A methodology for building expert systems

by HOWARD HILL
Knowledge Based Systems, Inc.
Oakbrook Terrace, Illinois

ABSTRACT

This paper provides a methodology, or process, that can be used to construct expert systems. The methodology is suited primarily for building troubleshooting, advisory, and diagnostic expert systems; however, it also can be useful for building other types of systems.

INTRODUCTION

This paper defines a methodology, or process, that can be used to build expert systems. This rather informal paper is intended for managers or engineers and programmers unfamiliar with expert systems. Therefore, the methodology described is practical, not theoretical. The thesis of this paper is that it is possible to build working expert systems in significantly less time and for less money than is generally recognized, and this paper explains how it can be done.

Generally, most of the effort required to build an expert system is in gathering and organizing knowledge from the domain expert. Existing knowledge engineering approaches are costly, time-consuming processes; they require much effort by both the domain expert and the knowledge engineer.

Most of the time in building an expert system is consumed by the itertative process of traditional knowledge engineering. A prototype system is built, tested, found to be wanting, and discarded. Information gleaned from this process is used to build yet another prototype which in turn is tested, found to be wanting, and ultimately discarded. This process continues until a system is developed that seems to work.

The traditional process of knowledge engineering can be improved by applying a concept software engineers have known for some time: it is much easier to do the job right the first time than to do it over and over. The cost of fixing bugs during the requirements phase of a software project is orders of magnitude less than the cost of fixing the same bugs after a product has been released to customers. The same principle applies to building expert systems. The development process can be reduced to only one iteration by first defining what the system should do, collecting all knowledge at one time, choosing the best knowledge representation from the start, and knowing what performance values the system must achieve to be accepted.

Many of the conventional problems often present in knowledge engineering vanish when this one-iteration approach is applied. For example, the problem of paradigm shift occurs when acquired knowledge exceeds the limits of a chosen representation. This problem is eliminated by selecting a knowledge representation paradigm after all knowledge has been collected.

Another cost present in building expert systems is the cost of the domain expert's time. Good domain experts are expensive; they have little time at best. Any good knowledge engineering methodology must minimize the use of their time.

FOUNDATIONS

There are a few key principles we use to construct this new methodology: problem decomposition, target system decomposition, and locality.

First, all people know far more than they can tell. People do not have complete access to the processes they use to solve problems. Indeed, as people acquire more expertise, they typically find it harder to state the reasoning and knowledge they used to solve problems. Further, if they attempt to do so, the offered reasoning model and knowledge often is incomplete. This situation is called the paradox of expertise.[1] Consequently, some authors advise prospective knowledge engineers to be wary of a domain expert's advice, and perhaps to seek less experienced domain experts.

Thus, much of the difficulty in knowledge engineering is caused by the difficulty of experienced domain experts to elucidate the methods and knowledge they use to solve problems. If a knowledge engineer can help with this problem, the cost of building an expert system will be reduced.

Although most expert systems work by modeling a domain expert's reasoning process and knowledge, this does not have to be the case. Commonly used knowledge representations, for example, do not mimic human thought processes. No expert I know thinks in terms of production rules, yet this is one of the most popular and successful knowledge representations used in expert systems today.

Second, the solution strategy used in an expert system also does not have to be one that matches a domain expert's problem solving technique. Once the requirement for an emulative solution strategy is relaxed, a rather mechanical knowledge acquisition process can be used to develop a solution strategy that will be used by the expert system. This will speed up the early stages of the development process.

The strategy of "divide and conquer" is well known in computer science. Using this strategy, problems are decomposed into separate subproblems, and the solution of the complete problem is obtained by combining the solutions to the subproblems. This process is useful for some problems because it can be much easier to solve many small subproblems than it is to solve an original problem.

Diagnosis, advisory, and troubleshooting problems usually can be decomposed using this technique. In such cases, the divide and conquer principle is applied recursively to a problem until the problem is broken down into atomic subproblems. These atomic subproblems represent the lowest level of problem that an expert system is designed to solve.

For example, an expert system designed to give advice about loans could have a problem decomposed into giving advice about commercial loans and retail loans. These two subproblems would be further decomposed into giving advice about various specific types of commercial and retail loans (e.g., asset-based commercial loans). The decomposition process would stop when the loan types being considered were the basic, or fundamental, types that a system's user would reasonably consider.

Further, not only can the expert system problem be decomposed using divide and conquer, the target system that the expert system is designed to troubleshoot, diagnose, or give advice about, can also be decomposed into many atomic subsystems.

An automobile can be considered to be a collection of subsystems such as an engine, wheels, drive train, body, and frame. Each of these subsystems also can be decomposed into subsystems. For example, some subsystems of a drive train would be the transmission, drive shaft, differential, and rear axles. This process would be repeated until a complete hierarchy consisting of the smallest meaningful subsystems for an automobile was obtained.

