

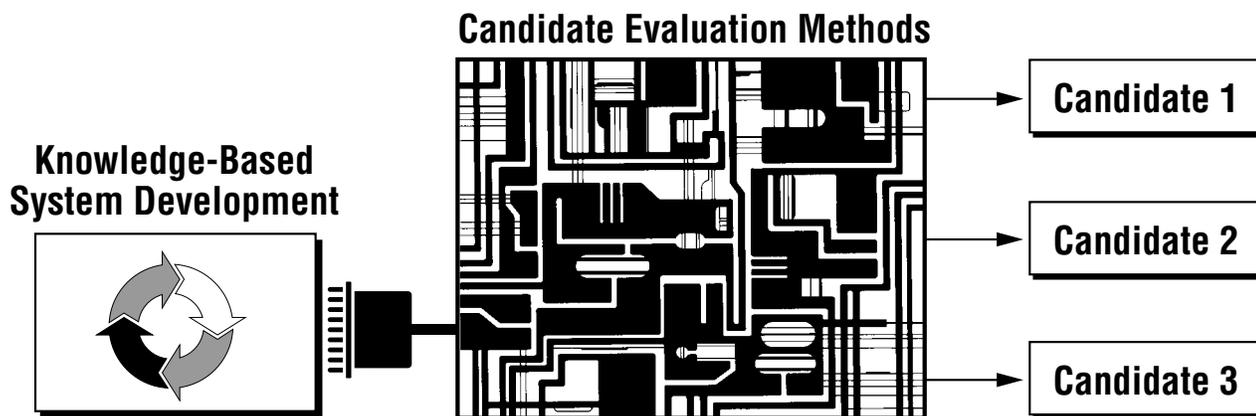
# A Task-Specific Problem-Solving Architecture for Candidate Evaluation

Michel Mitri

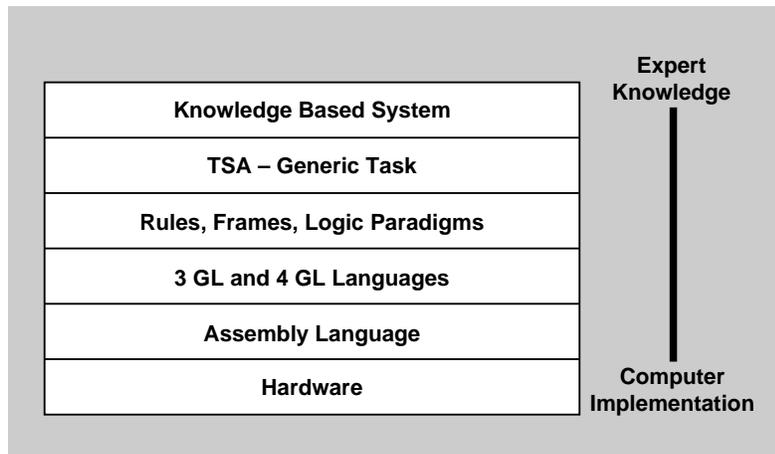
Many of the problems we encounter in our day-to-day lives involve deciding between a set of options, or *candidates*. In these kinds of problems, one needs to determine the worthiness of each candidate to select the best one. Other problems involve assessing an individual candidate's strengths and weaknesses to suggest ways of improving its performance. Both types of problems require the process of evaluation. In schools, for example, students are evaluated for both remedial purposes and selection and ranking.

*Task-specific architectures are a growing area of expert system research. Evaluation is one task that is required in many problem-solving domains. This article describes a task-specific, domain-independent architecture for candidate evaluation. I discuss the task-specific architecture approach to knowledge-based system development. Next, I present a review of candidate evaluation methods that have been used in AI and psychological modeling, focusing on the distinction between discrete truth table approaches and continuous linear models. Finally, I describe a task-specific expert system shell, which includes a development environment (Cevad) and a run-time consultation environment (Ceval). This shell enables nonprogramming domain experts to easily encode and represent evaluation-type knowledge and incorporates the encoded knowledge in performance systems.*

There is a wealth of research literature, in both AI and the social sciences, dealing with evaluation methodology. *Evaluation* is often described in terms of establishing numeric scores or qualitative ratings for a candidate. Candidates are usually represented in terms of attributes (criteria) that are relevant to the evaluation. Many models involve a weighted scoring process, where the weights indicate the importance levels of the attributes being scored. Such models have been used in education (Beggs and Lewis 1975), marketing (Wright 1975),



*The architecture is specifically designed for use by non-programming domain experts...*



*Figure 1. Levels of Abstraction for Expert System Development. The task-specific architecture and generic task paradigms are closer to the knowledge-use level of representation than traditional representational schemes such as rules, frames, and logic.*

software validation (Gaschnig et al. 1983; O’Keefe 1989), and a host of other domains. Indeed, evaluation is a such a ubiquitous task that an entire research discipline has grown that is entirely devoted to the study of evaluation techniques (Edwards and Newman 1982).

This article presents a task-specific architecture for reasoning and knowledge representation in candidate evaluation tasks. The architecture includes primitives for representing candidates, their attributes, importance levels, and numeric-qualitative performance measures. It includes a mechanism for establishing and interpreting evaluation results and recommending actions to take based on the evaluation. The architecture is specifically designed for use by nonprogramming domain experts and, thus, enhances the ability to quickly acquire and represent expert knowledge.

Before presenting the candidate evaluation architecture, I discuss the task-specific architecture paradigm, then compare some of the methods that have been used for evaluation in AI and the social sciences.

### Task-Specific Architectures

The notion of task-specific architectures was motivated by a dissatisfaction with the commonly used knowledge representation paradigms usually associated with expert systems, such as rules, frames, and logic, and their corresponding control regimes of forward and backward chaining, inheritance links, and predicate and clause selection. These

frameworks are general in their applicability but are expressed at too low a level of abstraction to make them useful and coherent in formulating complex problem solutions. Their level of abstraction is too close to the implementation level of software design, whereas a truly useful knowledge engineering tool should be expressed using constructs at the knowledge-use level (Newell 1982; Clancey 1985; Chandrasekaran 1983, 1986; Steels 1990). In other words, the tool

should be forced to think like the expert, not vice versa.

Task-specific architectures attempt to do this expert thinking by expressing their representation constructs in terms that are natural for the type of problem to be solved. Thus, they are less general in scope than the rule-based or object-oriented paradigms but make up for this lack of generality by being more expressive in their ontology and, therefore, making it easier to implement knowledge bases in the task areas to which they apply. Figure 1 illustrates this point.

In the early-to-mid 1980s, Chandrasekaran (1983, 1986) and his colleagues at The Ohio State University began to isolate various types of problem-solving methods. Chandrasekaran’s thesis was the following: “There exist different problem solving types, i.e. uses of knowledge, and corresponding to each is a separate substructure specializing in that type of problem solving” (1983, p. 9).

Chandrasekaran identified two of these problem-solving types in the process of developing a medical diagnostic system called MDX. These types were hierarchical classification and inferential data retrieval. The problem-solving regimes identified by Chandrasekaran were later termed generic tasks. As time went on, several additional generic tasks were added to the list, including abduction (Punch et al. 1990), simple design (Brown and Chandrasekaran 1986), structured matching (Bylander, Goel, and Johnson 1989), and functional reasoning (Sticklen, Chandrasekaran, and Bond 1989).

