KNOWLEDGE ACQUISITION

Knowledge acquisition is the process by which problem-solving expertise is obtained from some knowledge source, usually a domain expert. This knowledge is then implemented in an expert system program, which can provide expert assistance to nonexperts when and where a human expert is not available.

Traditionally knowledge acquisition is accomplished through a series of long and intensive interviews between a knowledge engineer, who is a computer specialist, and a domain expert, who has superior knowledge in the domain of interest. This process is usually referred to as *knowledge elicitation* to distinguish it from the more general knowledge acquisition term.

Experience has shown that knowledge acquisition from experts is the most difficult, time-consuming, and costly part of

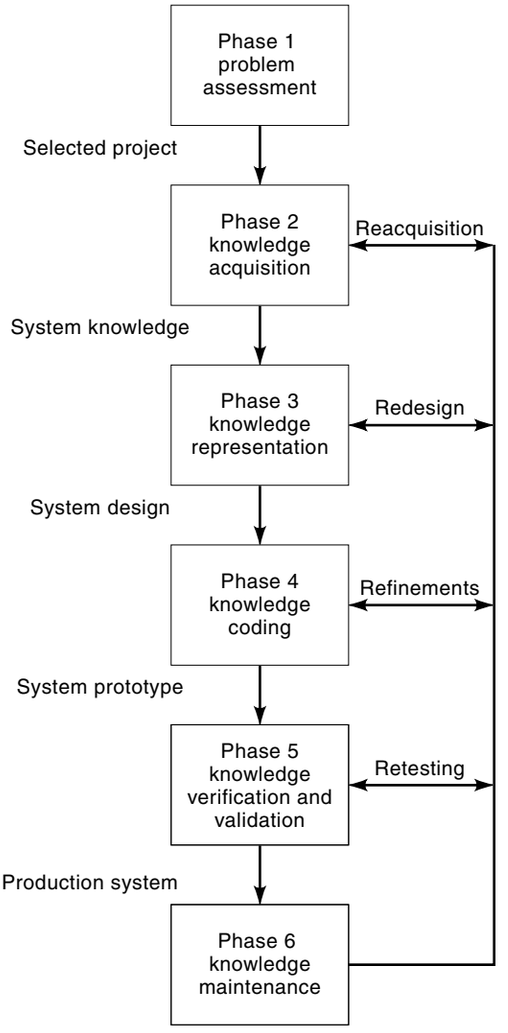

Figure 1. Phases of the knowledge engineering process. These phases can be used as a roadmap for developing an expert system. Although the phases appear sequential, there is considerable overlap and iteration in their execution.

developing an expert system (1). The difficulty of knowledge acquisition has stimulated research in developing machines that autonomously acquire knowledge without the assistance of humans. Although progress has been made in the area of automated knowledge acquisition, in the foreseeable future most of the knowledge for practical expert systems will be obtained through the interaction of domain experts and knowledge engineers.

THE KNOWLEDGE ENGINEERING PROCESS

Knowledge acquisition is an activity of a larger process used to develop expert systems, called *knowledge engineering*. The knowledge engineering process consists of a number of phases, each consisting of several tasks. Although knowledge engineering phases and tasks are usually shown in sequence, in practice they are conducted iteratively. Figure 1 depicts the phases of the knowledge engineering process. The following is a summary of the activities conducted in each phase:

Phase 1. Problem Assessment. This phase assesses the applicability and feasibility of an expert system solution to a particular problem.

Phase 2. Knowledge Acquisition. This phase involves the acquisition of knowledge from a domain expert and/or other sources of knowledge. It also involves interpreting, analyzing, and documenting the acquired knowledge.

Phase 3. Knowledge Representation. This phase involves the selection of a knowledge representation scheme and control strategy. Acquired knowledge is represented using the selected representation.

Phase 4. Knowledge Coding. This phase involves coding the knowledge using appropriate expert system development software.

Phase 5. Knowledge Validation and Verification. This phase ensures that the developed system performs at an acceptable level of expertise and that it correctly implements its initial specification.

Phase 6. Maintenance. This is an ongoing phase that corrects system errors and deficiencies. It also updates the system knowledge as the requirements evolve.

An interesting aspect of the iterative nature of the knowledge engineering process is its synergistic effect. Both the system and the development team improve their knowledge about the problem and how best to solve it as the development progresses.

DIFFICULTIES IN KNOWLEDGE ACQUISITION

Experience has shown that knowledge acquisition is a difficult, expensive, and time-consuming process. The major source of difficulty stems from a well-recognized fact in the field of cognitive psychology that eliciting knowledge from humans is an inherently difficult task (2). Humans are usually unaware of their mental processes when solving a problem (3). They may not be able to communicate their knowledge, not because they cannot express it, but because they are unaware of what knowledge they are using in their problem solving activities (4). Furthermore, humans provide explanation of their performance that is different from the way they actually perform their tasks (5).

Since most expert system projects rely on elicitation of knowledge from an expert by a knowledge engineer, many of the problems identified by cognitive psychologists are manifested. These problems include the following:

Experts may be unaware of knowledge used

Experts may be unable to articulate their knowledge

Experts may provide irrelevant, incomplete, incorrect, or inconsistent knowledge

Additional problems that add to the complexity of acquiring knowledge include the following:

Experts may not be available or may be unwilling to cooperate

Lack of well-defined knowledge acquisition methods

The complexities of dealing with a large number of participants with different backgrounds, different skills and knowledge sets, and using different terminology

The multiplicity of the sources of knowledge required for the system

The exponential growth in the complexity and interdependencies of knowledge with the size of the domain

The mismatch of the level of abstraction of knowledge between experts and computers

Potential interpersonal communication problems between the knowledge engineer and the expert

FUNDAMENTALS CONCEPTS OF KNOWLEDGE

Levels of Knowledge

Knowledge can be broadly classified into two levels: shallow knowledge and deep knowledge.

1. *Shallow Knowledge.* Surface level information that can be used to solve problems in very specific domains. Shallow knowledge is usually empirical and represents knowledge accumulated through experience of solving past problems. Although shallow knowledge can be easily represented by computers, it is limited in representing and solving problems of a knowledge domain, thus it is usually insufficient in describing complex situations.
2. *Deep Knowledge.* The fundamental knowledge about a problem represented by its internal structure, fundamental laws, functional relationships, etc. Deep knowledge can be applied to different tasks and under different situations. Deep knowledge is difficult to represent using computers, as it requires a complete and thorough understanding of the basic elements of knowledge and their complex interactions.

Types of Knowledge

In addition to the above two categories, knowledge can be classified by various types:

Declarative knowledge describes what is known about a problem. It is a descriptive representation of knowledge that includes simple statements that are either true or false. The factual statement "The sky is blue" is an example of declarative knowledge. Facts, concepts, and relations are typical examples of declarative knowledge.

Procedural knowledge describes how a problem is solved. It provides a step-by-step sequence of instructions on how to solve the problem. For example, "If the temperature falls below 50, turn on the heater." Rules, strategies, and procedures are examples of procedural knowledge.

Heuristic knowledge is a special type of knowledge that describes rules-of-thumb used to guide the reasoning process to solve a problem. Heuristic knowledge is acquired through extensive experience. Experts usually compile deep knowledge into simple heuristics to aid in problem solving.

Episodic knowledge is time-stamped knowledge organized as a case or episode. This knowledge can confer the capability to perform protracted tasks or to answer queries about temporal relationships and to utilize temporal relationships.

Meta-knowledge describes knowledge about knowledge. It is used to select other knowledge and to direct the reasoning on how to best solve a problem.

It is important to identify the type of domain knowledge to be acquired as different types of knowledge are best elicited by different techniques. In many situations, the domain knowledge consists of several types. In these situations, it is usually preferred to employ more than one technique to acquire the knowledge.

Sources of Knowledge

Knowledge may be obtained from a variety of sources. These sources can be divided into two main types: documented and undocumented. *Documented* sources include manuals, books, articles, reports, standard procedures, regulations, guidelines, pictures, maps, video, films, computer databases, and so on. *Undocumented* knowledge largely exists in human minds. Sources of undocumented knowledge include experts, end-users, and observed behavior.

PROCESS OF KNOWLEDGE ACQUISITION

The process of knowledge acquisition is a cyclical one. It begins with the collection and recording of knowledge, followed by its interpretation, analysis, and organization. Finally methods are designed for clarifying and collecting additional knowledge based on acquired knowledge. Figure 2 illustrates the knowledge acquisition process.

