with spaces and removes terms specific to the shell being used, such as *legalvals* and *automaticmenu*. The second component of this tool is designed for backward-chaining shells. It collects all terms that appear in the premises (the *if* parts) of the rules, and flags as potential mistakes those that do not also appear in the conclusions (the *then* parts) of the rules or in questions or goal statements. It also traces every term appearing in a premise to the conclusion of that rule, then to the premise of another rule, and so on until a goal statement is reached. If it does not reach a goal statement, then a rule is missing.

The third component of this tool, which is not yet completed, acts like Tieresias[8] in pointing out rules that do not fit the pattern established by the rest of the rules in the knowledge base. For example, when we ran this component on our animal-discrimination knowledge base, it said, "Rule 15 is suspicious, because most rules that mention the animal's fur also ask if it has spots. Rule 15 does not."

The fourth component comes into play at run time. The firing of each rule is recorded. Rules that never fire and rules that fire for all test cases are probably mistakes and are brought to the attention of the human expert.

SIZE

Cogito contains 800 rules, 200KB, and took 800 hours to build. DCD contains 60 rules, 40KB (plus an interface program and a large help file), and took 120 hours to build. To better understand these numbers, we performed statistical analysis of 25 simple expert systems written by students in partial fulfillment of the requirements of a course in expert systems in 1987. (We call these systems simple because they did not use external functions or special hardware.)

The average student spent 100 hours on the project: 15% of this time was spent learning the subject, 10% interviewing the expert, 60% developing and debugging the knowledge base, and 15% testing the system. The average knowledge base contained 150 rules and required 32KB. From these systems and from a less formal evaluation of 60 other student-generated expert systems constructed over a three-year time span, we concluded that the number of rules (or knowledge base entries) is not a good indication of the complexity of a system.

However, we also concluded that in routine expert systems, novice knowledge engineers consume three to four hours per kilobyte of knowledge base. Sophisticated knowledge engineers may expend as much as five to 10 hours per kilobyte, because they know how to collapse similar rules into more general expressions and might be using external functions.

APPROPRIATENESS

We used M.1 and OPS5. One is primarily a backward chainer; the other primarily a forward chainer. We do not think much difference exists between the two. For our problems, we could have made either type of chaining work. After learning each of these expert system shells, Senn remarked, "It's just another language."

This remark has another implication about the knowledge-transfer process. Expert systems have made one step forward accompanied, unfortunately, by one step backward. While expert systems permit a higher level of abstraction, they also demand that the knowledge base remains tightly coupled to the inference engine. Even though the knowledge has already been captured in Cogito or DCD, converting this knowledge to another inference engine would be difficult. A fundamental breakthrough occurred when traditional programming languages became independent of the processor; a similar breakthrough must be effected within the expert system realm.

How can one identify a task appropriate for a PC-based expert system? First, there must be a human who performs that task better than most other people. Second, the task's solution must be explainable by the human expert in words without relying on pictures. Third, the problem must be solvable a 20-minute or even one-hour telephone conversation with the expert. If it would take a human two days to solve the problem, it is far too complicated for an expert system; if the human gives the answer in two seconds, it is too simple. Fourth, problems that involve determining one of many possible solutions are ideal candidates for expert systems, such as answering the question, "What disease does the person have?" Fifth, the problem must encompass uncertainty or inexactness in the data, or a diagram on paper would be superior.

The first four criteria help predict whether the expert system will be successful, whereas the fifth helps decide whether a complex expert system shell is needed or if some simpler tool would suffice. The animal classification system (Figure 2) is an example of a problem domain that has no uncertainty or inexactness. This task could be performed better using a diagram such as Figure 2 than an expert system on a PC.

Given these criteria, giving advice for

A. Terry Bahill is a professor of systems engineering at the University of Arizona in Tucson. He is also a vice president of the IEEE Systems, Man, and Cybernetics Society and an associate editor of *IEEE Expert*.

Pat Harris is a systems engineer.

Erich Senn is head of technical staff with Telekuhrs, Schweiz, Zurich, Switzerland.

bringing up UNIX on a VAX computer was an inappropriate task for a PC-based expert system. It cannot be done in a one-hour conversation with an expert. We estimate that it would take an expert one or two days to do this task (it took us three months to do it the first time!). Cogito's 800 rules filled two floppy disks.

We succeeded in making an expert system that worked, but it was hard work. A more powerful tool (such as KEE [IntelliCorp], S.1 [Teknowlege], ART [Inference Corp.], or Knowledge Craft [Carnegie Group Inc.]) would have been more appropriate.

However, helping a person connect a terminal to a computer is an appropriate task for an expert system. In the future, we think most computer equipment manufacturers will have expert systems (interactive textbooks) to help their sales personnel install their equipment. This will insulate their research engineers from mundane questions and build confidence in the sales force, which will then be able to answer difficult questions about their product.