Generic tasks represent one stream in the

task-specific architecture philosophy. Characteristics of task-specific regimes in general and the generic-task approach in particular include the following: First, control knowledge is usually intertwined with domain knowledge. The knowledge base of an expert system is not divorced from the inference engine (or problem solver), as is the case with many other paradigms (Chandrasekaran 1986). Second, there is often a tendency to distribute knowledge among knowledge agents, or *specialists*, which each contain expertise in a limited area of the problem domain (and the problem-solving task) and which communicate with each other along prescribed, usually hierarchical channels (Sticklen et al. 1987). Third, general intelligence is not the goal, unlike with rules and frames, which are meant to capture any type of intelligent activity. A task-specific architecture is constrained in its applicability, sacrificing generality to achieve explicitness of control representation, richness of ontology, and high-level (abstract) control and domain knowledge representation. However, generic tasks (if not task-specific architectures in general) should be applicable across a wide range of domains and problems encountered by humans. For example, a truly generic diagnostic strategy should not be confined to medical diagnostic problems but should be applicable for diagnostic reasoning in nonmedical domains as well. Fourth, a task-specific architecture method should have an identifiable control regime and a known set of primitives for representing the knowledge that this task embodies. This approach implies that the strategy can be implemented in the form of an expert system shell (that is, template), where instead of rules, objects, procedures, and so on, as the knowledge representation constructs, one would directly encode the knowledge in terms that relate to the task primitives. These task primitives should be expressed at the proper level of granularity or abstraction for the task at hand.

The desired characteristics of task-specific architectures have significant implications for knowledge acquisition, as pointed out by Bylander and Chandrasekaran (1987). They discussed the interaction problem in knowledge representation, stating that “representing knowledge for the purpose of solving some problem is strongly affected by the nature of the problem and the inference strategy to be applied to the knowledge” (p. 232).

The implications are that the knowledge-acquisition process should be guided by the language or vocabulary of the problem-solving task at hand. Thus, an early goal of knowledge

acquisition is to identify the generic task(s) that are appropriate for the problem at hand. Once these tasks have been selected, said Bylander and Chandrasekaran, the interviewing process can be guided by, and expressed in terms of, the language constructs of the chosen generic tasks.

I would go a step further beyond this statement. In my view, expressing the architectural primitives of the task-specific architecture in terms of high-abstraction knowledge use-level constructs makes it possible for nonprogrammers to directly use a task-specific architecture shell without having to go through an intermediary knowledge engineer or computer programmer for encoding knowledge onto the computer. In other words, the programming language of a task-specific architecture (or the generic task) can itself serve as a knowledge-acquisition tool, provided that the language is implemented in a user-friendly environment that does not have the look or feel of a programming language (that is, it involves menus, graphic displays, and other user-friendly features).

This approach is being used at the Center for International Business Education and Research (CIBER) at Michigan State University (MSU). Here, College of Business students and faculty (most of whom have no computer programming experience) are using task-specific architecture shells to directly encode their knowledge pertaining to several areas in international marketing. I discuss one of these shells in this article, but first I discuss two approaches to evaluation in AI.

### **A Historical Perspective on Evaluation in AI: Samuel versus Berliner**

To set the stage for describing the candidate evaluation architecture, it is helpful to compare two alternative approaches to evaluation functions developed for game-playing expert systems. Both approaches significantly influenced the development of the candidate evaluation architecture that I present later. The first approach is Samuel’s (1959; 1967) signature table, used in checker-playing systems, which was later generalized by Bylander, Goel, and Johnson (1989) into the generic task of structured matching. The second approach is called SNAC (smoothness, nonlinearity, and application coefficients). It was developed by Berliner (Berliner 1977, 1979; Berliner and Ackley 1982) and involves a hierarchy of linear and nonlinear mathematical operators for scoring potential board moves

*...a major difference... is one of discrete (Samuel) versus continuous (Berliner) representations of the evaluation space.*

in backgammon that is similar but not identical to structured matching.

Because both checkers and backgammon can be viewed as state-space search problems with large state spaces, there is a great motivation for finding a way to quickly focus on the most promising states (board positions). One way to accomplish this task is through evaluation functions, which assign a measure of goodness to a board position based on features of the board position.

Samuel and Berliner both began their work on evaluation functions by using a simple *linear polynomial method*. In this method, a potential board position is scored using a linear-weighted sum, where each term in the summation pertains to a particular feature of the board position. Each term includes a *variable*, whose value is the numeric measure of goodness of this feature, and a *coefficient*, whose value is measure of the feature's weight of importance.

### Signature Tables and Structured Matching

Both Samuel and Berliner found that a straight linear polynomial method to evaluation had a significant drawback. The context of the evaluation was not accounted for, and therefore, decisions made based on this evaluation function could be in error. For Samuel, the problem is that the linear polynomial method treats individual features as if they are independent of each other and, thus, does not account for interactions between features. Samuel found that this approach greatly inhibited the quality of learning in his checker-playing system.

Thus, he introduced another technique that accounted for interactions between various features in the evaluation of a board position. This technique, called *signature table analysis*, used an n-dimensional array (called a signature table) to store the scores for various board positions. Each dimension of the array represented a feature of the board position. An array's dimension would be divided into as many positions as there were possible

values for the corresponding feature. Thus, if a feature could take on the values -2, -1, 0, 1, and 2, the corresponding array dimension would have five positions. Each cell in the array contained a value that was the score corresponding to the combination of feature values that mapped onto this cell. In this way, a board position's overall score was specific to the nonlinear combination of features that the board position took on.

Although this technique was able to account for interaction between features, it introduced a significant space-complexity problem. For a large number of features, each with a large number of potential values, the size of the array would grow to a prohibitive expanse. Samuel's solution for this problem was twofold. First, he restricted the number of possible values that a feature could take on. Second, he arranged his features into a hierarchy of signature tables. At the lowest level, the signature tables consisted of subsets of the board-position features themselves, with each feature having a greatly restricted range of possible values. Higher-level signature tables used lower-level tables as composite features, and because the number of these lower-level tables was considerably smaller than the number of board-position features themselves, the table output could have a larger range of possible values. Thus, through the use of a hierarchy of signature tables, Samuel was able to arrive at accurate assessments of the worthiness of potential board positions in a reasonable amount of time. This approach greatly improved the quality of play of his checker program and significantly contributed to his research in machine learning. Note, however, that any notion of weighted scoring, the primary activity of the linear polynomial method, was totally abandoned in the signature table approach. Eliminating this feature forced the space of potential feature values to be discrete, whereas the linear polynomial approach allows for a continuous space.

Bylander, Goel, and Johnson (1989) later developed a generic problem-solving method called *structured matching*, which is essentially

a generalization of Samuel's signature table. The architecture of structured matching involves a hierarchy of *matchers*, or truth tables, each mapping a limited set of parameter-value pairs onto a decision for the matcher. Parameters can be data about the world or output from lower-level matchers. Each row of the truth table includes a conjunctive clause of the parameter-value pairs and the resulting output value that occurs if the clause is satisfied. The different rows of the truth table have a disjunctive relationship to each other, with each row containing a different alternative output value.

