

# CHAPTER 6

## Review of Recent Developments and Objectives: Specific Projects

We now examine specific projects within the general application areas noted in Chapters 2, 3, and 4, broadly grouped into projects that support the objectives in the previous chapters, that is, of providing more complete performance, supporting integration of models, and improving model usability. The format of the projects follows the general format used in Pew and Mavor (1998). Where appropriate, this summary also comments on the feasibility and concerns that may arise if the projects are implemented in Soar, a current common approach for computer-generated forces. The estimates are uniformly optimistic to allow comparisons. The estimates are in terms of programmer or analyst time, and assume adequate supervision and cooperation with other organizations.

### 6.1 Projects Providing More Complete Performance

The projects presented here address the issues raised in Chapter 2. They are grouped into three main categories. We also note some potential additional uses for models of behavior as well as current uses in synthetic environments.

#### 6.1.1 Gathering Data From Simulations

It is very clear and consonant with Pew and Mavor (1998, chap. 12) that data need to be gathered to validate models of human and organizational behavior. An approach at which they hint is to instrument synthetic environments. Synthetic environments should be instrumented not only for playback, but also in a way to provide data for developing and testing models. While the data are not directly equivalent to real-world behavior, as the environment becomes more realistic the data should become more realistic as well.

A uniform representation for data from simulations should be created. This representation should be readable by humans, at least in some formats.

Creating summary measures will also be necessary. Otherwise the sheer volume of data may preclude its analysis. The individual actions of control are not likely to be useful on their own (e.g., pressing an accelerator) but will be required to build higher-level summaries. Creating these summaries is likely to represent an additional research agenda item requiring AI, domain knowledge, and some understanding of behavioral data.

The playback could be quite large for developing models. Analysis of data from synthetic environments might also provide insights into the quality of the simulation (e.g., how quickly someone could act and whether they were limited by the simulation's ability to display information) and provide insights about the implementation of doctrine (e.g., how often tank crews actually follow doctrine). When done in cooperation with a simulator's

developers, the resources required for this task could be quite modest. Otherwise, it could take some time. Developing initial automated summaries is a 6- to 12-month effort.

### 6.1.2 Understanding Expectations of Behavior

Providing realistic behavior requires understanding what people expect from other people and what aspects of an adversary are necessary for training. (These two may be quite different.) In one sense, this means understanding the Turing test: what is necessary to appear human? More important, however, is knowing what is necessary to train people. A model that passes the Turing test and appears human might be weaker or unusual in some way. Thus, training with the model might not result in transfer of learning or result in learning an incorrect behavior.

A useful exercise would be to study which characteristics of behavior make a model appear human (so-called believable agents). The model must start with competencies; it must be able to perform tasks. It should also include errors, hesitations, and variations in behavior.

Work with the Soar Quake-bot on how firing accuracy and movement speed make agents believable is an example of what is required (Laird & Duchi, 2000) to understand what people think is human. The Soar Quake-bot has been evaluated on such things as firing accuracy with observers asked to rate its humanness. The measure of humanness, however, does not reveal how good the Quake-bot is with respect to training. Nor does it reveal what aspects of the Quake-bot should be made more (or less) human to improve training. The current belief is that appearing (or behaving) more human makes a better opponent to train against, but we do not know of any evidence to support this belief.

Another example is the Fuzzy Logic Adaptive Model of Emotions (FLAME). In this work (Seif El-Nasr, Yen, & Ioerger, 2000), several models of a pet that followed the user around in a virtual house were tested for believability. The model that included learning and fuzzy behavior was the judged the best. While not a complete test, this type of project starts to find out what makes agents believable through tests. In this example, learning and emotions were both helpful.

A useful 6-month to 1-year study would be to examine a range of models and humans in a synthetic environment, noting observers' comments and behavior toward a range of behavior. It might be that these aspects make an agent appear human, but it might also include implicit effects, such as second-order (or lagged) dependencies in behavior. The results would be important for training and also useful for creating models used in analysis. This project is similar to, but conceived of separately from, a similar call proposed by Chandrasekaran and Josephson (1999) to develop a better understanding of how to and how far to develop models.

The results are also essential for understanding how to help modelers. The results will point out the most likely mismatches to be left in models because modelers do not consider such behavior abnormal. This will provide suggestions about where comparisons with data are particularly needed by models. As this is basically experimental work, less resources are needed, but the time to run the experiment and analyze the results will take up to a year for preliminary studies.

