EXPERT SYSTEMS:
A Computer Consultant Software Application for the 90's

P.L. Adams¹, G.L. Woodard¹, A. Williams¹, Dr. T. Clarke¹

1. NC A&T State University, Greensboro, NC

Abstract
At the present time vendor-saturated markets have forced the drop in computer retail prices allowing a large percentage of today's households to have a computer or enter the market for one. Quite often these consumers call back to support hotlines for help in everything from getting started to troubleshooting various problems associated with computers. In many cases support personnel are swamped with angry, frustrated customers wanting some answers. We have proposed to alleviate problems of overload and inaccessibility by developing an expert system that would emulate a computer consultant to on-line and potential on-line customers. Theoretically, this system could be used by support personnel or by a user interfacing directly either by terminal or modem. This paper will describe our approach to solving this problem as well as the design of an expert system.

I. Introduction to Expert Systems

The line dividing intelligent from nonintelligent systems keeps shifting as computers and their software become more capable. When computers were new 40 years ago and could do little more than arithmetic functions, some called them giant brains, implying intelligence. Today, 40 years later, we do not regard even the largest of computers with processing power exceeding 4 or 5 BIPS as intelligent. In fact, intelligence is not system dependent but program dependent. Thus, an expert system is a computer program designed to emulate the problem-solving capabilities of an expert in a particular domain.

The meaning of intelligence is broadly characterized by what human beings are able to do both mentally and physically. As human beings we have an intuitive feel for what that is. Looking at the human brain from the outside, we have considerable knowledge of what it can do. From the perspective of inside the brain, however, we have taken only a few steps toward actually understanding how it operates. We do not have enough knowledge at this point to implement the intelligence by simulating the detailed structures and functions of the brain artificially. Still, in the 40 years since John McCarthy coined the term "artificial intelligence," attempts at creating intelligent systems based on studies of particular problems and implementation of the facts and heuristics associated with these problems, have achieved some success. In the last few years, practical applications of expert systems have appeared. For example, XCON an expert system developed by Digital Equipment Corporation was used to configure their mainframe machines and MYCIN was used by doctors in assisting with disease diagnosis. The use of expert systems has even entered the world of games such as the popular chessmaster 2000, but not much farther than that. Yet as the research community explores the problems that these programs address, the resulting systems will no doubt come to seem commonplace and be deemed nothing less than intelligent.

The basic structure of an expert system (figure 1) consists of: (1) a knowledge base of facts and heuristics associated with the problem domain; (2) an inference engine for utilizing the facts and heuristics of the knowledge base in a non-procedural algorithmic fashion; (3) a working memory (temporary storage) for keeping track of the problem status, the input data for the particular problem, and the relevant history of what has been done thus far.

---

194
II. Project: A Computer Consultant System using PROLOG

Using the PROLOG language we developed an expert system that would answer questions that a user may have concerning computer hardware problems (and ultimately, software problems as well). The system uses an expert shell by McLaughlin and Stubblefield [1], to interface with the inference engine. In developing the knowledge-base (appendix A), we focused not so much on the problem to be solved but on strategies and methods used by expert consultants in solving the problem. We found that expert system technology is most suitable when either (1) finding any solution is difficult due to partial input, or (2) the criteria for an optimal solution could be stated but no algorithm exists for implementation.

In developing our expert system we had to make sure that two by-products resulted:

- **Users** would be able to understand the program’s temporary and final results throughout the session, and
- **Human experts**, when consulted, could see and understand how their knowledge related to rules and facts assembled into a program’s knowledge-base.

We found that for a protocol of this sort we could obtain better results by using a narrow domain of problems. After narrowing down the problem domain we constructed a truth table (Table 1) to verify the expert system’s answers to be accurate and correct. There were some cases, such as the “Drive System” problem where a different set of user-responses would generate the same conclusion thereby making the truth table valuable in detecting these anomalies. With this case and other similar cases provisions had to be made in the knowledge-base to counteract such problems. The problem chosen for our test case involved a possible bad disk drive on a computer system. We, first, had to determine the sequence an expert would use in solving the bad disk problem, and second, implement these sequences into the knowledge-base. Because the inference engine, when searching for a rule or fact to satisfy the user-generated query, stops at the first match it finds in the knowledge-base we had to make sure the first, second, and third, etc. stops would correspond to the steps an expert would take when solving the same problem. For instance, to determine that the computer itself is bad one must first check that the drive system is not faulty, the power is connected, the monitor is turned on etc. One wouldn’t automatically come to the conclusion that the system needs repair without first checking other options. With a case like this, the last rule to be matched should be the one stating that if all other rules fail then the problem must be a bad computer.

III. Problems Encountered in Design and Implementation.

When setting up the design of our expert system, we initially intended to use a natural-language interface to our knowledge-base. We felt that such an interface would improve the attractiveness of our expert system by increasing the scope of acceptable input beyond syntactically sugared formal queries by allowing the user to ask questions using the English (or natural) language. We found that in order to effectively design a good Expert System, a natural-language interface was not just attractive but critical for our project. We feel that in order to effectively utilize such a system it would have to be as user-friendly as possible (which would ultimately minimize training for the system) and it would have to produce precise explanations and clarifications when needed.

Because natural language processing is an A.I. problem in itself, incorporating it into our protocol proved to be a formidable and unattractive task. Although with enough time and effort a natural language interface of moderate performance and ability would be feasible and should prove to be well worth the trouble.