The inferencing action of structured matching is a goal-driven top-down traversal of the matcher hierarchy. The goal at each level is to determine the output value of the matcher. This determination is made by examining the truth table. If parameter values in the truth table are determined by lower-level matchers, then these matchers are examined. This process continues recursively until the value for the top-level matcher can be determined.

### Hierarchical Weighted Scoring

Another hierarchical static-evaluation technique was developed by Berliner and Ackley (1982). They had done previous work on linear polynomial evaluation functions (Berliner 1977) and found problems similar to those experienced by Samuel. Their domain, like Samuel's, was game playing, but the game was backgammon.

Berliner's earlier program, *BKG*, used a straight linear polynomial function to rate board positions. All board positions (that is, states in the state space) were treated identically, where the same linear polynomial function was used to evaluate each one. Berliner soon encountered problems similar to Samuel's, namely, that the linear polynomial was too rigid to account for the context of the board position. Samuel had described the context in terms of the interrelationships between features. Berliner expressed context by partitioning the space of board positions into state classes, which I discuss later.

In a new program called *QBKG*, Berliner and Ackley, like Samuel, moved from a straight linear polynomial function to a hierarchical representation of the evaluation features. However, they differed from Samuel by not entirely abandoning the linear polynomial method. Instead, primitive features would have weights and scores associated with them, and the weighted scores would be propagated through the hierarchy to obtain scores for higher-level aggregate features

(called *concepts* in Berliner and Ackley's terminology). In effect, they maintained the ability to deal with a continuous space of possible feature values; Samuel's method, however, forced this space to be discrete.

### Continuous versus Discrete Representation

Thus, a major difference between the two approaches to evaluation is one of discrete (Samuel) versus continuous (Berliner) representations of the evaluation space. Berliner and Ackley (1982) criticized discrete representations in evaluation functions, stating that they were fragile and suffered from the boundary problem.

By fragile, they meant that erroneous or noisy data could dramatically skew the results:

In a discrete medical diagnostic system using production rules, for example, an erroneous result on a test could prevent the system from ever making an accurate diagnosis, because the knowledge relating to the actual disease is not used, due to the non-satisfaction of the condition portions of the relevant productions (Berliner and Ackley [1982], p. 214).

They said continuous representations alleviate the fragility problem by ensuring that all relevant factors are taken into account by including them in an overall scoring process.

Actually, continuousness of representation as such is not what alleviates the fragility problem. Rather, it is the use of a compensatory scoring mechanism that dampens the effect of erroneous or incomplete input data. The effectiveness of the compensatory scoring mechanism applies to discrete representations as well.

By the *boundary problem*, Berliner and Ackley were referring to the tendency for systems with large grain sizes to make erroneous decisions when a feature's actual value (as it would appear in a continuous domain) lies in a grey area at the boundary between two possible discrete values and is arbitrarily mapped onto one of these discrete values. The idea here is that fine granularity is less likely to produce errors than coarse granularity. This problem can be significant with structured matching, which imposes a small limit on the number of allowable values that a parameter can take on. In the interest of computational tractability, structured matching forces coarse granularity, thus becoming vulnerable to boundary errors.

However, there are two problems with continuous representation based on scoring methods that make qualitative, discrete repre-

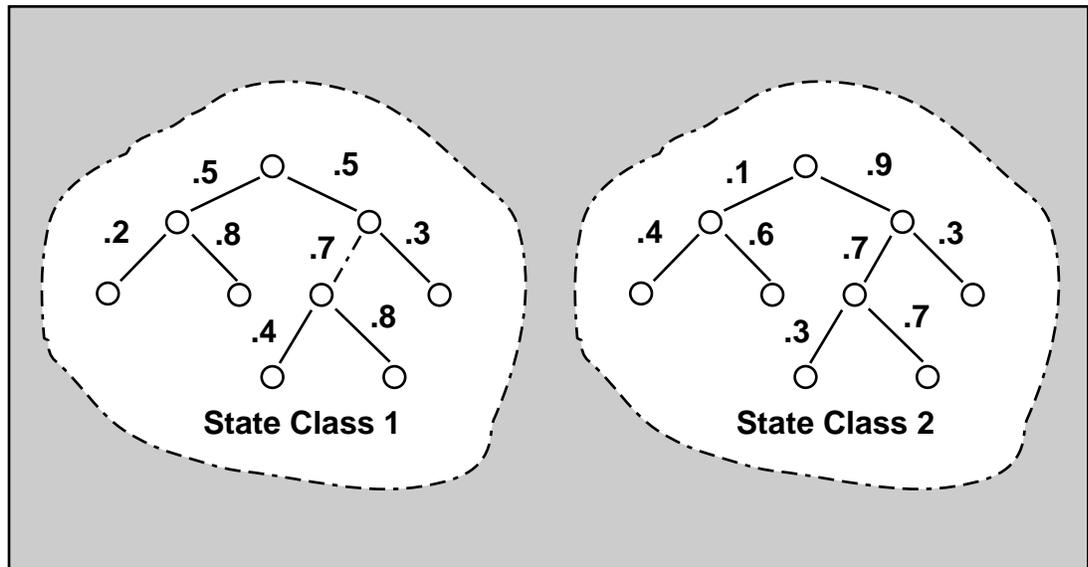


Figure 2. Berliner and Ackley's QBKG Program.

Using the SNAC method, QBKG involves a hierarchy of features. The context of evaluation is expressed using state classes, which allow for dynamic-weight modification. Scores are determined using a linear model and propagated up the hierarchy to determine overall score.

sentations more attractive. The first is that scoring methods cannot account for interactions between variables in the way that conjunctive-clause production rules can. The second is that explanation is much easier with discrete representations.

Berliner and Ackley's system does not appear to account for specific interactions between variables in the way that Samuel's signature tables can. However, by arranging their scoring into a feature hierarchy, they were able to provide explanations at various degrees of abstraction for their system.

**Context in Evaluation**

Another distinction between the Samuel's and Berliner's approaches involves the context of the evaluations they perform. Although both Berliner and Samuel express context in terms of abstraction levels using the hierarchical arrangement of their knowledge groups (signature tables and scorers), they differ in other expressions of the context of the game.

For Samuel, the context is represented specifically by the interactions of the variables at each signature table in the hierarchy, as previously described. Contextual factors and evaluative factors are treated uniformly: All are represented as variables in the system.

In contrast, Berliner's (1979) method, which does not explicitly show the interactions of variables, represents context by dividing possible board positions into *state classes*, which are categories describing overall aspects of the game. For example, one state class contains all *end-game board positions*, where the pieces for each player have already passed each other, and the two sides are racing to take their pieces off the board. This approach is in contrast to another state class, *engaged game*, where the opposing pieces can still land on one another. In this sense, context plays an important role in determining the weightings for the various features of a board position. For example, in an end-game context, it is no longer important to avoid having a piece stand alone because no opposing piece can land on the lone piece. This consideration would be important if the opposing players were engaged. This dynamic, context-based weight adjustment introduces a nonlinear flavor to the SNAC derivation of the linear polynomial evaluation function. Figure 2 illustrates this notion.