The third principle on which the methodology is based is the principle of locality. That is, if a target system has been properly decomposed, then separate atomic subproblems will be associated with each atomic subsystem. As a result, it should be possible to recursively decompose large target systems into hierarchies of atomic subsystems with one or more atomic subproblems associated with each atomic subsystem.

For diagnosis, advisory, and troubleshooting expert systems, the process of recursive decomposition of both the target system and the problem usually can be done by a knowledge engineer with only a minimal amount of help from the domain expert.

If presented with the complete decomposition of a target system, a domain expert will find it easy to elucidate the methods he or she used to solve the subproblem. Given a complete hierarchical decomposition of the target system and the problem, it is possible for a domain expert and knowledge engineer to elucidate the methods, that is, the knowledge and reasoning strategies, needed to solve every atomic subproblem. Therefore, since the solution to a problem is the sum of the solutions to all its atomic subproblems, it follows that the domain expert and knowledge engineer can obtain through target system and problem decomposition, the knowledge and reasoning strategies necessary to solve the complete problem.

METHODOLOGY

This section describes a methodology that can be used to build troubleshooting, advisory, and diagnostic expert systems.

Define System Objectives

First, it is necessary to determine what a proposed system will do. The system's function should be specified in a one- or two-page document that clearly states the objectives of the finished system. Documenting a system's objectives requires the cooperation of the system's end users, the domain expert, and the knowledge engineer.

Define Subsystems

After the problem that the expert system must solve is stated, the knowledge engineer uses the divide and conquer strategy to decompose both the target system and the problem into atomic subsystems. Each subsystem is then broken down to the lowest level that the expert system should address. An automobile-mechanic advisory system, for example, would not need to address the theory of condensed matter to advise a mechanic about changing a power-steering unit. Often, existing documentation can be used as a knowledge source for such a step.

Next, a knowledge engineer and the domain expert should identify every possible subproblem that can occur in each atomic subsystem. This step requires a series of interviews with the domain expert. At the end of this step, the problem should be completely decomposed into atomic subproblems, and each subproblem related to an atomic subsystem. Typically, every atomic subsystem will have several different subproblems associated with it.

Create Cause Tables

The domain expert must identify the causes of each atomic subproblem and the symptoms associated with each cause. Symptoms fall into two categories: those that increase the confidence that the cause is present, and those that rule out the cause. The end result of this step is a series of tables, one for each cause. Each table should list the subproblem to which a cause is connected, the subsystem to which the subproblem relates, and the symptoms that are associated with the cause. If more than one domain expert is available, then a Delphi technique can be used to increase the accuracy of the cause tables.

Write Knowledge Engineering Document

Next, the knowledge engineer writes a knowledge engineering document. This document contains the cause tables, the complete hierarchical decomposition of the target system, and a complete list of subproblems together with their relationship to the subsystems.

Perform Parretto Analysis

The knowledge engineering document contains all the domain knowledge needed for the system. However, because many of the possible subproblems and their causes stated in the tables actually may never occur, Parretto analysis is used to screen the data for plausibility. The basic idea behind Parretto's principle is that only a few of the potential causes account for most problems encountered. This is the basis for the 80/20 rule which states the 80 percent of the problems occur due to about only 20 percent of the causes.

In some systems (e.g., medical expert systems), it is necessary to include all potential causes, however unlikely. In such cases, a Parretto analysis is not needed.[2]

The Parretto analysis is accomplished best by analyzing historical problem records and using the data to weed out possible causes that (1) have never resulted in a problem and (2) are expected never to result in a problem. The assistance of a domain expert is necessary for this analysis.[1]

However, in most situations a reliable set of historical problem records will not exist. Therefore, it usually is necessary for a domain expert to estimate the likelihood of problems occurring as a result of each cause identified. Because people typically are not accurate at estimating probability, the domain expert should rank each cause according to the following estimates:

1. Almost always causes problems.
2. Commonly causes problems.
3. Occasionally causes problems.
4. Rarely causes problems.
5. Possibly can cause problems but there is no evidence of problems related to this cause.
6. Problems related to this cause will never occur.

After each cause has been analyzed, the Parretto analysis proceeds by plotting the ranked list of causes versus the likelihood of each cause resulting in a problem. The plot will take the form of a monotone decreasing curve. If the curve has a definite knee, and the knee occurs low enough, then causes below the knee can be eliminated safely from the knowledge engineering document. If no knee is found, then causes with a zero likelihood of resulting in problems can be eliminated safely.

The outcome of the Parretto analysis is a ranking of the problem-likelihood related to all possible causes in the knowledge engineering document together with a cutoff point that specifies the problem-likelihood limit for causes that will be included in the system. Only data associated with causes above the cutoff will be included in the system.