### 6.1.3 Including Learning in Models

Work on creating agents in synthetic environments has been successful, however, one particularly useful aspect that has not been modeled is learning. A worthwhile project would be to take a learning algorithm and put it to use within a synthetic environment, either as part of a problem solver or as an observer. There are a variety of learning algorithms and models that would be appropriate. Some examples include: connectionism; one of the hybrid learning architectures developed within the ONR program (Gigley & Chipman, 1999); Programmable User Models (Young, Green, & Simon, 1989b); Soar with learning turned on; ACT-R; EPAM; or any of a wide variety of machine-learning algorithms.

Creating a model that learns will be difficult. This task is large and would allow multiple subprojects to be attempted. It could be supported by a wide range of resources. Including learning with problem solving has been difficult in the past, but it is likely to lead to more accurate agents that may be useful for testing and developing tactics.

Soar models exist that function fairly well in a synthetic environment. If these could be used, a small project of a programmer-year or two should be able to create an initial model that learns in a synthetic environment. Attaching a learning component to find regularities in behavior is likely to take at least that much time. Both projects would provide potential PhD topics and are broad enough to be supported by a wide range of resources.

### 6.1.4 Including a Unified Theory of Emotions

There are three specific projects related to modeling emotions and other behavioral moderators in architectures that we can propose: (1) adding general emotional effects, (2) adding reactive emotions, and (3) testing emotional models with performance data. While work is ongoing implementing models like this in Soar (Chong, 1999; Gratch, 1999) and ACT-R (Belavkin et al., 1999), the domain is large. Projects can range from a few months to implement a simple emotional effect to several years or decades to incorporate a significant amount.

***Adding general emotional affects.*** As noted above, it is possible to start to realize emotions and affective behavior within toolkits like Sim\_Agent and general cognitive architectures like ACT-R and Soar. Including emotions will provide a more complete architecture for modeling behavior and a platform for performing future studies of how emotions affect problem solving. Including emotions may also provide a way to duplicate personality and provide another approach to account for appropriate variations in behavior. Hudlicka (1997) provides a list of intrinsic and extrinsic behavior moderators (similar to the categories suggested in Ritter, 1993b) that could be modeled. Boff and Lincoln (1988) provide a list of regularities related to fatigue and other related stressors that might be considered for testing against a general model of emotional effects. For example, by making ACT-R's motivation sensitive to local performance (rule successes and failures), we have fit the Yerkes-Dodson law (Belavkin & Ritter, 2000).

These models should move from applying to a single task to multiple tasks. They would then become modifications to the architecture and thus reusable.

**Adding reactive emotions.** Modeling reactive and long-term moderators as well as slower-acting behavioral moderators is worthwhile. The effect of stress also changes the state of the competence in important ways. A proportion of troops engaged in active combat will become ineffective as a result of fear and stress-fatigue. Stress would also be increased by the number of casualties taken by a given platoon, length of time without sleep, weather conditions, perceived chance of survival, and so on. Modeling these effects at the micro-level of individuals, following known distributions, would advance the realism of simulations in interesting ways and support teaching existing doctrine.

In production system architectures, these emotions can initially be implemented by changing the decision (rule-matching) procedure, adding rules to make parameter changes, and by augmenting working memory to include affective information (e.g., an operator or state looks good or bad). These types of changes are being applied to an existing model, which matches adult behavior well, to better match children's more emotional behavior (Belavkin et al., 1999). These emotional effects should improve the match to the children's performance by (1) slowing down performance in general, (2) slowing down initial performance as the child explores the puzzle driven by curiosity, and (3) abandoning the task if performance is not successful. This work should be extended and applied more widely.

**Testing emotional models with performance data.** Many of the theories of emotions proposed have not been compared with detailed data. Partly this may be because there is not always a lot of data available on how behavior changes with emotions. It is no doubt a difficult factor to manipulate safely and reliably. But the models must not just be based on intuitions.

The use of simulators may provide a way to obtain further data with some validity. Better instrumentation of some primary features of emotions (e.g., heart rate, blood pressure) is providing new insights (Picard, 1997; Stern, Ray, & Quigley, 2001) and will be necessary for testing models of emotions.

Some argue that emotions are necessary for problem solving. Examples of brain-damaged patients (e.g., Damasio's Elliot [1994]), who have impaired problem solving and impaired emotions, are put forward. It is not clear that emotions per se are required, or if multiple aspects of behavior are impaired as well as emotions by the trauma. Others argue from first principles that emotions (realized as changes in motivation due to local success and failure during problem solving in this example) can improve performance (Belavkin, 2001). A model compared with data may help answer whether this is true. Clearly, AI models of scheduling do not have the same troubles scheduling an appointment despite their lack of emotion.