**Explanation of Evaluation**

As previously mentioned, discrete knowledge representations tend to be better at facilitating the explanatory power of knowledge-based

systems than continuous representations. When dealing with overall scores, it is difficult to identify just what is wrong with the candidate being evaluated or exactly why one candidate is better than another.

Thus, it would appear that the signature table approach is superior to SNAC in this regard. With QBKG, Berliner and Ackley wanted to maintain the performance advantages of the continuous representation but keep the explanatory power of the discrete representations. They saw two main tasks in this regard: (1) isolate relevant knowledge (pertaining to a query for explanation) from irrelevant knowledge and (2) decide when quantitative changes should be viewed as qualitative changes. They handled the first task using their hierarchical arrangement of features. Thus, explanations at lower levels tended to be narrow and detailed, but explanations at higher levels were broad and unfocused. They handled the second task by partitioning the differences in scores between candidates (for any given feature) into *contexts*, which can be phrased as “about the same,” “somewhat larger,” “much larger,” and so on. In other words, they transformed the evaluation space from a continuous to a discrete representation after all the scoring had taken place, thereby avoiding the pitfall of losing valuable information because of arbitrary early classification. Once the evaluation space became discrete, the qualitative differences in scores could be displayed as explanations for choosing one candidate over another.

## Studies of Evaluation in the Social Sciences

The dichotomy between the Samuel’s signature table approach and Berliner’s SNAC method is mirrored in social science research on decision strategies used by people for candidate evaluation and selection. This section examines some studies that use linear models to emulate expert decision making. It also looks at models of consumer behavior that seem to incorporate both discrete and continuous decision models.

### Psychological Studies of Linear Models

Dawes and Corrigan (1979) conducted extensive studies where they attempted to emulate expert evaluative and classificatory decision making using linear models. Their studies compared the performance of actual expert judges with linear models of the judges’ decision-making processes and found that the

models consistently performed as well as or better than the judges themselves.

Dawes concluded that linear models work well in solving problems with the following structural characteristics: (1) each input variable has a conditionally monotonic relationship with the output, (2) there is error in measurement, and (3) deviations from optimal weighting do not make practical difference. By *conditional monotonicity*, Dawes meant two things: First, the ordinal relationship between an input variable and the output decision is independent of the values of other input variables (much like the assumption underlying Samuel’s and Berliner’s linear polynomial evaluation function). Second, the ordinal relationship between the input and output variables is monotonic; that is, a higher score on an input always results in a higher (or lower, given negative weighting) score on the output. Dawes suggested that most variables can be scaled in such a way that conditional monotonicity is possible.

It is interesting to note that Berliner and Ackley’s SNAC method managed to circumvent the conditional monotone requirement for linear models by introducing the notion of state classes and dynamic weight adjustment. Such a requirement rests on the assumption that the weight of each term in the linear function is constant or, at least, that its sign is constant. With SNAC’s state classes, the values (and even the signs) of the weights could change with changing context. Thus, the relationship between input variable value and output value need not be monotonic.

Error in measurement enhances the appeal of linear models because of the compensatory nature of such models. In other words, even if a particular input variable has an incorrect value assigned to it, other variables that go into the total score compensate for this error. Note that this argument is consistent with Berliner and Ackley’s fragility critique of discrete representations for evaluation functions.

Based partially on Dawes’s research, Chung (1987, 1989) empirically compared the linear model approach to a rule-based approach in terms of inductive knowledge-acquisition methods for classificatory problem types. His study indicated that the relative performance of systems using these approaches differed based on the type of classification problem they were applied against. Specifically, he found that tasks involving conditional monotonicity are good linear candidates, whereas those violating conditional monotonicity are better rule-based candidates. This finding is consistent with the findings of Dawes and Corrigan.

*Ceved is... intended to make it easy for a non-programmer to represent evaluation-type knowledge.*

### Models of Consumer Behavior

Research in consumer behavior has isolated two types of evaluative decision rules for product-purchase decision making; these are called compensatory and noncompensatory decision rules (Wright 1975; Schiffman and Kanuk 1987).

A *compensatory decision rule* is analogous to the linear polynomial method employed by both Samuel and Berliner in the early versions of their evaluation functions and described by Dawes and Corrigan in their study of human judgment. It is called compensatory because positive evaluations on some attributes can balance out negative evaluations on others.

Compensatory decision making can be represented naturally using Berliner's SNAC method because weights of importance for various attributes can explicitly be represented in the SNAC architecture. Also, modifying the compensatory evaluative knowledge of the system is easy with a SNAC-like approach. If an expert or knowledge engineer decides to change the importance level of a particular feature, all s/he has to do is change this feature's weight value. By contrast, the signature table approach (and structured matching) is awkward for representing the compensatory rule. The relative importance of each feature is not explicitly represented but, rather, is implied by the combination of feature values in a pattern of the truth table. This approach makes it difficult for the observer to ascertain which attributes are important and which are not. Also, changing the importance level of a single feature can require making changes to several patterns in a truth table, creating a maintenance nightmare for knowledge engineers dealing with large knowledge bases.

*Noncompensatory decision rules* deal with quick-reject or quick-accept situations. These situations can be expressed in a number of ways. For example, if one says "This candidate is unacceptable if it scores below a threshold for a certain variable," this example illustrates the *conjunctive rule* of consumer behavior. Alternatively, the phrase "I accept this candidate because s/he scores above a threshold on a particular feature" is an expression of the *disjunctive rule*.

The discrete nature of signature tables and structured matchers makes it easy and natural to represent noncompensatory evaluative knowledge. Structured matching in particular allows quick-reject or quick-accept parameters to be evaluated early and, thereby, cuts down on unnecessary computing. By contrast, linear polynomial methods, including SNAC, are not good at representing or reasoning using noncompensatory decision rules. A

continuous representation is unnecessary for such threshold issues. Additionally, the compensatory nature of weighted scoring makes it difficult to minimize the number of variables that need to be resolved to make a decision.

Thus, we see that linear models or variants thereof are generally better at representing compensatory evaluative knowledge, whereas truth tables are better for handling noncompensatory evaluation. Thus, a task-specific architecture for candidate evaluation must have attributes of both approaches to effectively handle the entire realm of evaluation.

### Candidate Evaluation as a Task-Specific Architecture

In the previous discussion, I described some of the literature background of evaluation techniques in AI and social science. I also presented a description and the motivation for the notion of generic tasks and task-specific architectures. One purpose of the research at CIBER is to develop a general, evaluative problem-solving methodology that combines the strengths of Berliner and Ackley's SNAC method and Samuel's signature table. The motivation for this work is to provide environments for nonprogramming domain experts to easily encode evaluative knowledge that can be used in expert systems.