The two real-world expert systems discussed here were designed to perform certain tasks, and they did them; so they were successful. However, the more important aspect of these endeavors was that we learned a great deal about making expert systems. We learned that an expert system shell is not Merlin's long-lost magic wand. You cannot do things with expert system shells that you could not do in any other language; but you can develop a prototype a lot faster. We learned that expert systems can be made friendlier than conventional computer programs. But we also learned that testing and validating expert systems is difficult. AI

This research was partially supported by grant no. AFOSR-88-0076 from the Air Force Office of Scientific Research.

REFERENCES

1. *UNIX Programmer's Manual Reference Guide.* Berkeley, Calif.: USENIX Assoc., 1985.

2. *UNIX Systems Manager's Manual.* Berkeley, Calif.: USENIX Assoc., 1985.

3. Moller, R., A.T. Bahill, and L. Swisher. "The Development of ESIAC, an Expert System for Identifying Autistic Children," in *Proceedings of the International Conference on Systems, Man, and Cybernetics.* New York, N.Y.: IEEE, 1985, pp. 875-878.

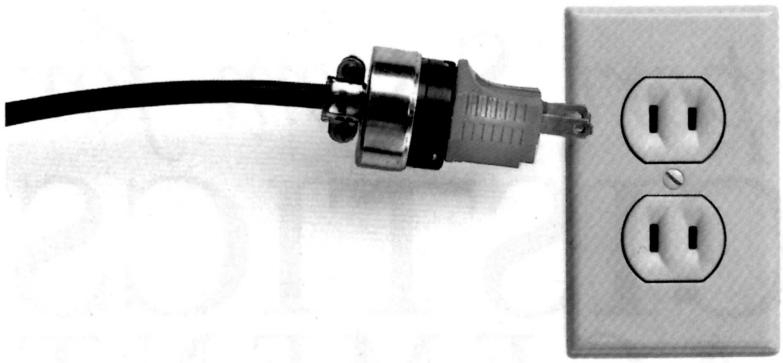
4. Forman, E.H., and T.L. Saaty. *Expert Choice.* McLean, Va.: Decision Support Software, 1986.

5. Saaty, T.L., and K.P. Kerns. *Analytic Planning: Organization of Systems.* New York, N.Y.: Pergamon Press, 1985.

6. Harmon, P., and D. King. *Expert Systems: Artificial Intelligence in Business.* New York, N.Y.: Wiley, 1985.

7. Edwards, W. "How to Use Multiattribute Utility Measurement for Social Decision Making." *IEEE Transcripts on Systems, Man, and Cybernetics* SMC-7: 326-340, 1977.

8. Buchanan, B.G., and E.H. Shortliffe. *Rule-Based Expert Systems.* Reading, Mass.: Addison-Wesley, 1984.



Now available in Turbo C,® Microsoft C,® JPI Modula 2,® and Logitech Modula 2.®

Turbo Expert. Now it doesn't take a genius to plug into Expert Systems.

For only $99.95, you can incorporate the power of a full-fledged Expert System into your TURBO PASCAL® programs. Seamlessly. Affordably. Finally. Actual Expert Systems, developed for simple use by any Turbo Pascal 4.0 programmer.

Take a look at all the features you suddenly have available with this single Turbo Pascal 4.0 Unit: The ability to create large Expert Systems, or even link multiple Expert Systems together. A powerful backward-chaining inference engine. Easy flow of both data and program control between Turbo Expert and the other parts of your program, to provide Expert System analysis of any database, spreadsheet, file or data structure. The ability to add new rules in the middle of a consultation, so your Expert Systems can learn—really *learn*—and become even more intelligent.

You also have the ability to create large rule bases and still have plenty of room left for your program, thanks to conservative memory use. You can link multiple rule bases, you'll be compatible with our Turbo Companion units, and you have available advanced features like date and time arithmetic, confidence factors, windowing, demons, agendas, blackboards, and more.

Imagine a single "EXE" file containing your user interface and data handling, *and* a full Expert System.

For a limited time, get a FREE copy of our Turbo SnapShot graphics package worth $79.95. We'll give one away with every copy of Turbo Expert sold between now and September 30. This package will let you capture graphics images from other programs and use them in any Turbo Pascal program.

You can even convert images from any CGA or EGA format to any other.

On top of all that, Turbo Snapshot has routines for graphic gauges and dials as well as mouse support. You'll have all you need for a sophisticated graphics front-end for your Expert Systems—free.

Call for more information or to order, (317) 876-3042. Software Artistry Inc., 3500 Depauw Blvd., Suite 2021, Indianapolis, IN 46268. Include $5.00 for shipping and handling.

SOFTWARE ARTISTRY

CIRCLE 18 ON READER SERVICE CARD