Implementing the logic of a real expert initiated some problems during the coding phase. Several truth tables were used to help solve the logical problems encountered.

IV. Choosing A User Interface Language.

The user interface is defined as an interface that provides the means by which users interact with the expert system. The user interface accepts entries, displays data to the user, and may provide graphics, rule editing, and other capabilities to help create and improve the quality of the knowledge-base and its interaction with users. User interface facilities may include windowing, graphics, query features, and others.

For our enhanced expert systems interface we incorporated the skills of Mr. Adrian Williams, a programmer from North Carolina A&T State University. C language was decided upon as the user-interface language.
because of its ability to interface at very low levels while providing exceptional creativity from a high-level standpoint, and because of its extreme popularity in the programming environment. C provides a structured skeleton without limiting the creativity of the programmer, while consistently producing extremely fast and efficient executable programs.

V. Computer Consultant User Interface

The computer consultant must have a way to communicate with the user, therefore an interface is needed. PROLOG is an excellent language for decision making, and other types of logical tasks, but as a standard it is not a suitable programming language for advanced input and output tasks. The purpose of advanced input and output tasks is to make an interface that is more user friendly as well as attractive. Examples of languages that do more advanced input and output functions are BASIC, Pascal, COBOL, C, and many others. For our project C was chosen because of portability across multiple machines.

The program interface will serve as the link between the expert system and the user. The interface will be designed to accept normal command entries and these entries will be translated to PROLOG commands. The derived PROLOG commands will then be sent to the PROLOG program. The program will write its results out to a file for the interface to read, translate, and output the results of the command. The purpose of this is so that the user may use the expert system without prior knowledge of the PROLOG language.

The interface allows the expert system to be easily maintained, because changes can be made to the PROLOG program without the user knowing. The user, once comfortable with the system, will hardly notice any changes due to upgrades. Also since the user interface is separate from the expert system, the interface itself may easily be modified to improve user friendliness, or easy upgrades without loss of information.

Another advantage to this type of interface is that it may also be linked to other knowledge-bases (expert systems). Therefore it lends the expert systems to easy expandability. One of the future projects for this computer consultant expert system is to link it to a knowledge-base for natural language, so the user can type in plain sentences and get the desired information.

The user interface is designed for advanced output with windows, for user friendliness. However, should the computer or terminal be unable to handle the advanced input and output then the program will set itself up for standard input and output, while trying to retain its ease of use. This type of design will allow for more portability.

The interface will communicate with the user by prompting the user for a command or query. The user will then type a command or select from a list of choices. The interface will translate the command into a PROLOG statement.

VI. Conclusion

In concluding this paper it is important to note that this is
an ongoing project. At present we are hard at work
developing a new expert shell to interface with all
enhancements. While we still have much to learn, this
research has increased our understanding and knowledge of
artificial intelligence and the role it will play in the world
beyond the year 20013. The question that will be asked in
the future: "Is it M A N or M A C H I N E . . .," must now
be researched. It is toward this end we continue our search
for computer intelligence.

Appendix A Knowledge-Base
(Facts and Rules)
go(X) :- solve(correct_prob(X)).
go(X) :- init,write('again'),nl,go(X).

rule((correct_prob(Advice) :- bad_component(Y),
correct(Y,Advice)),100).

rule((bad_system(computer) :-
not(boot_up),good_drive_system,not(drive_reads),
not(bad_system(drive_system)),80).

rule((bad_system(drive_system) :-
not(drive_reads), not (drive_writes), not(good_drive_system)),90).

rule((bad_system(drive_system) :-
not(drive_empty),drive_closed,not(good_drive_system),
drive_keeps_running),60).

rule((bad_system(drive_system) :-
not(drive_empty),drive_closed,not(good_drive_system),
drive_does_not_run),50).

rule((bad_system(bootup_disk) :-
not(boot_up),drive_reads,drive_writes,good_drive_system,
not(drive_keep_running),not (drive_empty),drive_closed),80).

rule((bad_component(computer) :- bad_system(computer),
not(drive_empty),drive_closed),50).

rule((bad_component(disk_drive) :- bad_system
(drive_system), not(drive_empty),drive_closed),50).

rule((bad_component(diskette) :- bad_system
(bootup_disk)),50).

rule((correct(computer,'your computer is not working
correctly - it needs repair')),100).
rule((correct(disk_drive,'disk_drive is bad, replace
disk_drive')),100).
rule((correct(diskette, diskette is not a bootup or diskette is
unusable')),100).

% The below "askables" should correspond to the kinds of
questions that will need to be asked the user. 

askable(boot_up,['Does the computer boot up?']).
askable(good_drive_system,['Is your drive system good']).
askable(drive_reads,['Does the drive read']).
askable(drive_writes,['Does the drive write']).
askable(drive_keep_running,['Does the drive keeprunning']).
askable(drive_does_not_run,['The drive does not run - is this
correct']).
askable(drive_closed,['Is the drive closed']).
askable(drive_empty,['Is the drive empty']).
askable(power_on, ['Is the power turned on to your monitor']).
askable(cable_connected, ['Is the monitor's cable connected
to the computer's monitor port']).
askable(controls_adjusted, ['Are the monitor's controls
adjusted correctly']).
askable(good_monitor, ['Is your monitor old']).

References
Intelligence and the Design of Expert Systems,