### Developmental Principles for a Candidate Evaluation Task-Specific Architecture

The philosophy guiding the development of the candidate evaluation architecture is based on the following principles:

First, the architecture should allow for all types of evaluative decision making, including both compensatory and noncompensatory evaluation, using both discrete and continuous representations of the evaluation space. The architecture must allow for quick-reject or quick-accept decisions as well as a thorough assessment of the strengths and weaknesses of the candidate being evaluated. It should also allow the evaluation process to be sensitive to non-evaluative, contextual factors in the environment.

Second, the architecture should adhere to the task-specific architecture school of thought and as much to the generic task framework as possible. In particular, its conceptual primitives should be natural for representing evaluative knowledge, and the problem-solving method it embodies should be applicable over a wide range of problem domains.

Third, the architecture should allow for a rich explanatory facility, taking into account combinations of features and expressing various levels of abstraction. The evaluation and explanation process should be easy for the novice to interact with and should provide effective, valid evaluations and recommendations.

Fourth, the architecture should be able to be implemented in an expert system shell. This shell should be directly usable by non-programming domain experts, who will use the architecture framework to encode their knowledge in performing evaluative expert systems. There should be no need for an intermediary AI programmer to encode evaluative knowledge. Thus, the knowledge-acquisition bottleneck should be eased somewhat.

### Overall Description of the Candidate Evaluation Architecture

The candidate evaluation architecture meets these developmental principles in the following ways:

It incorporates all the evaluation methods previously mentioned. For compensatory decision making, it uses a linear model approach that, like Berliner's *SNAC*, provides for dynamic weight adjustment based on context. Like Samuel's signature table approach, it also accounts for the interaction of variables, at least at an abstract level, by mapping combinations of feature ratings onto recommendations. It also provides for quick-reject or quick-accept (noncompensatory) decisions by allowing the developer to set threshold levels for any single feature or composite feature.

The architecture adheres to the task-specific architecture and generic task philosophies in two ways: First, its semantic structure and conceptual primitives are at a level of abstraction that is meaningful for evaluative tasks. The primitives include a hierarchy of composite features (dimensions of evaluation); a set of evaluative questions (equivalent to *QBKG*'s primitive features); a set of contextual questions for dynamic weight adjustment (serving the same purpose as *SNAC*'s state classes); and a set of recommendation fragments that, like Samuel's signature tables, account for interactions of feature ratings. These elements are discussed later. Second, in keeping with generic task requirements, the architecture is applicable across a wide variety of domains. Many problems requiring evaluative reasoning are solvable using this architecture.

The architecture has a rich explanation facility intertwined with its conceptual structure. Like *QBKG*, the hierarchy of features allows for explanation at various levels of

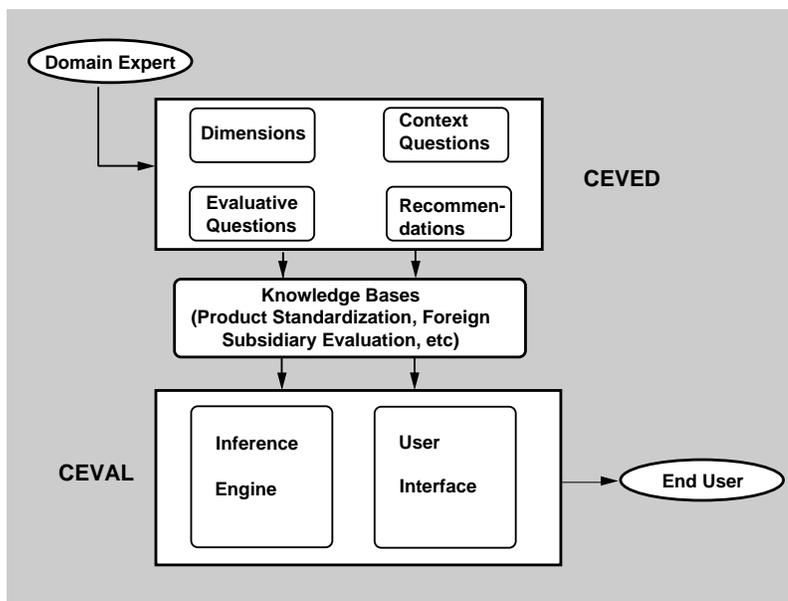


Figure 3. Structural Components and Flow of Knowledge in the Cevd-Ceval System.

The domain expert creates knowledge bases using Cevd. The end user is led through the consultation using Ceval.

abstraction. Also like *QBKG*, there is a postevaluation mapping from the continuous space (score) onto a discrete space (feature rating), which allows for qualitative expressions of the evaluation. In addition, because recommendations are tied to combinations of feature ratings, the system can provide explanations for interactions of variables, like Samuel's signature tables and Bylander's structured matching. Finally, textual explanations can be tied to specific questions or dimensions.

The architecture has been implemented in an expert system shell, including a development environment (Cevd) and a run-time module (Ceval). It is easy to use and has been employed by graduate students and domain experts in international marketing. None of the users had prior computer programming or AI experience, yet they were able to quickly learn to use the tool and have developed a dozen international marketing-related expert systems using the tool.

### The Candidate Evaluation Shell—Cevd and Ceval

In this section, I describe the shell for implementing candidate evaluation. It involves two components: The candidate evaluation editor (Cevd) is the development module, and the

<b>Dimension Name</b>		
Country Environment		
<b>Parent Dimension</b>		
Foreign Subsidiary Performance		
<b>Weight</b>	<b>Ratings</b>	<b>Threshold</b>
30%	Excellent Moderate Poor	85 45 0

Figure 4. A Sample Dimension Entry Screen in Cevd.

The developer uses this screen to enter the dimension's name, parent, importance level (weight), ratings, and threshold scores.

candidate evaluator (Ceval) is the run-time module.

### Candidate Evaluation Editor (Cevd)

Cevd is a development environment intended to make it easy for a nonprogrammer to represent evaluation-type knowledge. Four main types of objects can be represented using Cevd: dimensions of evaluation, contextual questions, evaluative questions, and recommendation fragments (figure 3). In keeping with the requirements for a user-friendly development environment and to avoid the feel of a programming language, Cevd requires only two types of input from the user: menu choice and text processing. The Cevd text-processing facility is for typing in explanations and recommendations; it includes many text-editing features found in standard word processors, including cut and paste and text file import and export.

**Dimensions of Evaluation.** Cevd allows the developer to define a hierarchy of abstracted candidate features, called *dimensions*, that serve as the baseline for evaluating the candidate. A dimension is made up of the following attributes (figure 4): (1) the dimension's name; (2) its parent dimension; (3) a

list of qualitative ratings (a verbal evaluative description of the dimension, for example, "excellent," "fair," or "poor") and corresponding threshold scores (a threshold score is a minimum quantitative score for which a given rating holds); (4) the dimension's *weight of importance*, that is, the degree to which the dimension contributes to the overall score of its parent; (5) an optional explanation message (the developer can write a comment here to help the end user understand what the dimension is measuring); and (6) optional threshold messages (these messages are tied to a threshold score for the dimension; for example, the developer can define a reject message that would appear if the final score for a particular dimension falls below a specified threshold).

Dimensions are related to each other in a parent-child relationship (using the parent attribute), producing a tree-structured hierarchy. A parent dimension's overall score is based on a linear-weighted sum of the scores of its children dimensions. Figure 5 shows a sample dimension hierarchy for a foreign subsidiary evaluation expert system.