Build Control Flow Model

Next, it is necessary to build a control flow model. Although the strategy the expert system will use does not closely mimic the strategy the domain expert uses, the overall strategy used by the domain expert is best to use in the expert system. The flow model is a high-level description of the steps the domain expert uses to solve the specified problem. The flow model will form the basis for the control-rule metaknowledge the expert system will need. A typical flow model consists of a flowchart or an equivalent design language specification of the steps the domain expert takes to solve the problem.

Verify Knowledge Engineering Document

After the flow model is constructed, it is necessary to verify and inspect the knowledge that will reside in the finished system. This step consists of carefully inspecting, on a line-by-line basis, each piece of information in the knowledge engineering document. Four issues must be considered. First, the knowledge engineer should verify that the proper decomposition into atomic subsystems and atomic subproblems has been achieved. Second, the engineer must ensure that the causes and the symptoms for each cause within each subproblem have been identified correctly. Third, the problem-likelihood for every cause within the document must be verified. Finally, the consultation flow model must be reviewed for correctness.

The inspected knowledge engineering document is a valuable asset. It documents exactly what knowledge will be included in the finished system, completely specifies the behavior of the system, and acts as a reference guide for maintainers of the system.

Define Shell Parameters

The next step is to decide which expert system shell parameters to use. This step consists of choosing a knowledge representation, an inferencing strategy, a user interface design, the knowledge debugging parameters, and deciding on the nature and strategy behind the system's explanations. Many good books exist that can help knowledge engineers choose shell parameters. Since all knowledge required in the system has been collected prior to this step, the process should be straightforward.

Code Knowledge Base

Once a shell[3] has been acquired, the engineer codes the knowledge contained in the knowledge engineering document into a knowledge base the shell can use. Generally, the cause table knowledge will be simple to code into either production rules or frames and will form the bulk of the knowledge base. The control flow-model knowledge will encode into metaknowledge that controls the order in which rules or frames are invoked.

Establish Test Cases

While the knowledge base is being coded, a set of test cases can be collected or constructed from scratch with the help of the domain expert. The test cases should be selected to reflect actual problems the system should be able to solve. Remember, the purpose of testing is to find errors—not prove correctness. Each test case, therefore, should be included because it has the potential to find an error in the knowledge base.

Building an expert system using the methodology described in this paper, only a few possible types of bugs can occur. The primary type of bug that could be found is the "insufficiency bug," an error caused by missing knowledge. Insufficient knowledge usually is results from using too high a threshold in the Parretto analysis. Another type of bug could occur due to incorrect partitioning of the target system into atomic subsystems. Incorrect partitioning causes interactions between the subsystems that the causes tables do not address. As a result, the system will not be able to solve some problems correctly. Insufficiency bugs are also a consequence of incorrect partitioning.

Many of the problems frequently encountered in traditional expert system development will not occur, or will rarely occur when this methodology is used. Such bugs include paradigm shift, contradiction, subsumption, incorrect knowledge, and bugs due to the inability to access knowledge, (i.e., a missing relation in a frame, or a rule conclusion that is neither a goal nor subgoal).

Test and Validate

After the system is coded, testing must be done. This consists of running the test suite through the system and observing the system's behavior. Because no human activity is perfect, bugs will exist in any non-trivial knowledge base. Testing and fixing these bugs is a necessary part of knowledge engineering.

When testing is complete, validation can be performed. Validation consists of comparing the accuracy of the expert system against a known "gold standard" for accuracy.

Several levels of gold standards can be used. One of the best is to build a validation test suite of cases by selecting cases from actual problems that have occurred in the past. Accuracy is established by giving these cases to a panel of human domain experts and measuring its solution accuracy as well as the extent to which it agrees on the solutions to each case. These numbers form a baseline against which the expert system's accuracy is measured.

Actual validation is done by running the calibrated validation test suite through the expert system and comparing the system's accuracy to the accuracy of the domain-expert panel. Although systems vary in accuracy, generally it is possible to achieve the same level of accuracy as the domain expert panel.

Once the system is finished, the knowledge engineering document and the system should be placed under change control. All changes to the system must be reflected in the knowledge engineering document.

CONCLUSION

I have used the methodology described in this paper to build several systems, and found it superior to the methods generally discussed in the literature. By concentrating on doing the knowledge engineering only once, and doing it right the first time, it is possible to build expert systems much more rapidly than is commonly supposed.

REFERENCES

1. Waterman, D. A. *A Guide to Expert Systems.* Reading, MA: Addison-Wesley, 1985.
2. Buchanan, B. and E. Shortliffe. *Rule-Based Expert Systems.* Reading, MA: Addison-Wesley, 1985.
3. Hayes-Roth, F., D. Waterman, and D. Lenat. *Building Expert Systems.* Reading, MA: Addison-Wesley, 1985.