AN OVERVIEW OF CURRENT TOOLS FOR DEVELOPING KNOWLEDGE–BASED EXPERT SYSTEMS

James H. Garrett, Jr.
Assistant Professor, Department of Civil Engineering
Carnegie Mellon University, Pittsburgh, PA 15213

James Garrett received his BSCE, MSCE, and Ph.D. from Carnegie Mellon University in Pittsburgh, PA. He joined the faculty at Carnegie Mellon University in 1990, after having spent three years on the faculty at the University of Illinois at Urbana–Champaign. He has been performing research in the areas of standards processing, object–oriented building modeling and neural network applications in engineering.

Summary

Knowledge–based expert system (KBES) technology was presented to the international structural engineering community in the early 1980's and experienced an exponential growth of application to Civil Engineering problems in the latter part of that same decade. The intent of this paper is to discuss what has happened since Fenves, Maher and Sriram described this technology in IABSE Periodica in 1985. Several more recent methods for building and using KBES are discussed: object–oriented modeling, model–based reasoning, case–based reasoning, and inductive machine learning.

1. Introduction

The purpose of this paper is to present a brief overview of some of the current methods and techniques being used to develop knowledge–based expert systems. Because these systems began as pure rule–based systems, this form is briefly described in the first section. Following this discussion, other more recent KBES technologies (or more general programming paradigms) are discussed. These technologies include: (1) object–oriented methods, which are now being used to represent the structured knowledge within a domain; (2) model–based reasoning, which is used to reason qualitatively about the solutions of complex problems; (3) case–based reasoning, which is used to reason from past solutions to form solutions to new problems; and (4) inductive machine learning, which is used to acquire more general problem solving knowledge from past solutions to problems. Together, these techniques provide increased capability for today's KBES in the areas of representation, inference, and knowledge acquisition.

2. Traditional (Rule–based) Knowledge–Based Expert Systems

Although the concept of knowledge–based systems is actually "old" by today's software time–scale (it came on the computer science scene in the early 1970's), it has taken the engineering community several years to understand and grasp the potential of this technology. In 1985, Fenves, et al., authored a paper within the IABSE Periodica describing the nature of, possible application of, and implications of this technology[3]. At that time, the technology was predominantly a rule–based technology, where they described the main components of a KBES as:

- a knowledge–base — a collection of IF–THEN rules that represent pieces or "chunks" of decision–making knowledge that are applied during the development of a solution to a problem;
- a context or working memory — a formal data structure for representing the initial information known about the problem as well as the evolving solution being developed through application of the rules in the knowledge base; and
- an inference engine — a pattern–matching program for applying the "chunks" of knowledge in the knowledge base to the problem (and its partial solution) described in the context.

A knowledge–based system has several key qualities that differentiates this form of computer program from its procedural counterparts that all stem from its separation of knowledge from its usage: transparency of the knowledge applied, transparency of the solution being developed, and a capacity for incremental development.

Because the rules are developed and represented as separate "chunks" of knowledge, each hopefully specifying the preconditions under which it is applicable, those pieces can be individually observed and understood thus leading to the claim of transparency of knowledge applied. In a procedural program, the pieces of knowledge used are inextricably intertwined with the control of these pieces of knowledge and thus one must look at the entire program to get a sense of the knowledge within.

Because the inference engine of a knowledge–based system searches over this set of rules and keeps track of what rules match the context at what points in the solution process, it is possible to reconstruct the evolution of the solution process thus leading to the claim of transparency of solution. A procedural program normally does not keep track of the path it followed during the solution of a problem and only issues a final answer.

Finally, because the knowledge that is represented within the knowledge base can be both general and much more specific and one can rely on the inference engine to "figure out" which to apply, it is possible to develop a knowledge base that consists mostly of general rules in the early stages of system development that is later augmented with more specific rules as the knowledge acquisition process matures. Thus, the system will perform at a reasonable level of proficiency, but not expertly, in the early stages of development. It will not be capable of more expert problem solving capabilities until the more specific, exceptional pieces of knowledge have been added to its knowledge base. This represents the concept of incremental growth capability. Most procedural programs do not allow for more than one level of problem solving knowledge.