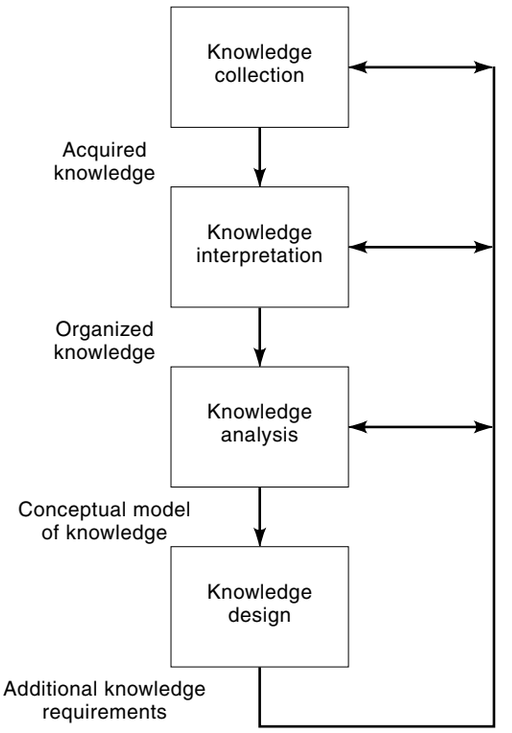

Figure 2. The knowledge acquisition process. The process is cyclic; information obtained from each cycle is used to design new ways to acquire knowledge.

Knowledge Collection

Knowledge collection is the task of acquiring knowledge from a knowledge source. Usually, this step requires significant interaction between an expert and a knowledge engineer. At the initial stages of knowledge collection, information obtained from the expert represents a broad overview of the domain and the general requirements of the expert system. Later stages of knowledge collection are characterized by their narrow focus, with emphasis on the details of how the expert performs the various tasks. Knowledge acquisition sessions are recorded and transcribed in preparation for interpretation and analysis.

Knowledge Interpretation

This task involves reviewing the collected information and the identification and classification of key pieces of knowledge, such as facts, concepts, objects, rules, problem-solving strategies, and heuristics. In early iterations of the cycle, the knowledge collected will be of a rather general nature. During later stages, different and deeper problem-solving knowledge will be uncovered.

Knowledge Analysis

This task takes the key pieces of knowledge uncovered during the knowledge interpretation phase and forms theory on the representation of knowledge and problem-solving strategies used. It requires assembling the acquired knowledge into related groups and storing them in the knowledge dictionary. The output of this task is a conceptual model of the domain knowledge that shows the information an expert system will require, the reasoning it will perform, and the sequence of steps it will take to accomplish its task. A variety of graphical techniques are typically used to develop the conceptual model. These techniques include flowcharts, cognitive maps, inference networks, decision tables, and decision trees.

Knowledge Design

Following the completion of the collection, interpretation, and analysis tasks, some concepts and problem-solving strategies emerge as requiring further investigation and clarification. This task identifies this information and designs an agenda that includes clarifying old issues and discussing new ones with the expert during the following iteration of the acquisition cycle.

While theoretically the cycle could continue indefinitely, in practice, the process is repeated until the resulting system meets some acceptable performance measures.

PARTICIPANTS IN KNOWLEDGE ACQUISITION

The main participants in knowledge acquisition are the domain expert, the knowledge engineer, and the end-user. Each participant plays an important role in knowledge acquisition and must possess certain qualifications to contribute effectively to the knowledge acquisition process.

The Expert

The expert is usually the primary source of knowledge for most expert system projects. The expert's main task is to communicate his or her domain expertise to the knowledge engineer for encoding into an expert system. In addition to possessing extensive knowledge and problem-solving skills in a given domain, and expert should have the following qualifications:

1. Ability to communicate the problem-solving knowledge
2. Willingness and eagerness to participate in the project
3. Ability to work well with others
4. Availability for the duration of the project

The Knowledge Engineer

The main responsibility of a knowledge engineer is to acquire, analyze, interpret, design, and encode the knowledge. Knowledge engineers must have the technical skills for interpreting, analyzing, and coding the collected knowledge. Additionally they should have the following qualifications:

1. Good communications and interpersonal skills
2. Good knowledge elicitation and interviewing skills
3. Good project management skills

The End-User

End-users are an important, yet often ignored, additional source of knowledge. They provide a high-level understanding of the problem. They are particularly useful in providing a general perspective and insight early on during the knowledge elicitation process. Some of the qualifications required for end-users to support knowledge acquisition include availability and willingness to participate in the project, and having on open-minded attitude toward change.

METHODS OF KNOWLEDGE ACQUISITION

Knowledge acquisition methods are classified in different ways and appear under different names in different literature. In this article we follow a classification based on the degree of automation in the acquisition process. The classification divides knowledge acquisition methods into three categories: manual methods, combined manual and automated methods, and automated methods (6). This classification is depicted in Fig. 3.

Manual methods are largely based on some kind of interview between an expert and a knowledge engineer. The knowledge engineer elicits knowledge from the expert during interviewing sessions, refines it with the expert, and then represents it in a knowledge base. The two manual methods commonly used are interviews (structured, unstructured, and questionnaire) and case-based methods (protocol analysis, observation, and case studies). In some cases, an expert may play the role of a knowledge engineer and self elicits the knowledge without the help of a knowledge engineer.

Combined manual and automated methods use techniques and tools to support both experts and knowledge engineers in the knowledge acquisition process. Methods intended to support experts provide an environment for constructing the knowledge base with little or no support from a knowledge engineer. Methods intended to support knowledge engineers provide an environment of acquiring and representing knowledge with minimal support from the experts.

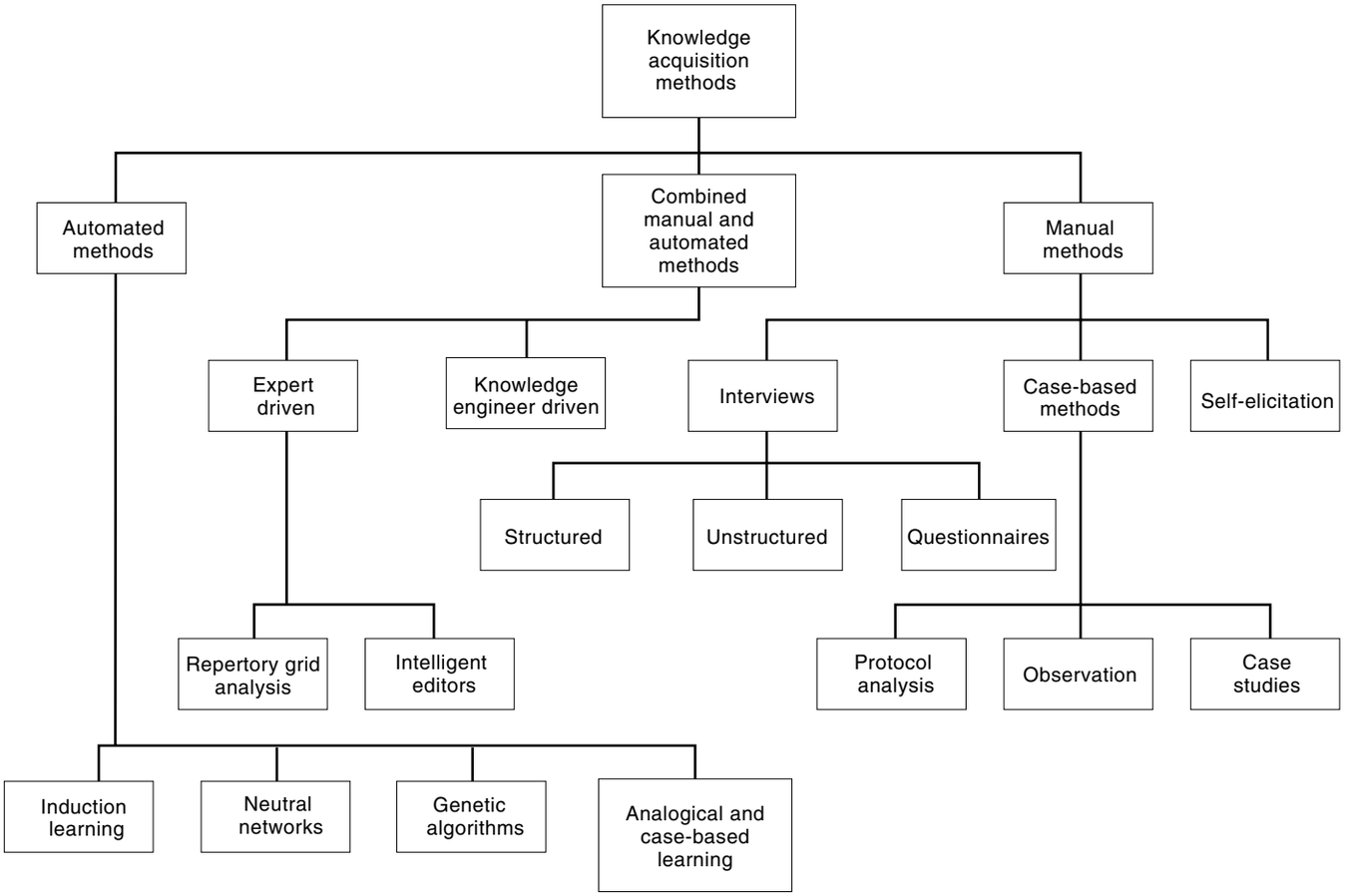

Figure 3. Knowledge acquisition methods. This classification is based on the degree of automation in the acquisition process and is useful in identifying appropriate knowledge acquisition methods for a given project.