### 6.1.5 Including Errors

There are two premises that underpin the modeling of erroneous behavior. The first premise is that the attribution of the label *error* to a particular action is a judgment made in hindsight. The identification of the erroneous action forms the starting point for further investigation to identify the underlying reasons *why* a particular person executed that particular action in that particular situation. In other words, the erroneous action arises as the result of a combination of factors from a triad of sources: the person (psychological and

physiological factors), the system (in the most general sense of the term), and the environment (including the organization in which the system is deployed).

The second premise acknowledges that an error is simply another aspect of behavior. In other words, any theory of behavior should naturally encompass erroneous behavior. The behavior can be judged as erroneous only with respect to a description of what constitutes correct behavior.

Once these premises are accepted, it becomes apparent that modeling erroneous behavior is actually an inherent and important part of modeling behavior. If the psychological and physiological limitations of human behavior are incorporated into a model of human behavior, then particular types of erroneous behavior should naturally occur in certain specific situations. The corollary of this argument is that an understanding of erroneous behavior can be used as the basis for evaluating models of behavior. So, if a human performs a task correctly in a given situation, the model should also be able to perform the task correctly in the same situation. If the situation is changed, however, and the human generates erroneous behavior as a result, the model should also generate the same erroneous behavior as the human in the new situation, without any modifications being required to the model.

Modeling error therefore depends on understanding the concept of error—its nature, origins, and causes—and central to this is the need for an accepted means of describing the phenomenon (Senders & Moray, 1991). In other words, a taxonomy of human error is required with respect to these tasks.

The utility of the taxonomic approach, however, depends on the understanding that the taxonomy is generated with a particular purpose in mind. In other words, the taxonomy has to reflect:

- A particular notion of what constitutes an error.
- A particular level of abstraction at which behavior is judged to be erroneous.
- A particular task or domain.

There is a need to be very clear about the classes of errors and their origin in the models so that the appropriate ones can be included. In the military context, for example, a major source of error is communication breakdown. One approach to developing an appropriate taxonomy of errors for the military domain is to use the scheme that lies at the heart of the CREAM (Hollnagel 1998). The CREAM purports to be a general purpose way of analyzing human behavior in both a retrospective and a predictive manner. Although the method was developed on the basis of several years of research into human performance, mainly in the process industries, it is intended to be applicable to any domain.

The CREAM uses a domain-independent definition of what constitutes an erroneous action (also called error modes or phenotypes). One of the goals of the CREAM is to be able to identify the chain of precursors for the various error modes. Identifying the chain is achieved by means of a set of tables that define categories of actions or events. At the highest level, there are three types of tables:

- Human (or operator),
- Technological (or system), and
- Organizational (or environment).

Within these categories there are sub-category tables. So, for example, the human tables include observation, interpretation, planning, and so on.

The individual actions or events are paired together across tables on the basis of causality or, to use a more neutral term, in a *consequent-antecedent relationship*. When the CREAM is used to analyze a particular accident or incident retrospectively, the aim is to build up the list of possible chains of events and actions that led to the accident or incident.

The contents of the tables are domain-specific, so the first step in developing the taxonomy for agents in synthetic environments depends on identifying the appropriate categories of events and actions for the military domain. These categories and the links between individual actions or events will be generated from a combination of knowledge elicited from domain experts and a review of the appropriate literature.

The second step is to generate the possible chains of actions and events that precede the various error modes, based on information available from reports of real accidents or incidents. This process will involve access to desensitized accident or incident reports—like those used in the Confidential Human Factors Incident Reporting Programme (CHIRP; Green, 1990) originally operated by the RAF Institute of Aviation Medicine—that can be analyzed and coded using the domain-specific CREAM tables generated in Step 1. Where omissions from the tables are detected, or links between actions do not already exist, these should be added to the tables.

The possible causal chains of events or actions generated by the second step will provide the basis for a specification of behavior in a particular situation. Models of behavior should yield the same sequences of actions and events in the same circumstances. The specification of behavior can therefore be used to test the models of behavior for compliance, during development, with the model being modified as appropriate to match the specification.

In addition, the results of the analysis of the incident behavior provide a basis for evaluating the veracity of synthetic environments. Performance in the real world (as described in the incident reports) can be compared with the way people behave when performing in the synthetic environment. There should be a high degree of correspondence between the two. If there is a mismatch, the mismatch suggests that there is a difference between the real world and the synthetic environment, which may be worth further investigation to identify the source of the difference.