3. Additional Methods and Technologies for KBES

Many successful applications of the more traditional rule–based KBES technology have been described in the literature and employed [1][9]. However, while developing these systems, and many others like them, the KBES community has come to realize several important limitations of the knowledge–based technologies of the last decade:

- knowledge is not homogeneous in structure and many different knowledge representation paradigms (frames, rules, procedures, formal logic, neural networks, etc.) are needed in order to adequately represent the problem solving knowledge brought to bear on a particular problem;
- knowledge is not homogeneous in level of abstraction and knowledge at various levels of detail is needed in order to develop preliminary, high–level solutions to problems, as well as more detailed solutions, at various stages in the problem solution process;

- knowledge acquisition is an enormously difficult, time–consuming, error–prone task, is the most important task in developing a knowledge–based system, and is one area in which tools for assistance are most needed; and
- most current knowledge–based systems are only static "snapshots" of the knowledge used in problem solving and must be made capable of self–modification over time and as more experience is gained.

Hence, KBES developers have looked toward researchers in AI and other fields of Computer Science to help them solve these limitations. There are several methodologies and technologies that are receiving a lot of attention and represent some of the advancements that will eventually find their way into mainstream knowledge–base system development: object–oriented knowledge representation, model–based (qualitative) reasoning, cased–based reasoning, and inductive machine learning. The following sections describe each of these emerging technologies in more detail.

3.1. Object–Oriented Modeling

Object–oriented methods are now being used for organizing and representing various forms of knowledge. The basic building block of an object–oriented representation is the *object* — a modular, self–contained collection of descriptive attributes and the methods (procedural or rule–based) for manipulating those attributes. Representation in an object–oriented environment first requires the description (declaration of attributes and methods) of the general types of objects that populate the domain (class objects), and then requires the generation of instances of the class objects to describe the particular entities being modeled.

In object–oriented representations, everything is an object. Objects can represent concepts, physical objects, processes, etc. In all cases, objects possess a set of attributes and relationships, both of which are represented as slots in the object. Attributes represent descriptive pieces of information about the object and may be represented by either a static value or a method, where a method describes how the value of an attribute is computed. Methods may also cause side–effects in other objects. Relationships, also represented using slots, represent links to other objects. For example, an object may have a "next–to" slot that is filled with the name of another object representing the fact that the two objects are next to each other. Other objects can access the attributes and relationships of an object by *sending a message* to the object that "owns" the attribute or relationship.

While, the most popular way to represent domain–dependent behavior is with production rules, they alone are inadequate for representing domain object definitions and their static relationships. It is common today to use an integrated approach to knowledge representation, where objects represent the natural structure of the physical objects and abstract concepts within the domain that are to be reasoned with, and rules are used to represent the decision–making knowledge (i.e., the derivation of object attributes that are conditional in nature). Objects provide a powerful foundation for a rule–based inference by providing:

- a powerful language for describing domain objects that can be used within rules,
- a set of inference mechanisms (inheritance and ValueClass/cardinality checking) that can automatically reach a lot of conclusions that are needed by a rule, such as class membership and default values, and
- a powerful and flexible language for defining the rules themselves.

The advantages of an object–oriented representation are: 1) the methodology greatly aids in structuring the knowledge; 2) it enables rules and procedures to be more generic making the

knowledge–base easier to understand (one can write rules based on a class of objects and have them apply to all subclasses); 3) it compartmentalizes the knowledge, thus reducing the complexity; and 4) through graphical display of attributes and relationships, the knowledge base is more understandable. For more information on object–oriented modeling, see [10].