Automated methods minimize or even eliminate the roles of both experts and knowledge engineers. They are based on machine learning methods and include learning by induction, neural networks, genetic algorithms, and analogical and case-based reasoning.

It is important to note that the categories of the above classification are not mutually exclusive as some overlap can exist between them.

MANUAL METHODS

Interviews

Interviews are the most common elicitation method used for knowledge acquisition. It involves a two-way dialog between the expert and the knowledge engineer. Information is collected by various means, and subsequently transcribed, interpreted, analyzed, and coded. Two types of interviews are used: unstructured and structured. Although many techniques have been proposed for conducting interviews, effective interviewing is still largely an art.

Unstructured Interviews. Unstructured interviews are conducted without prior planning or organization. They are an informal technique that helps the knowledge engineer gain a general understanding of the problem, its most important attributes, and general problem-solving methods. During unstructured interviews, the knowledge engineer asks some opening questions and lets the expert talk about the problem, its major objects, concepts, and problem-solving strategies. The role of the knowledge engineer is limited to asking clarifying questions or redirecting the interview toward more interesting areas.

Unstructured interviews appear in several variations (6). In the talk-through interview the expert talks through the steps he follows to solve a specific problem. In the teach-through interview, the expert plays the role of an instructor and explains what he does and why he does it in order to solve a problem. In the read-through interview the expert instructs the knowledge engineer on how to read and interpret the documents used for the task.

Unstructured interviews have several advantages. They are useful in uncovering the basic structure of the domain, the main attributes of the problem, and the general problem-solving methods used by the expert. They are appropriate during the early stages of knowledge acquisition when the knowledge engineer is exploring the domain.

However, unstructured interviews suffer from a number of drawbacks (7). First, unstructured interviews lack the organization for the effective transfer of knowledge. Second, due to

lack of structure, domain experts find it difficult to express important elements of their knowledge. Third, experts interpret the lack of structure as requiring little or no preparation. Fourth, data collected from an unstructured interview is often unrelated. Fifth, very few knowledge engineers can conduct an effective unstructured interview. Finally, unstructured situations do not facilitate the acquisition of specific information from experts.

Structured Interviews. Structured interviews maintain a focus on one aspect of the problem at a time by eliciting details on that aspect before moving to a different one. This focus is maintained by structuring the interview based on a prior identification of the problem's key issues obtained through earlier unstructured interviews or other sources. The interview structure forces an organized exchange between the expert and the knowledge engineer and reduces the interpretation problems and the distortion caused by the subjectivity of the expert.

Structured interviews require extensive preparation from the part of the knowledge engineer. In addition, conducting and managing the interview properly require attention to several issues. Some of the basic issues relate to items such as setting up the interview, scheduling the session, choosing the interview location, and the conduct of the first interview. Other issues include knowing how to begin and end the interview and how to ask questions in a way that will provide the desired information. Many guidelines exist in the literature on how to conduct effective structured interviews. For example see the guidelines suggested by McGraw and Harbison-Briggs (7), Prerau (8), and Scott et al. (9).

The main advantage of structured interviews is their focus and the resulting detailed information obtained on a given issue. They are usually easier to manage, and the information collected is easier to analyze and interpret. Structured interviews are particularly useful in identifying the structure of the domain objects and their properties, concept relationships, and general-problem solving strategies.

The main limitation of structured interviews is that concepts unrelated to the interview focus may not be discovered. This limitation is manifest particularly when the knowledge engineer is not fully aware of the topics' main issues. Additionally, structured interviews provide little insight on procedural knowledge.

Questionnaires. Although questionnaires are not strictly an interviewing method, they complement interviews by asking the expert to clarify already developed topics during advanced stages of knowledge acquisition.

Task-Based Methods

Task-based methods refer to a set of techniques that present the expert with a task and attempt to follow his or her reasoning in solving the problem. Task-based methods can help the knowledge engineer in identifying what information is being used, why it is being used, and how it is being used. The methods that can be grouped under this approach include protocol analysis, observation, and case studies.

Protocol Analysis. In protocol analysis, the expert is asked to perform a real task and to verbalize at the same time his or her thought processes while performing the task. Usually a recording is made, using a tape or video recorder, which later becomes a record or protocol that traces the behavior of the expert while solving a problem. As with interviews, this recording is transcribed, analyzed, reviewed, and coded by the knowledge engineer.

The main difference between a protocol analysis and an interview is that a protocol analysis is mainly a one-way communication. The knowledge engineer's task is limited to selecting a task, preparing the scenario, and presenting it to the expert. During the session, the expert does most of the talking as the knowledge engineer listens and records the process.

The main advantage of protocol analysis is that it provides immediate insight of problem-solving methods, rather than retrospectively after the fact. It is particularly useful for nonprocedural types of knowledge, where the expert applies a great deal of mental and intellectual effort to solve a problem.

However, several cognitive psychologists have argued that asking experts to verbalize their problem-solving knowledge while performing a task creates an unnatural situation that influences task performance (10). In addition, some problems, such as ones that involve perceptual-motor tasks, do not have a natural verbalization. Forcing an expert to think aloud in these situations can lead to the collection of misleading and inaccurate information.

Observation. Another useful knowledge acquisition technique is observing the expert in the field while solving a problem. Observation is usually conducted at the place where the expert makes the actual decisions. Experience has shown that the realism of the expert problem-solving approach is greatly influenced by the usual physical environment of the problem.

The main advantage of this approach is that it allows the knowledge engineer to observe the decision making of the expert in a realistic environment. It provides an unbiased and unobtrusive technique for collecting knowledge. It is particularly useful for collecting information on procedural knowledge.

The main disadvantage is that observations are usually expensive and time consuming. A large amount of information is usually collected from which only a small fraction is useful.

Case Studies. A case is an actual problem that has been solved in the past together with its solution and the steps taken to solve it. There are two primary ways a case study is used for knowledge elicitation: retrospective and observational case studies (11). In a *retrospective* case study, the expert is asked to review a case and explain in retrospect how it was solved. The expert begins by reviewing the given recommendation and then works backward to identify the problem concepts and knowledge components used to support this recommendation. In an *observational* case study, the expert is asked to solve the problem while the knowledge engineer observes the problem-solving approach of the expert.

Several types of cases could be used in conjunction with either the retrospective or observational case studies. The two common types used by knowledge engineers are the typical case and the unusual case. The typical case represents a situation that is well understood and known by the expert. The results of a typical case usually reveal the typical knowledge used by the expert to solve a problem. The unusual case represents an unusual or novel situation that requires a deeper-

level of problem-solving knowledge. Usually typical cases are used initially in the project when a general understanding of the domain and the problem-solving expertise is required. Unusual cases are used later in the project when deeper knowledge is needed to provide greater problem-solving expertise to the system.

A main advantage of case studies is that information is obtained in the context of a realistic situation, thus providing more accurate insight into problem-solving strategies. Case studies usually reveal more specific problem-solving knowledge than that obtained from interviewing techniques. Retrospective case studies have the further advantage of not interfering with the problem-solving activity, since retrospection requires the expert to recall from memory the information needed to solve the problem, rather than actually solving the problem.

A major disadvantage of the case method, particularly the retrospective case study, is that it may provide incomplete information and few details on the domain under study. Another disadvantage is the expert's bias toward typical situations solved in the past which could produce inconsistent results. Selecting an unusual but solvable case study could be challenging and presents yet another limitation for this approach.

Self-Elicitation. In some cases, the expert may have both the technical interest and the needed training to play the role of a knowledge engineer. In this case the expert may acquire and represent the knowledge directly without the intermediary of a knowledge engineer. This process can be accomplished through self-administered questionnaires or through self reporting. Self reporting can take the form of an activity log, knowledge charts, introductory tutorials, or other similar documents that report on the problem-solving activities of the expert.

A main problem with self-elicitation methods is that experts are usually not trained in knowledge engineering methods and techniques. The resulting knowledge tends to have high degree of bias, ambiguity, new and untested problem-solving strategies, as well as vagueness about the nature of associations among events (11). In addition, experts lose interest rapidly in the process, and consequently the quality of the acquired knowledge decreases as the reporting progresses. Self-elicitation methods are useful when experts are inaccessible and in the gathering of preliminary knowledge of the domain.

COMBINED MANUAL AND AUTOMATED METHODS

Manual knowledge acquisition methods are usually time consuming, expensive, and even unreliable. Combined manual and automated methods use techniques and tools designed to reduce or eliminate the problems associated with manual methods. They are designed to support both experts and knowledge engineers in the knowledge acquisition process.