**Contextual Questions.** *Contextual questions* are multiple-choice questions designed to establish the weights of importance of various features (dimensions) based on the context. They essentially serve the same purpose as Berliner's state classes. The contextual questions contain the following attributes: (1) the dimensions (features) to which the question pertains, (2) the question's text, (3) a list of multiple-choice answers and a corresponding list of weight-adjustment values (each weight-adjustment value specifies the direction and degree to which the chosen answer will change each associated dimension's weight (using multiplication), and (4) an optional explanation message. During a consultation, the answers that a user gives for contextual questions will determine the final weights that each dimension (aggregate feature) takes on. Contextual questions are asked before evaluative questions.

**Evaluative Questions.** Cevd allows the developer to define multiple-choice evaluative questions that will be presented to the end user during a consultation. Questions are grouped into question sets. Each question set is associated with a lowest-level dimension (that is, a leaf node in the dimension hierarchy).

An evaluative question contains the following attributes (figure 6): (1) the question text, (2) the question's *weight of importance* (which identifies the degree to which this question contributes to the overall score of its question

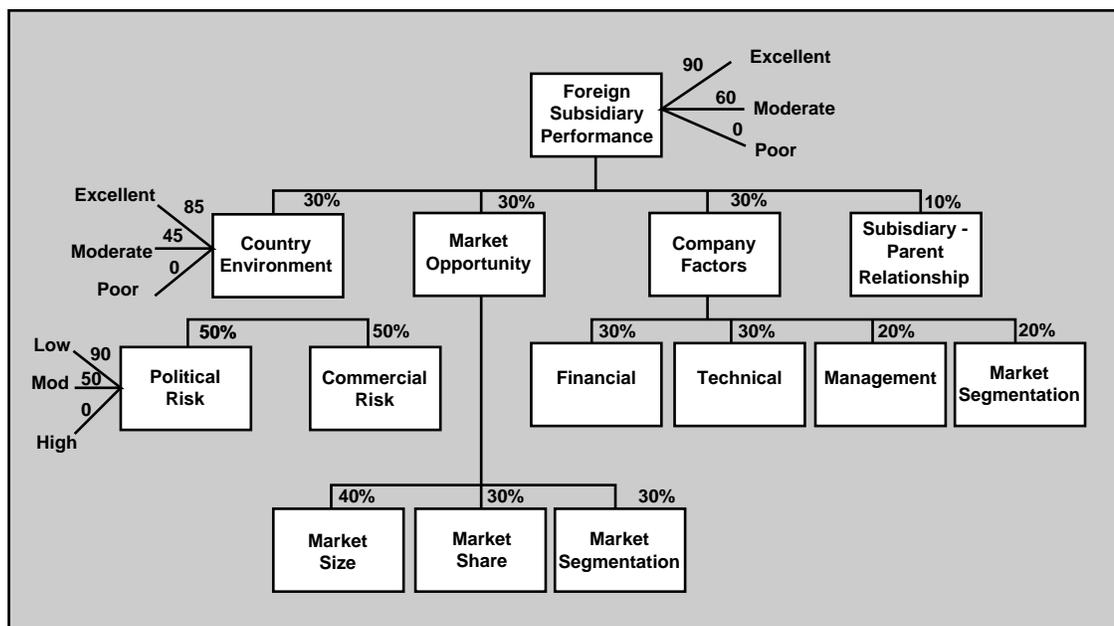


Figure 5. Example Dimension Hierarchy for Foreign Subsidiary Evaluation (some ratings and thresholds are shown).

set), (3) a list of answers and their corresponding scores (the answers are presented to the end user as a menu to select from during a consultation; depending on the answer chosen, its corresponding score will be assigned to this question), and (4) optional threshold and explanation messages (similar to the messages defined for dimensions).

During a consultation, the overall score of a question set is a linear-weighted sum of the questions' weights and their scores based on user answers during a consultation. This score provides the rating for the question set (leaf-node dimension) and is propagated upward to contribute to the scores of the dimension's ancestors in the dimension hierarchy.

**Recommendation Fragments.** Ceved allows the developer to define recommendation fragments and a recommendation presentation strategy. Recommendation fragments are linked to (and triggered by) combinations of dimension ratings. The recommendation presentation strategy controls the order in which recommendation fragments appear and allows some recommendation fragments to suppress others.

A recommendation fragment includes the

following attributes: (1) the recommendation heading (a one-line description), (2) the recommendation fragment's presentation conditions (the condition consists of a list of dimensions and their desired ratings; a recommendation fragment is presented only if the corresponding dimension's have the desired ratings), (3) the recommendation fragment's text, and (4) the recommendation fragment's local presentation strategy (which includes a list of all other recommendation fragments that this recommendation fragment suppresses or prevents from appearing and a list of all recommendation fragments that it is suppressed by; thus, the developer of a candidate evaluation knowledge base can prevent redundant or conflicting recommendation fragments from appearing together).

The global recommendation presentation strategy involves a recommendation ordering strategy and a recommendation suppression strategy. The *ordering strategy* allows the developer to describe, in general terms, the order in which recommendation fragments should be presented to the end user during a consultation. For example, the developer can specify that more abstract fragments come before less abstract ones or that fragments tied to more

<b>Question</b>	
What is the level of price controls in the host country?	
<b>Question Set</b>	<b>Weight</b>
Commercial Risk	15%
<b>Answers</b>	<b>Score</b>
Extensive Moderate Low None	0.00 40.0 70.0 100.0

Figure 6. A Sample Evaluative Question Entry Screen in Ceval.

The developer uses this screen to enter the question's text, weight, possible answers and corresponding scores, and the dimension (question set) to which the question belongs.

important dimensions should appear before fragments tied to less important ones. The *suppression strategy* allows the developer to define, in general terms, which types of recommendation fragments prevail if two or more redundant or conflicting fragments have satisfied their presentation conditions.

**Candidate Evaluator (Ceval)**

Ceval is the run-time inference engine that executes the knowledge bases developed using Ceval. It presents the questions to the users, inputs their answers, scores and rates the dimensions of the dimension hierarchy based on these answers, and presents a final recommendation to the user based on the dimension ratings and the recommendation presentation strategy.

Ceval's inference behavior can be described as a weighted depth-first traversal of the dimension hierarchy. First, contextual questions are asked to determine the weights of the various dimensions in the hierarchy. If the resulting weight of any dimension is zero, this dimension and its subtree are pruned from the search, thereby reducing the

number of questions asked. Then, the depth-first traversal takes place, where the most important (that is, high weight) dimensions are explored first. When a leaf-node dimension (an evaluative question set) is reached, the evaluative questions in this set are presented to the user, and the user's answers are input. Then, Ceval determines the score of the question set using a linear-weighted sum of the questions' weights and the answers' scores and propagates the score up the tree to determine minimum and maximum possible scores for each ancestor of the question set dimension (figure 7). If any dimension's (or question's) score falls below (or above) its quick-reject (or quick-accept) threshold, a message appears recommending to terminate the evaluation and make a reject (accept) decision immediately. In this manner, Ceval implements noncompensatory evaluative reasoning.