One other beneficial side effect of the CREAM analysis is that the resultant chains of actions and events can be used in training personnel to manage error. If common chains of actions or events can be identified, it may be possible to train personnel to recognize these chains, and take appropriate remedial action before the erroneous action that gives rise to the incident is generated.

Initial models that include erroneous behavior can best be created with an existing model. One to three years of work should lead to an initial model that includes some errors and has been validated against human behavior.

### 6.1.6 Including a Unified Theory of Personality

It would be useful to identify features that lead to modeling personality, problem-solving styles, and operator traits. While models that choose between strategies have been created, there are few models that exhibit personality by choosing between similar strategies (although see Nerb, Spada, & Ernst, 1997, for an example used to put subjects in a veridical, but artificial, social environment). Personality will be an important aspect of variation in behavior between agents.

Including personality requires a task (and the model) to include multiple approaches and multiple successful styles. It is these choices that can thus appear as a personality. If the task requires a specific, single approach, it is not possible to express a strategy. Psychology, or at least cognitive psychology, has typically not studied tasks that allow, or particularly highlight, multiple strategies. Looking for multiple strategies has also been difficult because it requires additional subjects and data analysis that before has not represented real differences in task performance. Differences in strategies, however, lead to variance in behavior (e.g., Delaney, Reder, Staszewski, & Ritter, 1998; Siegler, 1987).

There appear to be at least the following ways to realize variance in behavior that might appear like personality: learning, differences in knowledge, differences in utility theory and initial weighting, and differences in emotional effects. Including a subset of these effects in a model would fulfill a need for a source of regular, repeatable differences between agents in a situation.

All of the current cognitive architectures reviewed here and in Pew and Mavor (1998) can support models of personality. These types of changes should be straightforward, as long as there are multiple strategies. In Soar, personality can be expressed as differences in task knowledge, as well as differences in knowledge about strategy preferences either absolutely or based on different sets of state and strategy features. ACT-R appears to learn better and faster which strategy to use compared with a simple Soar model, but ACT-R requires additional state features (Ritter & Wallach, 1998). Models in both architectures can, however, modify their choice of strategies. The role of (multiple) strategies has been investigated within the EPAM architecture in several tasks, such as concept formation (Gobet et al., 1997) and expert memory (Gobet & Simon, 2000; Richman, Gobet, Staszewski, & Simon, 1996; Richman et al., 1995).

These models could also be crossed with emotional and other non-cognitive effects to see how personality types respond differently in different circumstances (broadly defined). This could even be extended to look at how teams with different mixes of personalities work together under stress.

The amount of work to realize a model in this area will depend on the number of factors taken into account by the model. Providing a full model of personality and how it interacts with tasks and with other models is a fantasy at this point. However, a minimal piece of work would take an existing model and give it more of a personality. A more extensive

project over a year or two would apply several of these techniques and see how it starts to match human data.

### 6.1.7 Including a Model of Situation Awareness and Rapid Decision Making

Novices have to do problem solving. Experts can do problem solving but save effort (or improve their problem-solving performance) by recognizing solutions based on the problem. Viewed broadly, a model that does this transition starts to provide an explanation of situation awareness and Rapid Decision Making (RDM) as a result of expertise and recognition.

Able (Larkin, 1981) and its recent re-implementations (Ritter et al., 1998b) provide a simulation explaining the path of development from novice to expert in formal domains (i.e., those where behavior is based on manipulated equations such as physics or math). The early (or barely) Able model works with a backward-chaining approach, that is, it starts with what is desired and chooses domain principles to apply based on what will provide the desired output. This approach is applied recursively until initial conditions are found. The chunking mechanism in Soar gives rise to new rules that allow the model to use a forward-chaining method that is faster. That is, from the initial conditions new results are proposed. The rules are applied until the desired result is found. Students at the University of Nottingham have applied the Able mechanism to several new domains. Their examples are available at [www.nottingham.ac.uk/pub/soar/nottingham/student-projects.html](http://www.nottingham.ac.uk/pub/soar/nottingham/student-projects.html).

Work could be done to translate this mechanism, which has worked in Lisp and in several versions of Soar, into other architectures and extend it from a simulation to a full process model. This would require a rather modest amount of effort, less than a programmer-year to get started if the programmer was familiar with Soar. Applying it in a realistic domain would take longer.