Methods to Support the Experts

Repertory Grid Analysis. Repertory grid analysis (RGA) is one of a number of elicitation techniques that attempt to gain insight into the expert's mental model of the problem domain. It is based on a technique, derived from psychology, called the *classification interview.* When applied to knowledge acquisition, these techniques are usually aided by a computer. Repertory grid analysis is based on Kelly's model of human thinking, which is called *personal construct theory* (12). According to this theory, people classify and categorize knowledge and perceptions about the world, and, based on this classification, they are able to anticipate and act on everyday decisions.

The RGA involves the following steps:

1. *Construction of Conclusion Items.* These items are the options that will be recommended by the expert system. For example, the conclusion items of an investment portfolio advisor might include the following options: 100% investment in savings; a portfolio with 100% stocks (portfolio 1); a portfolio with 60% stocks, 30% bonds, and 10% savings (portfolio 2); or a portfolio with 20% stocks, 40% bonds, and 40% savings (portfolio 3).
2. *Construction of Traits.* These traits are the important attributes that the expert considers in making decisions. For example, using the investment portfolio advisor example, traits might include age, investment amount, and investment style. Traits are identified by picking three conclusion items and identifying the distinguishing characteristics of each from the two others. Each trait is given values on a bipolar scale (i.e., a pair of opposite values). In the investment portfolio advisor example, the identified traits could have the following values: young/old, small/large, and conservative/aggressive.
3. *Rating of Conclusion Items According to Traits.* The expert rates each conclusion item on a scale of one to five. Five is given to an item that satisfies the left hand pole of the trait and one to an item that satisfies the right-hand pole. The answers are recorded in a grid as shown in Table 1.
4. *Rule Generation.* Once the grid is completed, rules are generated that provide decision items given a desired trait importance.

A number of knowledge acquisition tools have been developed based on the RGA method. The best known tool of this group is the *expertise transfer system* (ETS) (13). The ETS is used to build a knowledge system through several iterative steps: (1) experts are interviewed to uncover conclusion items, problem-solving traits, trait structure, trait weights, etc., (2) information acquired from the expert is built into information bases, (3) information bases are analyzed and built into knowledge bases (rules, frames, or networks), (4) knowledge bases are incrementally refined using test case histories, and

Table 1. A Repertory Grid for an Investment Portfolio Advisor

Attribute	Age (young-5, old-1)	Investment Amount (small-1, large-5)	Investment Style (conservative-1, aggressive-5)
Savings	2	1	1
Portfolio 1	4	4	5
Portfolio 2	3	3	3
Portfolio 3	2	2	2

(5) knowledge bases are implemented into expert systems. Other representative tools in this category include KRITON (15), and AQUINAS (16).

Intelligent Editors. An intelligent editor allows the domain expert to capture the knowledge directly without the intermediary of a knowledge engineer. The expert conducts a dialog with the editor using a natural language interface which includes a domain-specific vocabulary. Through the intelligent editor, the expert can manipulate the rules of the expert system without knowing the internal structure of these rules.

The editor assists the expert in building, testing, and refining a knowledge base by retrieving rules related to a specific topic and reviewing and modifying the rules if necessary. The editor also provides an explanation facility. The expert can query the system for conclusions given a set of inputs. If the expert is unhappy with the results, he can have the editor show all the rules used to arrive at that conclusion.

Some editors have the ability to suggest reasonable alternatives and to prompt the expert for clarifications when required. Other editors have the ability to perform syntax and semantic checks on the newly entered knowledge and detect inconsistencies when they occur.

A classic example of intelligent editors is a program called TEIRESIAS that was developed to assist experts in the creation and revision of rules for a specific expert system while working with the EMYCIN shell (1).

Methods to Support the Knowledge Engineer

Several types of tools have been developed to support the knowledge engineer. They include knowledge-base editors, explanation facilities, and semantic checkers.

Knowledge-Base Editors. Knowledge-base editors facilitate the task of capturing the knowledge and entering it into the knowledge base. They provide syntax and semantic checks to minimize errors and ensure validity and consistency. Several types of editors exist. Rule editors simplify the task of defining, modifying, and testing production rules. Graphic editors support the development of structured graphic objects used in developing the knowledge base (17).

Explanation Facilities. Explanation facilities support the knowledge engineer in acquiring and debugging the knowledge base by tracing the steps followed in the reasoning process of the expert to arrive at a conclusion.

Semantic Checkers. Semantic checkers support the construction of and changes to knowledge bases. They ensure that no errors or inconsistencies exist in the knowledge.

AUTOMATED METHODS

Automated methods refer to the autonomous acquisition of knowledge through the use of *machine-learning* approaches. The objective of using machine learning is to reduce the cost and time associated with manual methods, minimize or eliminate the use of experts and knowledge engineers, and improve the quality of acquired knowledge. In this section we discuss five of these approaches. They include inductive learning, neural networks, genetic algorithms, case-based reasoning, and analogical reasoning.

Inductive Learning

Inductive learning is the process of acquiring generalized knowledge from example cases. This type of learning is accomplished through the process of reasoning from a set of facts to conclude general principles or rules.

Rule induction is a special type of inductive learning in which rules are generated by a computer program from example cases. A rule-induction system is given an example set that contains the problem knowledge together with its outcome. The example set can be obtained from the domain expert or from a database that contains historical records. The rule-induction system uses an induction algorithm to create rules that match the results given with the example set. The generated rules can then be used to evaluate new cases where the outcome is not known.

Consider the simple example set of Table 2 which is used in approving or disapproving loans for applicants. Application for a loan includes information about the applicant's income, assets, and age. These are the decision factors used to approve or disapprove a loan. The data in this table show several example cases, each with its final decision. From this simple example case, a rule-induction system may infer the following rules:

1. If income is high, approve the loan
2. If income is low, but assets are high, approve the loan
3. If income is medium, assets are medium, and age is middle or higher, approve the loan

The heart of any induction systems is the induction algorithm used to induce rules from examples. Induction algorithms vary from traditional statistical methods to neural computing models.

A classic and widely used algorithm for inductive learning is ID3 (18). The ID3 algorithm first converts the knowledge matrix into a decision tree. Irrelevant decision factors are eliminated, and relevant factors are organized efficiently.

Rule induction offers many advantages. First, it allows knowledge to be acquired directly from example cases, thus avoiding the problems associated with acquiring knowledge from an expert through a knowledge engineer. Second, induction systems can discover new knowledge from the set of examples that may be unknown to the expert. Third, induction can uncover critical decision factors and eliminate irrelevant ones. In addition, an induction system can uncover contradictory results in the example set and report them to the expert.

Induction systems, however, suffer from several disadvantages. They do not select the decision factors of a problem. An

Table 2. Example Dataset from a Loan Application Database Used for Rule Induction

Name	Annual Income	Assets	Age	Loan Decision
Applicant A	High	None	Young	Yes
Applicant B	Medium	Medium	Middle	Yes
Applicant C	Low	High	Young	Yes
Applicant D	Low	None	Young	No

expert is still needed to select the important factors for making a decision. They can generate rules that are difficult to understand. They are only useful for rule-based, classification problems. They may require a very large set of examples to generate useful rules. In some cases, the examples must be sanitized to remove exception cases. Additionally, the computing power required to perform the induction grows exponentially with the number of decision factors.

Neural Networks

Neural networks are a relatively new approach to building intelligent systems. The neural network approach is based on constructing computers with architectures and processing capabilities that attempt to mimic the architecture and processing of the human brain. A neural network is a large network of simple processing elements (PEs) that process information dynamically in response to external inputs. The processing elements are simplified representation of brain neurons. The basic structure of a neural network consists of three layers: input, intermediate (called the *hidden layer*), and output. Figure 4 depicts a simple three-layer network.

Each processing element receives inputs, processes the inputs, and generates a single output. Each input corresponds to a decision factor. For example, for a loan approval application, the decision factors may be the income level, assets, or age. The output of the network is the solution to the problem. In the loan approval application a solution may be simply a "yes" or "no." A neural network, however, uses numerical values only to represent inputs and outputs.

Each input x_i is assigned a weight w_i that describes the relative strength of the input. Weights serve to increase or decrease the effects of the corresponding x_i input value. A summation function multiplies each input value x_i by its weight w_i and sums them together for a weighted sum y. As Fig. 5 illustrates, for j processing elements, the formula for n input is:

$$y_j = \sum_j w_{ij} x_i$$

Based on the value of the summation function, a processing element may or may not produce an output. For example, if the sum is larger than a threshold value T, the processing element produces an output y. This value may then be input to other nodes for a final response from the network. If the total input is less than T, no output is produced. In more sophisticated models, the output will depend on a more complex activation function.

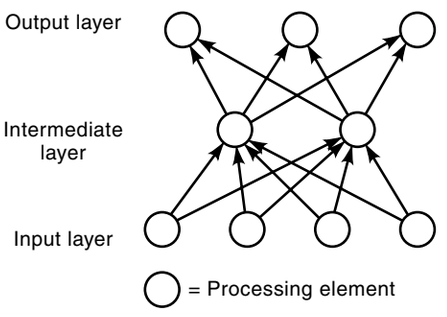

Figure 4. A three-layer neural network architecture. The layers of the network are the input, intermediate (hidden), and output layers.