After propagating the score up the tree, Ceval attempts to establish the qualitative rating for the ancestor dimensions of the question set. If any ratings can be determined (that is, if the minimum and maximum scores for a dimension are within a range that corresponds to a single rating for this dimension), Ceval triggers any recommendation fragments tied to these dimension ratings. Each triggered recommendation fragment checks its presentation conditions, and if they are satisfied, the recommendation fragment is added to a recommendation agenda list. The ordering and suppression strategies are then used to order and prune the recommendation list.

After the recommendation fragments are processed, Ceval resumes the traversal of the tree. It continues this traversal until either all the questions have been asked or enough questions have been asked to qualitatively rate all the dimensions deemed relevant by the end user. Then, the recommendation fragments that remain on the recommendation list are presented to the user based on the ordering and suppression strategies. Thus, the overall recommendation is a combination of the recommendation fragments whose conditions have been met and those that were not suppressed.

As an example, see figure 8. Assume that the user obtained Excellent for Foreign Subsidiary, Moderate for Country Environment, and Moderate for Commercial Risk. In this case, recommendation fragments 1, 2, and 3 all satisfy their conditions. However, because fragment 2 suppressed fragment 1, fragment 1 is deleted from the final recommendation fragment list. Also, if the ordering strategy indicates that highly abstract recommendation

*Ceval is the run-time interference engine that executes the knowledge bases developed using Cevad.*

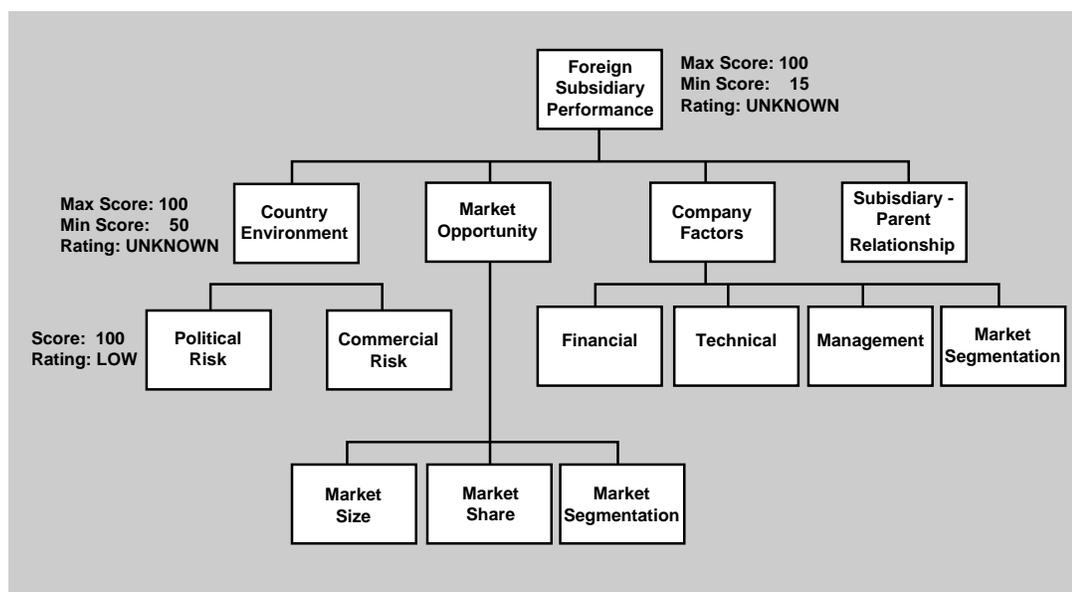


Figure 7. A Possible Scenario of Score Propagation Part Way through a Ceval Consultation.

Here, the user has just completed the Political Risk portion, scoring 100 percent. Rating is known for Political Risk, and max and min scores propagate up the tree. Ratings are still unknown at higher levels in the hierarchy.

fragments appear after more detailed recommendations, recommendation fragment 3 is displayed before recommendation fragment 2.

Ceval allows the user to specify whether s/he wants detailed explanations to appear during the question-and-answer process and determine whether s/he wants the recommendations to appear after each score propagation or only at the end of the entire consultation. In addition, the user can save a consultation for rerunning at a future time and can save recommendations that result from these consultations.

### Strengths and Weaknesses of Cevad-Ceval

As mentioned earlier, one strength of the Cevad-Ceval shell is its ease of use for non-programming domain experts. At CIBER, we have found that the development of expert systems is greatly facilitated by the use of this and other task-specific architecture-oriented shells. Our use of task-specific architectures speeds knowledge acquisition and expert system development because the domain expert is directly involved in encoding his (her) knowledge on the computer. Figure 3 illustrates how the domain expert interacts with Cevad to encode his(her) knowledge.

Another strength is that explanation in

Ceval is expressed in terms of the evaluation task, making it easier for the end user to comprehend. When a user asks why a particular recommendation is given, the system responds by indicating the score-rating of the dimension(s) that resulted in the recommendation. The user can then get further information about the subdimensions or questions that led to this score-rating. Also, the user is shown how important the various dimensions and questions are and how these importance levels were obtained. Thus, the structure of the candidate evaluation architecture causes explanations that are expressed in terms of evaluative reasoning rather than in terms of rule tracing, as in general-purpose shells.

These two strengths result from the task-specific nature of the shell. However, task specificity also leads to a lack of flexibility. Obviously, not all tasks are evaluative in nature. Cevad-Ceval cannot handle non-evaluative tasks. Other shells are needed.

You might notice that the imposition of multiple-choice answers causes the system to be noncontinuous. In fact, the boundary problem cited by Berliner and Ackley is not solved using this tool. However, the fragility problem is solved because of the use of a weighted scoring scheme. In addition, despite the lack of true continuity, there are two char-

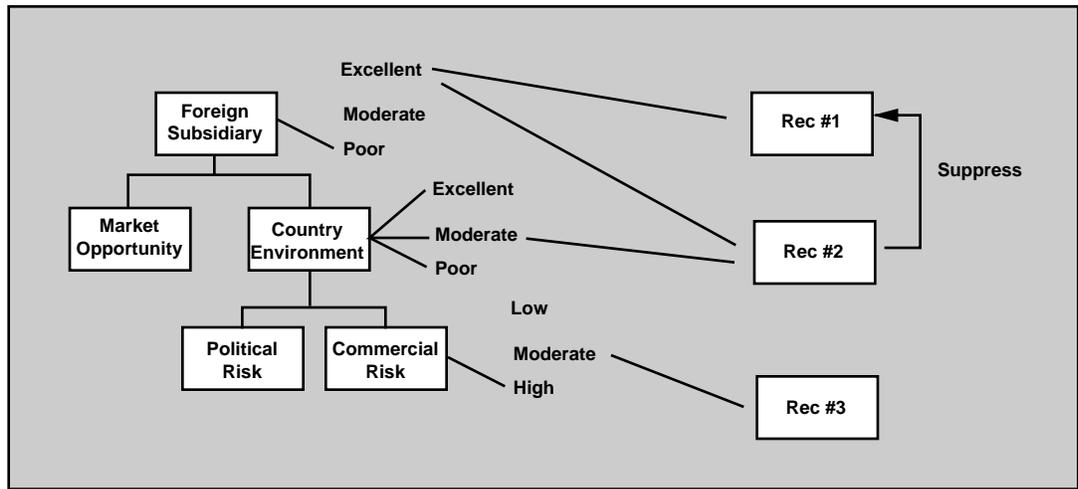


Figure 8. Dimension Ratings and Recommendation Fragments.