3.2. Model–Based Reasoning

Model–based reasoning, also known as qualitative reasoning, provides developers of knowledge–based systems with a tool for assisting both knowledge acquisition and reasoning at multiple levels of abstraction. Model–based reasoning is based on the way in which humans initially develop causal models for, or initially reason with, complex systems. We do not start with the detailed descriptions of the causal relationships about the components of a system. Rather, we reason qualitatively about the way in which these components (or larger groups of components) interact to get a sense for how the system will behave. Hence, model–based reasoning first involves building a qualitative model of a complex system, consisting of a set of identified subcomponents and their qualitative causal interrelationships (e.g., A causes B to increase). Next, this model is used to: 1) qualitatively reason about effects given observations about existing symptoms (causes); 2) qualitatively reason about possible causes given observations about existing conditions (effects); or 3) developing more detailed causal relationships between causes and effects that can be used to predict values of quantities instead of just qualitative trends thus guiding the knowledge acquisition process. When the more detailed causal relationships are known, the qualitative model is not discarded, but remains as a more abstract description of the system which can be used to reason about the system in earlier stages of system development when many of the system details are yet to be determined. The existence of both qualitative and detailed descriptions of causal relationships thus provides the multi–level reasoning capability described as the second deficiency of current knowledge–based systems. There are many different directions being taken by researchers in this field (Harandi and Lange [6], Forbus [5], Kuipers [8])

In summary, such a model is useful for performing diagnostic causal reasoning. Once the set of components have been properly connected, it can be used to reason from causes to effects or from effects to causes. When an expert reviews the results from either of these reasoning exercises, he may discover inconsistencies, or the inability of the model to reason from opposite changes in inputs, and at that time may modify the causal equation for a component. In this way, the model is refined and thus also assists in knowledge acquisition.

3.3. Case–Based Reasoning

Case–based reasoning, also known as reasoning by analogy, provides developers of knowledge–based systems with an alternative to the traditional distillation of knowledge into a set of static rules. Case–based reasoning is based on the way in which experienced humans sometimes solve problems, i.e., by determining which of the myriad of past experiences is closest to the current problem and modifying the former solution to develop a solution to the current problem. Hence, the case–based reasoning paradigm takes a large collection of previous experiences, called cases, and determines which case is most similar to the current case. Having determined which case is most similar, and having identified which aspects are similar and which are different, the case–based reasoning system then transforms the past solution into one that is viable for the current situation. Having the ability to reason about the similarity of previous cases (and their solutions) with the current case being addressed and then to develop an analogous solution offers a dynamic problem solving paradigm that begins to address the fourth limitation of current knowledge–based systems (namely, their "snapshot" nature).

Kolodner describes three steps in performing a simple case–based inference: 1) recall a previous case that is similar to the current case; 2) focus on the appropriate parts of that previous case as it relates to the current situation; and 3) use the appropriate parts of the previous case to derive an appropriate solution to the current case [7]. More than one case may be employed in developing a solution to the current problem; different cases may be similar to the current situation in different ways and together the subset of past cases provides enough past knowledge to solve the current problem [7].

In order to make effective use of past experiences and to determine the degree of similarity between past cases and the current situation, it is not sufficient to only represent the final design solution. What is needed about past cases are: the goals and subgoals of the past problem, the constraints imposed on that problem at the goal and subgoal levels, the solution derived for each of these goals and constraints, and the rationale or plan behind the solution derived for the goals and subgoals. It is usually the case that the goal hierarchy, constraints and rationale are represented using an object–oriented representation such as that described in section 3.1. Kolodner defines a past case to be similar to a current situation if the past case has many of the same goals and most important constraints [7]. There is no currently agreed upon definition of similarity.

Once a past case has been identified as being similar to the current case under consideration, the next step is to determine what about the past case can be used in solving the current problem. There are several ways in which the successful solution of a goal from a past case can be employed in the development of a solution to a similar goal in the current case [7]: 1) the solution used in the past case can be directly transferred to the new case; 2) the solution used in the past case can be modified for the new case based on the differences in constraints and subgoals between the past and current case; and 3) the method by which the solution in the past case was derived can be transferred directly to the current case and used to derive a solution for the current case. Hence, using the design representations and plans of past cases, a design plan and solution for a new design problem can be constructed.

Case–based reasoning systems provide an alternative to the traditional, static rule–based systems. One can easily see that as the case–base is built up with descriptions of past experiences encountered and successfully, or unsuccessfully, resolved, the performance of the system will drastically improve — a characteristic that is also present in systems capable of learning, some of which are described in the next section.

3.4. Inductive Machine Learning

Learning is a generalized term denoting the way in which people (and computers) increase their knowledge and improve their skills. Machine learning has been a goal of AI researchers since beginning of AI, where researchers strived to understand the process of learning, and to create computers that can be taught rather than programmed. Two methods for machine learning are described below: induction from symbolic rules and neural network–based mappings.