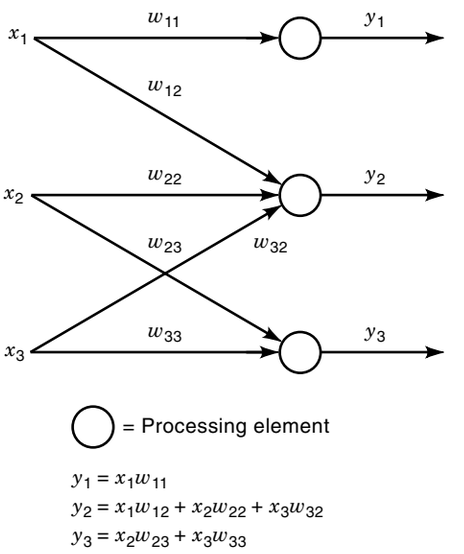

Figure 5. Summation function for a number of neurons. The figure shows how the weight of an input can increase or decrease the effects of that input.

Learning in a Neural Network. The knowledge in a neural network is distributed in the form of internode connections and weighted links. These weights must be learned in some way. The learning process can occur in one of two ways: supervised and unsupervised learning.

In *supervised learning,* the neural network is repeatedly presented with a set of inputs and a desired output response. The weights are then adjusted until the difference between the actual and desired response is zero. In one variation of this approach, the difference between the actual output and the desired output is used to calculate new adjusted weights. In another variation, the system simply acknowledges for each input set whether or not the output is correct. The network adjusts weights in an attempt to achieve correct results. One of the simpler supervised learning algorithms uses the following formula to adjust the weights w_i:

$$w_{i(\text{new})} = w_{i(\text{old})} + \alpha * d * \frac{x_i}{|x_i|^2}$$

where α is a parameter that determines the rate of learning, and d is the difference between actual and desired outputs.

In *unsupervised learning,* the training set consists of input stimuli only. No desired output response is available to guide the system. The system must find the weights w_{ij} without the knowledge of a desired output response.

Neural networks can automatically acquire knowledge from historical data. In that respect they are similar to rule-induction. They do not need, however, an initial set of decision factors or complete and unambiguous sets of data. Neural networks are particularly useful in identifying patterns and relationships that may be subsequently developed into rules for expert systems. Neural networks could also be used to supplement rules derived by other techniques.

Genetic Algorithms

Genetic algorithms refer to a variety of problem-solving techniques that are based on models of natural adaptation and evolution. They are designed the way populations adapt to and evolve in their environments. Members that adapt well are selected for mating and reproduction. The descendants of these members inherit genetic traits from both their parents. Members of this second generation that also adapt well are selected for mating and reproduction and the evolutionary cycle continues. After several generations, members of the resultant population will have adapted optimally or at least very well to the environment.

Genetic algorithms start with fixed population of data structures that are candidate solutions to specific domain tasks. After requiring these structures to execute the specified tasks several times, the structures are rated for their effectiveness at domain solution. On the basis of these evaluations, a new generation of data structures is created using specific *genetic operators* such as reproduction, crossover, inversion, and mutation. Poor performing structures are discarded. This process is repeated until the resultant population consists only of the highest performing structures.

Many genetic algorithms use eight-bit strings of binary digits to represent solutions. Genetic algorithms use four primary operations on these strings:

1. *Reproduction* is an operation that produces new generations of improved solutions by selecting parents with higher performance rating.
2. *Crossover* is an operation that randomly selects a bit position in the eight-bit string and concatenates the head of one parent with the tail of the second parent to produce a child. Consider two parents designated *xxxxxxxx* and *yyyyyyyy*, respectively. Suppose the second bit position has been selected as the crossover point (i.e., *xx* : *xxxxxx* and *yy* : *yyyyyy*). After the crossover operation is performed, two children are generated, namely *xxyyyyyy* and *yyxxxxxx*.
3. *Inversion* is a unary operation that is applied to a single string. It selects a bit position at random, and then concatenates the tail of the string to the head of the same string. For example, if the second position was selected for the following string ($x_1x_2 : x_3x_4x_5x_6x_7x_8$), the inverted string would be $x_3x_4x_5x_6x_7x_8x_1x_2$.
4. *Mutation* operation ensures that the selection process does not get caught in a local minimum. It selects any bit position in a string at random and changes it.

The power of genetic algorithms lies in that they provide a set of efficient, domain-independent search heuristics for a wide range of applications. With experience, the ability of a genetic algorithm to learn increases, enabling it to accumulate good solutions and reject inferior ones.

Analogical Reasoning and Case-Based Reasoning

Analogical reasoning is the process of adapting solutions used to solve previous problems in solving new problems. It is a very common human reasoning process in which new concepts are learned through previous experience with similar concepts. A past experience is used as a framework for solving the new analogous experience. Analogical learning consists of the following five steps:

1. Recognizing that a new problem or situation is similar to a previously encountered problem or situation
2. Retrieving cases that solved problems similar to the current problem using the similarity of the new problem to the previous problem as an index for searching the case database
3. Adapting solutions to retrieved cases to conform with the current problem
4. Testing the new solutions
5. Assigning indexes to the new problem and storing it with its solution

Unlike induction learning, which requires a large number of examples to train the system, analogical learning can be accomplished using a single example or case that closely matches the new problem at hand.

KNOWLEDGE ANALYSIS

After knowledge is collected, it must be interpreted and analyzed. First a transcript of the knowledge acquisition session is produced. This transcript is then reviewed and analyzed to identify key pieces of knowledge and their relationships. A variety of graphical techniques are used to provide a perspective of the collected knowledge and its organization (14).

Knowledge Transcription

Following the knowledge collection phase, an exact and complete transcript of the knowledge acquisition session is usually made. This transcript is used as a basis for interpreting and analyzing the collected knowledge. Transcription can also be partial. In case of a partial transcription, notes taken during knowledge acquisition session can be used to guide the selection of what should be transcribed.

Each transcript is indexed appropriately with such information as the project title, session date and time, session location, attendees, and the topic of the session. A paragraph index number is assigned to cross-reference the source of knowledge extracted from the transcript with the knowledge documentation. This cross-referencing facilitates the effort of locating the source of knowledge if additional information is needed.

Knowledge Interpretation

Knowledge interpretation begins by reviewing the transcript and identifying the key pieces of knowledge or "chunks" (19). Usually declarative knowledge is easy to identify. Procedural knowledge is harder to recognize, as it can be scattered across the transcript, making it harder to relate. In addition to identifying key pieces of knowledge, an important goal of reviewing the transcript is to identify any issues that need further clarification by the expert.

Several techniques can be used in knowledge interpretation: (1) using handwritten notes taken during the knowledge acquisition session in knowledge identification, (2) highlighting of key information in the transcript using word pro-

cessing software features or a pen, and (3) labeling each piece of knowledge with the type of knowledge it represents.

Knowledge Analysis and Organization

After identifying the different types of knowledge, they need to be analyzed and classified. This effort includes the following steps:

1. Recording each identified piece of knowledge with other related pieces in the *knowledge dictionary*. A knowledge dictionary is a repository that maintains, in alphabetical order, a description of each type of knowledge, for example, objects, rules, problem-solving strategies, and heuristics.
2. Organizing, classifying, and relating the pieces of knowledge collected with similar knowledge stored in the knowledge dictionary. This is a complex iterative step that requires the involvement of the expert to confirm and help refine the structure of knowledge developed.
3. Reviewing the collected knowledge to identify those areas that need further clarification.

Graphical techniques that show how the different pieces of knowledge are related are particularly useful. The next section overviews some of the knowledge representation methods that support both the knowledge engineer and expert in analyzing knowledge.

KNOWLEDGE REPRESENTATION

Knowledge acquired from experts and other sources must be organized in such a way that it can be implemented and accessed whenever needed to provide problem-solving expertise. Knowledge representation methods can be classified into two broad types: (1) those that support the analysis of the acquired knowledge and the development of a conceptual model of the expert system, and (2) those that support the implementation formalism of the development environment.

The first type of representation, called *intermediate representation,* allows knowledge engineers to focus on organizing, analyzing, and understanding the acquired knowledge without concerning themselves with the representation formalisms of the implementation environment. The intermediate representation is continually refined and updated through additional knowledge acquisition until the knowledge engineers are satisfied that they have a sufficiently complete model to guide the implementation design. Intermediate representation methods are usually pictorial and include flowcharts, graphs, semantic networks, scripts, fact tables, decision tables, and decision trees.

The second type of representation, called the *implementation representation,* is used to create an implementation design for the chosen development environment. The conceptual model is mapped directly into the representation model of the development environment without the need to understand the function that the knowledge should serve. Implementation representation often uses frames or production rules.