These ratings and fragments are linked using trigger-condition links (dashed lines). Recommendation fragments can be linked to each other using suppression links. Suppression links help prevent redundant or conflicting recommendation fragments from appearing in the same recommendation.

acteristics of the candidate evaluation architecture that give it a pseudocontinuous flavor. First, the mapping of answers to scores allows for ratio representation, not merely nominal or ordinal. Second, contextual weight adjustment significantly increases the number of possible points in the evaluation space.

Nevertheless, limiting the user input to multiple-choice format is a weakness in the architecture. At CIBER, we are currently working to improve the flexibility of the shell, with the aim of allowing the user to type in numeric answers that can be mapped onto scores using continuous functions.

### Conclusion

Task-specific architectures are a growing area in AI research and practice. They aid in all phases of expert system development, from knowledge acquisition and problem analysis to the development of performance systems. This article briefly discussed the task-specific architecture approach to knowledge engineering and its implications for knowledge acquisition and compared two methods for the task of evaluation (truth tables and linear models). It then described a new task-specific architecture called candidate evaluation, which incorporates both evaluative methods into a domain-independent framework and a

shell that implements the candidate evaluation architecture called Ceved-Ceval.

Ceved and Ceval provide an easy way to develop and implement interactive expert systems for candidate evaluation tasks. Ceved provides the development interface for easy knowledge acquisition and representation of evaluative expertise. Ceval provides a consultation environment for asking questions and providing recommendations to end users.

This shell was used to develop expert systems in several areas of international marketing, including freight forwarder evaluation, company-readiness-to-export assessment, foreign distributor evaluation, foreign subsidiary evaluation, joint-venture partner selection, and the assessment of product standardization feasibility. In addition, the shell was used to develop a management consulting tool for evaluating a company's corporate logistics strategies.

Ceved and Ceval were written in AION ADS and run on a PC/AT platform.

### Acknowledgments

My thanks go to S. Tamer Cavusgil, executive director of MSU CIBER, Professors Jon Sticklen, Carl Page, and George Stockman of MSU's Computer Science Department, William McCarthy of MSU's Accounting Department, Stephen Schon of Kennen International, and

Omar K. Helferich of A. T. Kearney, Inc., for their valuable insights and support. Funding for this research was partially provided by grants from the U.S. Department of Education and Ameritech, Inc. Software development assistance was provided by the programming staff at A. T. Kearney, Inc.

## References

- Beggs, D., and Lewis, E. 1975. *Measurement and Evaluation in the Schools*. Boston: Houghton Mifflin.
- Berliner, H. 1979. On the Construction of Evaluation Functions for Large Domains. In Proceedings of the Sixth International Joint Conference on Artificial Intelligence, 53–55. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Berliner, H. 1977. Experiences in Evaluation with BKG—A Program That Plays Backgammon. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 428–433. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Berliner, H., and Ackley, D. 1982. The QBKG System: Generating Explanations from a Non-Discrete Knowledge Representation. In Proceedings of the Second National Conference on Artificial Intelligence, 213–216. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Brown, D., and Chandrasekaran, B. 1986. Knowledge and Control for a Mechanical Design Expert System. *IEEE Expert* 1(7): 92–100.
- Bylander, T., and Chandrasekaran, B. 1987. Generic Tasks in Knowledge-Based Reasoning: The “Right” Level of Abstraction for Knowledge Acquisition. *The International Journal of Man-Machine Studies* 26:231–243.
- Bylander, T.; Goel, A.; and Johnson, T. 1989. Structured Matching: A Computationally Feasible Technique for Making Decisions, Working Paper, The Ohio State Univ.
- Chandrasekaran, B. 1986. Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert* 1(3): 23–30.
- Chandrasekaran, B. 1983. Toward a Taxonomy of Problem-Solving Types. *AI Magazine* 4:9–17.
- Chung, H. 1989. Empirical Analysis of Inductive Knowledge-Acquisition Methods. *SIGART Newsletter (Special Issue on Knowledge Acquisition)* 108:156–159.
- Chung, H. 1987. A Comparative Simulation of Expert Decisions: An Empirical Study, Information Systems Working Paper, 5-88, Anderson Graduate School of Management, Univ. of California at Los Angeles.
- Clancey, W. J. 1985. Heuristic Classification. *Artificial Intelligence* 27:289–350.
- Dawes, R., and Corrigan, B. 1979. Linear Models in Decision Making. *Psychological Bulletin* 81(2): 95–106.
- Edwards, W., and Newman, J. 1982. *Multiattribute Evaluation*. Beverly Hills, Calif.: Sage.
- Gaschnig, J.; Klahr, P.; Pople, H.; Shortliffe, E.; and Terry, A. 1983. Evaluation of Expert Systems: Issues and Case Studies. In *Building Expert Systems*, eds. F. Hayes-Roth, D. Waterman, and D. Lenat, 241–280. Reading, Mass.: Addison-Wesley.
- Newell, A. 1982. The Knowledge Level. *AI Journal* 19(2): 87–127.
- O’Keefe, R. 1989. The Evaluation of Decision-Aiding Systems: Guidelines and Methods. *Information and Management: The International Journal of Information Systems Applications* 17(4): 217–226.
- Punch, W.; Tanner, M.; Josephson, J.; and Smith, J. 1990. PEIRCE: A Tool for Experimenting with Abduction. *IEEE Expert* 5(5): 34–44.
- Samuel, A. 1967. Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress. *IBM Journal of Research and Development* 11(11): 601–617.
- Samuel, A. 1959. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development* 3:211–229.
- Schiffman, L., and Kanuk, L. 1987. *Consumer Behavior*. Englewood Cliffs, N.J.: Prentice-Hall.
- Steels, L. 1990. Components of Expertise. *AI Magazine* 11(2): 29–49.
- Sticklen, J.; Chandrasekaran, B.; and Bond, W. 1989. Distributed Causal Reasoning. *Knowledge Acquisition* 1:139–162.
- Sticklen, J.; Smith, J.; Chandrasekaran, B.; and Josephson, J. 1987. Modularity of Domain Knowledge. *International Journal of Expert Systems: Research and Applications* 1:1–15.
- Wright, P. L. 1975. Consumer Choice Strategies: Simplifying vs. Optimizing. *Journal of Marketing Research* 12:60–67.



**Michel Mitri** is a senior knowledge engineer at the Center for International Business Education and Research (CIBER) at Michigan State University (MSU), where he designs expert system software for international marketing applications. He is a Ph.D. candidate in computer science at MSU, majoring in AI. He is also a founding partner of Kernen International, a firm devoted to providing consulting services and decision support software for international business. Mitri is the developer of the Cevd and Ceval programs.