Induction is one type of machine learning (of which there are many) that has received a lot of attention and presents valuable solutions for some of the limitations of current knowledge–based systems. Induction is the process of learning how to perform a task (and recording this knowledge in the form of if–then rules) by being presented with examples of how it should behave. The most common inference process used for learning from examples is *generalization*. We say that rule A is more general than rule B, if rule A applies in all situations that rule B does and then some more. For example, "5 cards that are all of the same suit is a flush" is more general than "5 cards that are all hearts is a flush. [2]" Often description A is more general than description B because description A places fewer constraints on any relevant situations for which the rule applies. Specific rules are generated from examples and then generalized by modifying the specific rules, while maintaining their validity

for the given cases. Thus, the representation chosen for the rule space is important and must be able to support generalization. For example, predicate calculus supports generalization by permitting the following: 1) turning constants into variables, 2) dropping conditions, and 3) adding conditions, to name a few of the operations.

There are currently several problems plaguing such induction systems. Some of the rules induced are incomprehensible. A knowledge engineer still plays a major role in inductive learning by instructing the system as to what attributes are important and what attributes are not. If the attributes are not filtered by a knowledge engineer, the noise might be so great (in the form of unusable, or extremely specific rules) that the program will never perform efficiently and at an expert level. The rules induced from a set of cases is very dependent on the scope of the test cases. If the test cases do not represent a broad set of situations, the rules will be incomplete and possibly over–generalized. The programming of a learning system lies in the selection of test cases or the breadth of the experiences fed into the learning systems. Hence, inductive learning systems will only work for domains where a large body of experience exists, and will not learn any principles not embodied in the experiences presented to them.

4. Summary

The purpose of this paper was three–fold: 1) to briefly review rule–based knowledge–based system techniques; and 2) to briefly describe some of the newer techniques being used to develop current knowledge–based systems. Due to space limitations, several other interesting subjects, such as machine learning and neural networks, were left out of this review. Many of the systems currently being developed are still using technology that is 20 years old. However, many others are being developed that incorporate various forms of machine learning, use more descriptive, flexible models of the domain, and make much greater use of past performance for improving their performance on new problems. While knowledge–based systems are still in their "formative years", the prospect of having a system that 1) acts as a knowledgeable colleague capable of providing advice when asked, 2) grows in capability as it experiences various problems and their solutions, and 3) serves to record in much more detail the evolution of problem solutions, is still very exciting and deserving of more research attention.

5. References

[1] *Proceedings of the IABSE Colloquium on Expert Systems in Civil Engineering*, IABSE, Bergamo, Italy, 1989.

[2] COHEN, P. R. and E. A. FEIGENBAUM, *The Handbook of Artificial Intelligence – Vol 3.*, Morgan Kaufmann, pp. 325–334, 1982.

[3] FENVES, S. J., M. L. MAHER, and D. SRIRAM, "Knowledge–Based Expert Systems in Civil Engineering", *IABSE Periodica*, Number 4, 1985, pp 63–72.

[4] FENVES, S. J., "Expert Systems: Expectations versus Realities", in [1], pp 1–19.

[5] FORBUS, K., "Qualitative Physics: Past, Present, and Future", in *Exploring Artificial Intelligence*, ed. H. Shrobe, Morgan Kaufmann, 1988.

[6] HARANDI, M. T. and LANGE, R., "Model–Based Knowledge Acquisition", Chapter 4 in *Knowledge Engineering – Vol 1. Fundamentals*, H. Adeli (ed.), pp. 103–129, McGraw–Hill, New York, 1990.

[7] KOLODNER, J. L., "Extending Problem Solving Capabilities Through Case–Based Inference," in Proceedings of the Case–Based Reasoning Workshop, sponsored by DARPA, May, 1988.

[8] KUIPERS, B., "Qualitative Simulation", *Artificial Intelligence*, Vol 29, No 3, 289–388, 1986.

[9] MAHER, M. L., ed., *Expert Systems for Civil Engineers: Technology and Application*, ASCE, 1987.

[10] RUMBAUGH, J., M. BLAHA, W. PREMERLANI, F. EDDY, W. LORENSEN, *Object–Oriented Modeling and Design*, Prentice Hall, 1991.