Each representation method emphasizes certain aspects of the knowledge represented. The choice of a representation method will depend on how well the representation schemes support the structure of the problem. In this article, we consider six of the most common knowledge representation techniques:

Logic
Production rules
Frames
Semantic networks
Objects–attribute–value triplets
Scripts
Decision tables
Decision trees

Logic

Logic is the oldest form of knowledge representation. It uses symbols to represent knowledge. Operators are applied to these symbols to produce logical reasoning. Logic is a formal well-grounded approach to knowledge representation and inferencing. There are several types of logic representation techniques. The two approaches used in artificial intelligence and expert system development are propositional logic and predicate calculus.

Propositional Logic. A proposition is a statement that is either true of false. Symbols, such as letters, are used to represent different propositions. For example, consider propositions A and B used to derive conclusion C:

A = Employees work only on weekdays
B = Today is Saturday
C = Employees are not working today

Propositional logic provides logical operators such as AND, OR, NOT, IMPLIES, and EQUIVALENCE that allows reasoning using various rule structures. Table 3 lists the propositional logic operators and their common symbols.

The AND operator combines two propositions and returns true if both propositions are true. The OR operator combines two propositions and returns true if either or both propositions are true. The NOT operator is a unary operator that returns false if proposition A is true, otherwise it returns true if proposition A is false. The EQUIVALENCE operator returns true when both propositions have the same truth assignment. The IMPLIES operator indicates that if proposition A is true, then proposition B is also true.

Table 3. Logical Operators and their Symbols

Operator	Symbol
AND	$\wedge$, &, $\cap$
OR	$\vee$, $\cup$, +
NOT	$\neg$, $\sim$
IMPLIES	$\supset$, $\rightarrow$
EQUIVALENCE	$\equiv$

Table 4. Truth Table for IMPLIES Operator

A	B	A → B
T	T	T
T	F	F
F	T	T
F	F	T

A truth table is used to show all possible combinations of an operator. Table 4 shows the truth table for the IMPLIES operator.

Since propositional logic deals only with the truth of complete statements, its ability to represent real-world knowledge is limited.

Predicate Calculus. Predicate calculus is an extension of propositional logic that provides finer presentation of knowledge. It permits breaking down a statement into the objects about which something is being asserted and the assertion itself. For example in the statement *color*(*sky,blue*), the objects sky and blue are associated through a color relationship.

Predicate calculus allows the use of variables and functions of variables in a statement. It also uses the same operators used in propositional logic in addition to two other symbols, the universal quantifier ∀ and the existential quantifier ∃, that can be used to define the range or scope of variables in an expression. Inferencing capability in predicate calculus is accomplished through the use of these operators.

Since predicate calculus permits breaking statements down into component parts, it allows for a more powerful representation model that is more applicable to practical problems.

Production Rules

Production rules are a popular knowledge representation scheme used for the development of expert systems. Knowledge in production rules is presented as condition-action pairs: IF a *condition* (also called antecedent or premise) is satisfied, THEN an action (or consequence or conclusion) occurs. For example:

IF the sky is clear
THEN it is not going to rain

A rule can have multiple conditions joined with AND operators, OR operators, or a combination of both. The conclusion can contain a single statement or several statements joined with an AND. A certainty factor, usually a value between −1 and 1, can also be associated with a rule to capture the confidence of the expert with the results of the rule (20).

Production rules represent the system's knowledge base. Each rule represents an independent portion of knowledge that can be developed and modified independently of other rules. An inference mechanism uses these rules along with information contained in the working memory to make recommendations. When the IF portion of a rule is satisfied, the rule fires and the statements in the THEN part of the rule are added to the working memory. These statements can trigger other rules to fire. This process continues until the system reaches a conclusion.

Production rules offer many advantages. They have simple syntax, are easy to understand, are highly modular, and their results are easily inferred and explained. Production rules, however, are not suitable for representing many types of knowledge, particularly descriptive knowledge. They could also be difficult to search, control, and maintain for large complex systems.

Semantic Networks

Semantic networks are graphic depictions of a domain's important objects and their relationships. It consists of nodes and arcs that connect the nodes. The nodes represent the objects and their properties. Objects can represent tangible or intangible items such as concepts or events. The arcs represent the relationships between the objects. Some of the most common arc types are the IS-A and HAS-A type. The IS-A relationship type is used to show class membership, that is, an object belongs to a larger class of objects. The HAS-A relationship type indicates the characteristics of an object.

Figure 6 shows a simple example of a semantic network. In this example, the Pyramid node is connected to a property node, indicating that "a pyramid has faces." It is also connected to the Structure node via an IS-A link, indicating that "a pyramid is a structure." The Structure node is connected to a Material node via a MADE OF link, and the Stone, Wood, and Steel nodes are connected to the Material node via an IS-A link.

A very useful characteristic of semantic networks is the concept of *inheritance.* Inheritance is the mechanism by

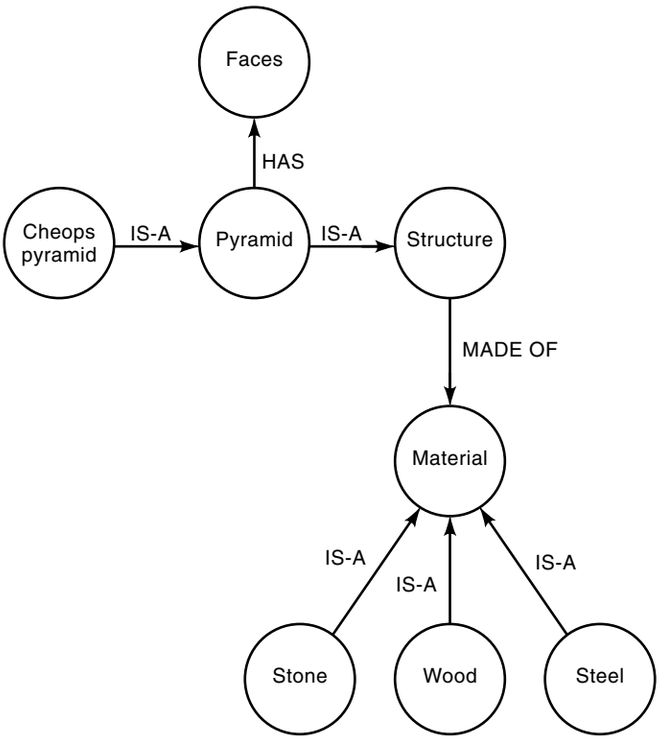

Figure 6. Example of a simple semantic network. Nodes represent objects and links represent relationship between the objects. An object connected to other objects through an IS-A relationship inherits the characteristics of these objects.

```
("Cheops" Pyramid
(A-KIND-OF-(VALUE pyramid))
(MATERIAL(VALUE limestone granite))
(BASE-LENGTH(VALUE 233m))
(HEIGHT(VALUE 146m))
(NO.-OF-FACES(DEFAULT fget))
(ANGLE(VALUE if-needed))
(BASE-AREA(VALUE if needed))
(VOLUME(VALUE if-needed))
(NO.-OF-SATTELITES(DEFAULT fget))
```

Figure 7. Example of a frame for the Cheops pyramid. This frame illustrates different types of slots that contain attribute value pairs, default values, conditions for filling a slot, pointers to other related frames, functions, or procedures that are activated under different conditions.

which nodes connected to other nodes through an IS-A relationship inherit the characteristics of these nodes. A main advantage of inheritance is that it simplifies adding new knowledge to the network. When a new node is added, it inherits a wealth of information throughout the network via the IS-A links. Similarly, when a general node is added (e.g., the Structure node), other nodes inherit its properties.

Semantic networks have many advantages as a knowledge representation scheme. They are easy to understand and provide flexibility and economy of effort in adding new objects and relationships. They provide a storage and processing mechanism similar to that of humans, and the inheritance mechanism provides an efficient way of inferencing. Semantic networks have also several limitations. Exceptions offer potential difficulty to the mechanism of inheritance, and, because semantic networks do not represent sequence and time, procedural knowledge is difficult to represent.

Frames

A frame is a data structure that includes both declarative and procedural knowledge about a particular object. In that respect, frames are similar to objects used in object-oriented programming. A frame consists of a collection of *slots* that may be of any size and type. Slots have a name and any number of subslots, which are called *facets.* Each facet has a name and any number of values. Figure 7 depicts a simple frame for the Cheops pyramid.

Facets contain information such as attribute value pairs, default values, conditions for filling a slot, pointers to other related frames, functions, and procedures that are activated under different conditions. The conditions that can activate a procedure are specified in the IF-CHANGED and IF-NEEDED facets. An IF-CHANGED facet contains a procedural attachment, called a *demon.* This procedure is invoked when a value of a slot is changed. An IF-NEEDED facet is used when no slot value is given. It specifies a procedure that is invoked to compute a value for the slot.

For example, the Cheops pyramid frame of Fig. 7 has attribute value slots (A-KIND-OF, MATERIAL, BASE-LENGTH, HEIGHT), slots which take default values (NO.-OF-FACES and NO.-OF-SATTELITES), and slots with attached IF-NEEDED procedures (ANGLE, BASE-AREA, VOLUME). The value "fget" in the default values slots is a function call that retrieves a default value from another frame such as the general pyramid frame for which Cheops is a KIND-OF. When activated, the fget function recursively looks for default values for the slot from ancestor frames until one is found.

Frames are usually connected together to form a hierarchical structure. This hierarchical arrangement of frames allows inheritance. Each frame inherits the characteristics and behavior of all related frames at higher levels of the hierarchy. For example the Cheops pyramid frame is linked to a general pyramid frame that contains information common to all pyramids. In this case the Cheops pyramid frame inherits all the descriptive and procedural information of the pyramid frame.

Inferencing in frames is based on the premise that previous experiences with objects and events create certain expectations about newly encountered objects and events. First, knowledge about an object or situation is stored in long-term memory as a frame. Then, when a similar object or situation is encountered, an appropriate frame is retrieved from memory and used for reasoning about the new situation.

Frames have many advantages. They are a powerful mechanism for representing knowledge, since both declarative and procedural information are captured. In addition, slots for new attributes and procedures are easy to set up. Frames, however, have a rather complicated reasoning. As a result, the implementation of their inferencing mechanism is difficult.

Objects-Attribute-Value Triplets

An object, attribute, and value triplet, also known as the *O-A-V triplet* is another way of representing knowledge. *Objects* can represent physical or abstract items. *Attributes* are properties of the objects, and *values* are specific values that an attribute has at a given time. An attribute can have single or multiple values. These values can be static or dynamic. Figure 8 illustrates a simple O-A-V triplet.

O-A-V triplets can be considered as a variation of either the semantic networks or frames. They are useful in depicting relationship between objects, such as inheritance, part-of, and causal relationships.

Scripts

Scripts are framelike structures used to represent stereotypical situations such as eating in a restaurant, shopping in a supermarket, or visiting a doctor. Similar to a script for a play, the script structure is described in terms of roles, entry conditions, props, tracks, and scenes. *Roles* refer to the people

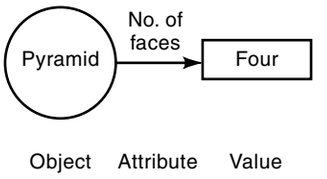

Figure 8. Example of a simple objects–attributes–values triplet. An object–attribute–value triplet is useful in depicting relationships between objects.

involved in the script. *Entry conditions* describe the conditions that must be satisfied before the events described in the script can occur. *Props* are the items used in the events of the script. *Track* refers to variations that might occur in a particular script. Finally, *scenes* are the sequence of events that take place for the script situation. Figure 9 depicts a typical script. It is adapted from the well-known restaurant example used to show how knowledge is represented in scripts.

Similar to frames, reasoning with scripts begins with the creation of a partially filled script that describes the current situation. A known script with similar properties is retrieved from memory using the script name, preconditions, or any other keywords as index values for the search. The slots of the current situation script are then filled with inherited and default values from the retrieved scripts.

Scripts offer many of the advantages of frames, particularly the expressive power. However, as is similar to frames, they and their inference mechanisms are difficult to implement.

Script name	:	Restaurant
Track	:	Fast-food restaurant
Roles	:	Customer Server
Props	:	Counter Tray Food Money Napkins Salt/Pepper/Catsup/Straws
Entry Conditions	:	Customer is hungry Customer has money
Scene 1	:	Customer parks car Customer enters restaurant Customer waits in line at counter Customer reads the menu on the wall and makes a decision about what to order
Scene 2	:	Customer gives order to server Server fills order by putting food on tray Customer pays server
Scene 3	:	Customer gets napkins, straws, salt, etc. Customer takes tray to an unoccupied table Customer eats food quickly
Scene 4	:	Customer cleans up table Customer discards trash Customer leaves restaurant Customer drives away
Results	:	Customer is no longer hungry Customer has less money

Figure 9. Example of a restaurant script. Scripts are used to represent stereotypical situations. A script's structure is described in terms of tracks, roles, props, entry conditions, scenes, and results.

Decision Tables

A decision table is a two-dimensional table that enumerates all possible combinations of attribute values and the conclusions that can be made for each combination of these values.

Attributes								
Age[1]	Y	Y	Y	Y	O	O	O	O
Investment Amount[2]	S	S	L	L	S	S	L	L
Investment Style[3]	C	A	C	A	C	A	C	A
Conclusions								
Savings	X	X			X	X		
Portfolio 1				X				
Portfolio 2			X					X
Portfolio 3							X	

[1]Y = young; O = old.
[2]S = small; L = large.
[3]C = conservative; A = aggressive.

Figure 10. Example of a decision table for an investment portfolio advisor. The upper half of the table enumerates all possible combinations of attribute values. The lower half shows the conclusions that can be made for each combination of these values.

An example of a decision table is shown in Fig. 10. This example gives an expert's recommendations for investment decisions based on age, amount of investment, and investment style.

Decision tables are suitable for a small number of decision attributes, each with a small number of possible values. If the number of attributes or possible values is large, the decision table becomes quite complex. Decision tables are suitable as an intermediate representation for documenting and analyzing knowledge. It is not possible to make inferences directly from the tables, except through rule induction.

Decision Trees

Decision trees are a graphic representation of a problem domain search space. A decision tree is composed of nodes and branches. Initial and intermediate nodes represent decision attributes, and leaf nodes represent conclusions. A path from the root node to a leaf node corresponds to a decision path that might be encountered in the problem domain. Figure 11 shows the decision tree version of the problem presented as a decision table in Fig. 10.

Decision trees are useful not only to show the problem-solving steps, but also the order in which input data is requested and the reasoning steps the expert system should take to reach a conclusion. Decision trees are more natural for experts to understand and use than formal methods such as rules of frames. They are particularly useful to represent the knowledge of identification systems (diagnostics, troubleshooting, classification, etc.).

VALIDATION AND VERIFICATION OF KNOWLEDGE

An important activity of knowledge acquisition is the testing and evaluation of the quality and correctness of the acquired knowledge and its implementation. This activity can be separated into two components: validation and verification (21). *Validation* refers to determining whether the "right" system was built, that is, whether the system does what it was meant to do at an acceptable level of accuracy. Validating the knowledge involves confirming that the acquired knowledge is suf-

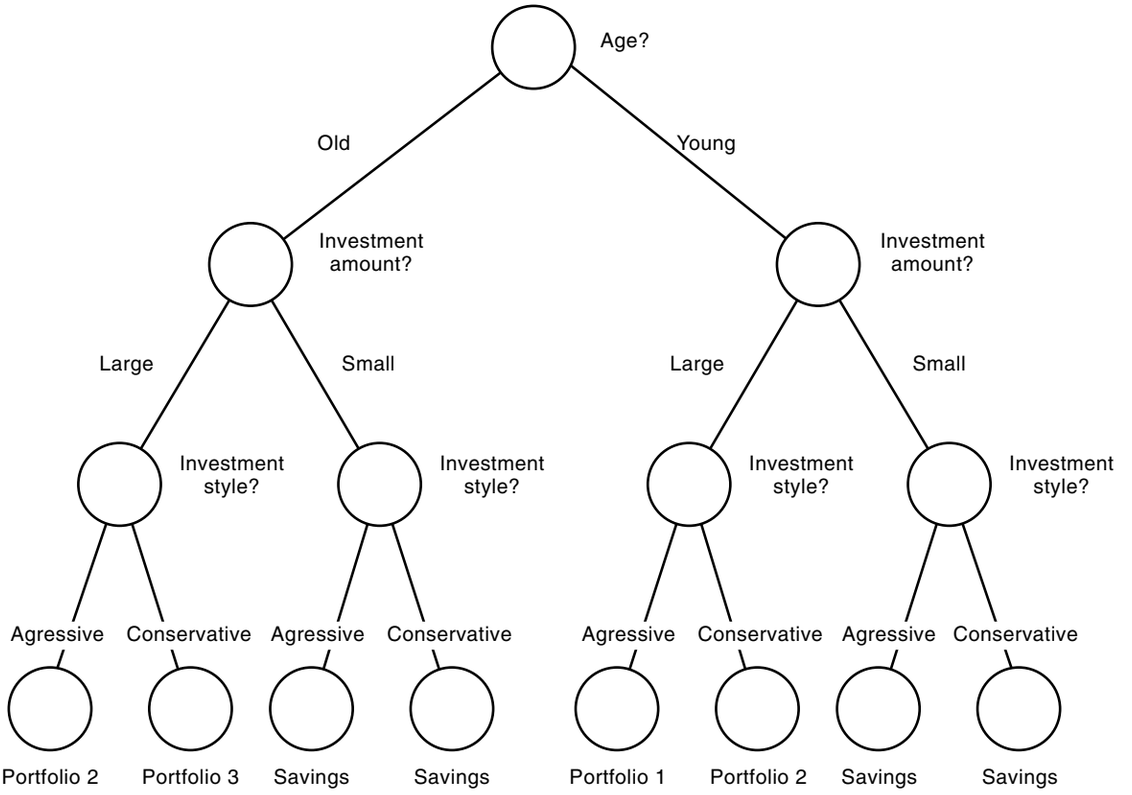

Figure 11. Example of a decision tree for the investment portfolio advisor of Figure 10. Initial and intermediate nodes represent decision attributes, and leaf nodes represent conclusions.

ficient to perform the task at a sufficient level of expertise. *Verification* refers to determining whether the system was built "right," that is, whether the system correctly implements its specifications. Verifying a system means that the program accurately implements the acquired knowledge as acquired and documented.

Validation and verification of knowledge are highly interrelated. Errors in the knowledge implementation are often discovered during validation when the acquired knowledge is checked to see if it performs the desired task at a sufficient level of expertise.

Validation and Verification as Part of Knowledge Acquisition

Expert systems are developed iteratively; therefore, they inherently include repeated validation and verification testing as part of their development. Each time a version of the expert system program is run to test the knowledge, the correctness of the program is checked as well. Thus, in addition to finding deficiencies in the acquired knowledge, the knowledge acquisition cycle detects and corrects programming errors. Validation and verification during knowledge acquisition can occur before implementation has begun using manual simulation or after initial implementation by testing the evolving prototype.

Validation Using Manual Simulation

Early in developing an expert system and before implementation has begun, knowledge acquisition follows a basic development cycle: (1) eliciting knowledge, (2) interpreting, analyzing, and organizing acquired knowledge, and (3) testing knowledge. Using this approach, the expert analyzes a test case and manually uses hand simulation of the acquired knowledge. The results of the expert's analysis are compared with those of the hand simulation. If the results differ, the appropriate area of knowledge is revised and corrected. This process is repeated until no discrepancies occur between the expert's analysis and the results of the simulation of the acquired knowledge.

Validation Using an Evolving Prototype

When enough knowledge is acquired to allow a prototype implementation, the knowledge acquisition process follows a modified cycle consisting of the following steps: (1) eliciting knowledge, (2) interpreting, analyzing, and organizing acquired knowledge, (3) implementing knowledge, and (4) testing knowledge. During the testing phase, a test case is presented to the expert and run using the evolving prototype. The results of the expert's analysis are compared against the results of the prototype. If the results differ, the portion of the knowledge that produced the discrepancy is identified and is manually simulated to see if it agrees with the expert's analysis. If manual simulation produces results that agree with the expert, then an implementation error is likely the source of the discrepancy. If manual simulation does not agree with the expert's analysis, acquired knowledge is revised, modified, or expanded until it comes into agreement

with the expert analysis. This process is repeated throughout the knowledge acquisition phase.

Validation testing during expert system development could be conducted by the domain expert or by a group of consulting experts. Using multiple experts has the advantage of removing potential biases of single experts and generally reveals and corrects more errors in the expert system's knowledge and implementation. It also provides the nontechnical benefit of adding credibility to the validation effort.

On the other hand multiple experts might disagree and provide contradicting opinions. In that case, one of several approaches can be used to integrate the expert's opinions (22). These techniques include selecting the majority decision, blending different lines of reasoning through consensus methods, such as Delphi, applying analytical models used in multiple-criteria decision making, selecting a specific line of reasoning based on the situation, and using blackboard systems that maximize the independence among knowledge sources by appropriately dividing the problem domain.

Validation of the Developed Expert System

In some domains the correctness of the expert system recommendation can be trivially determined without the need for comparison against human expert's judgment. In other domains, the correctness of the results needs to be confirmed by experts who generally agree on the quality of the system's recommendations.

Validation of the developed system is accomplished by comparing the developed system's operational results against the judgment of the expert. A variation of this approach is to run a number of test cases on the developed system and compare the system's recommendations against the results obtained by the human experts.

If feasible, it is highly recommended to evaluate the performance of the expert system in the field under actual operating conditions. This approach provides the most realistic validation of the system in addition to convincing potential users of the value of the system, if the tests are successful.

As in the case of validating an expert system during development, validating a developed expert system can be accomplished using a single expert or multiple experts. These are usually the same experts that performed validation testing during the expert system development.

Verification of the Expert System Program

Verification ensures that the program accurately implements the acquired knowledge. The knowledge acquisition process by its nature uncovers errors not only in the knowledge but in the implementation as well. Implementation errors are often identified during validation when the knowledge of the system is checked for correctness.

In addition to ensuring that the coded knowledge reflects the documented knowledge accurately, verification requires checking the expert system program for internal errors in the knowledge base and the control logic that provides the inferencing mechanism. For example, a rule-based system should not have redundant, conflicting, inconsistent, superfluous, subsumed, or circular rules. In frame-based systems, there should not be any slot with illegal values, inheritance conflicts that are unresolved, or circular inheritance paths. Most rule- and frame-based systems provide capabilities for checking many of these potential problems. Other testing methods should be employed for potential problems not checked automatically. Software systems with better testing and error detection capability enhance the verification phase of the system. Verifying the control logic that performs inferencing can be minimized if the project is utilizing a standard, commercial off-the-shelf tool.

BIBLIOGRAPHY

1. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat (eds.), *Building Expert Systems,* Reading, MA: Addison-Wesley, 1983.
2. R. E. Nisbett and T. D. Wilson, Telling more than we can know: Verbal reports on mental processes, *Psychol. Rev.,* **84**: 231–259, 1977.
3. N. Dixon, *Preconscious Processing,* Chichester UK: Wiley, 1981.
4. H. M. Collins, *Changing Order: Replication and Induction in Scientific Practice,* London: Sage, 1985.
5. L. Bainbridge, Asking questions and accessing knowledge, *Future Comput. Syst.,* **1**: 143–149, 1986.
6. E. Turban, *Expert Systems and Applied Artificial Intelligence,* New York: Macmillan, 1992.
7. K. L. McGraw and K. Harbison-Briggs, *Knowledge Acquisition: Principals and Guidelines,* Englewood Cliffs, NJ: Prentice-Hall, 1989.
8. D. S. Prerau, *Developing and Managing Expert Systems: Proven Techniques for Business and Industry,* Reading, MA: Addison-Wesley, 1990.
9. A. C. Scott, J. E. Clayton, and E. L. Gibson, *A Practical Guide to Knowledge Acquisition,* Reading, MA: Addison-Wesley, 1991.
10. J. Evans, The knowledge elicitation problem: a psychological perspective, *Behav. Inf. Technol.,* **7** (2): 111–130, 1988.
11. J. Durkin, *Expert Systems: Design and Development,* New York: Macmillan, 1994.
12. D. D. Wolfgram, *Expert Systems,* New York: Wiley, 1987.
13. G. A. Kelly, *The Psychology of Personal Constructs,* New York: Norton, 1955.
14. J. H. Boose, *Expertise Transfer for Expert Systems Design,* Amsterdam: Elsevier, 1986.
15. A. J. Diederich, A. I. Ruhmann, and A. M. May, Kriton: A knowledge acquisition tool for expert systems, *Int. J. Man–Mach. Stud.,* **26** (1): 29–40, 1987.
16. J. H. Boose and J. M. Bradshaw, Expertise transfer and complex problems: Using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems, *Int. J. Man-Mach. Stud.,* **26** (1): 3–28, 1987.
17. M. Freiling et al., Starting a knowledge engineering project: A step-by-step approach, *AI Mag.,* 150–164, Fall 1985.
18. P. R. Cohen and E. A. Feigenbaum, *The Handbook of Artificial Intelligence,* vol. 3, Reading, MA: Addison-Wesley, 1982.
19. J. Bell and R. J. Hardiman, The third role—the naturalistic knowledge engineer, in D. Diaper (ed.), *Knowledge Elicitation: Principles,Techniques and Applications,* New York: Wiley, 1989.
20. E. Shortliffe and B. G. Buchanan, A model of inexact reasoning in medicine, *Math. Biosci.,* **23**: 351–375, 1968.
21. R. M. O'Keefe, O. Balci, and E. P. Smith, Validating expert system performance, *IEEE Expert,* **2** (4): 81–90, 1987.
22. S. M. Alexander and G. W. Evans, The integration of multiple experts: a review of methodologies, in E. Turban and P. Watkins (eds.), *Applied Expert System,* Amsterdam: North Holland, 1988.

MAGDI N. KAMEL
Naval Postgraduate School

KNOWLEDGE ACQUISITION. See MACHINE LEARNING.

KNOWLEDGE-BASED SYSTEMS. See EXPERT SYSTEMS.

KNOWLEDGE BASE, FINANCIAL. See BUSINESS GRAPHICS.

KNOWLEDGE DISCOVERY IN DATABASES. See DATABASE MINING.