### 6.1.8 Using Tabu Search to Model Behavior

The internal architecture of a combatant might be constructed from a perceptual module that is closely coupled to the synthetic environment and can be modified by plug-in items that alter the incoming data to be processed (night-vision aids, etc.). The results of perception are crudely classified using a learning system such as a multi-layer perceptron, which triggers a rapid emotional response and consequent reactive behavior. This behavior might be generated using an SDM that finds the nearest match to previous scenarios and is capable of producing a sequence of outputs rather than a single-state result. Both perception and emotional response are calibrated by a perceptual and personality model that may be unique to individual entities, albeit assigned from a known distribution.

The cognitive processing would be rule-based using an established cognitive model for example, ACT-R, with planning activities augmented by a Tabu search. There would be interactions between the state of the entity (including its emotional state) and the cognitive processing based on psychological data on human performance under stress. This approach is similar and perhaps a generalization of Sloman's meta-architecture, and the Soar and PSI architectures.

## 6.2 Projects Supporting Integration

The projects presented here roughly address the issues raised in Chapter 3. Integration is approached in two ways here: integrating model components and integrating the model with simulations in more psychologically plausible ways. Several projects described in this subsection could be equally at home in the set of projects for making modeling routine because the two areas are related.

### 6.2.1 Models of Higher-Level Vision

It has been argued that an understanding of higher-level vision is necessary for continued development of models in synthetic environments (Laird, Coulter, Jones, Kenny, Koss, & Nielsen, 1997) and we agree (Ritter et al., 2000). Neisser's (1976) perceptual cycle is just starting to be explored with models.

There are several areas of Higher-Level Vision (HLV) that are of particular interest for military modeling. These areas include:

- How information from long-term memory indicates incoming danger or serious change in the environment.
- How HLV directs attention.
- How HLV integrates various aspects of information, or integrates information occurring at different times.
- How HLV can be used to facilitate learning.
- How HLV can be used in planning and problem solving.

To put it simply, HLV is at the interface between Lower-Level Vision (LLV) and postulated memory entities such as productions, schemas, concepts, and so on. At the present time, this interface is poorly understood, perhaps because LLV and long-term memory are not understood in a sufficiently stable way. (However, see Kosslyn & Koenig, 1992, for neuropsychological hypotheses about HLV.)

Most models of cognition such as Soar and ACT-R (actually, most architectures reviewed by Pew & Mavor, 1998) use modeler-coded information, which avoids dealing with the interface between LLV and long-term memory constructs. Neural nets for vision have been used to go from pixel-like information to features or even higher but have not been incorporated into higher-cognition models. CAMERA (Tabachneck-Schijf, Leonardo, & Simon, 1997), and to a certain extent EPAM (Feigenbaum & Simon, 1984; Richman & Simon, 1989), explore ways in which features may be extracted from low-level representation, and may be combined into long-term memory constructs.

The relationship of HLV and problem solving is undoubtedly an area where more research should be carried out. For example, modeling instruction and training requires a theory of how low-level acoustic input merges with low-level visual input and connects to long-term memory knowledge. In some cases vehicles and gunfire will be heard rather than seen and sounds will direct visual attention in the appropriate direction. Perceptual models of hearing are also well-developed and exploited with dramatic success in, for example, the

MPEG-2 compression standard that is likely to form the basis for much broadcast and recorded sound in the future. The variance among individuals is large for both auditory and visual perception, and both processes are degraded temporarily or permanently by intense overload, as is likely in a military environment.

Work extending this approach to create integrated architectures (Byrne et al., 1999; Hill, 1999; Ritter & Young, 2001) is ongoing. Significant progress will require at least a year-long project, and a longer period would be more appropriate.

## 6.2.2 Tying Models to Task Environments

Tying cognitive models to synthetic environments in psychologically plausible ways should be easier. There are two approaches that seem particularly useful and plausible that we can ground with particular suggestions for work. They are consistent with Pew and Mavor's (1998, p. 200) short-term goal for perceptual front-ends.

The first approach is to provide a system for cognitive models to access ModSAF's display and pass commands to it. This approach has the advantage that it hides changes in ModSAF from the programmer/analyst and from the model. The disadvantage is the need for ModSAF experts, programmers, users, time, and money to make it work. There has been such a system created for Soar models to use ModSAF (Schwamb, Koss, & Keirse, 1994), but it is our impression that this system, although it was quite useful, needs further development and dissemination.

The second approach is to create a reusable functional model of interaction based on a particular graphics system or interface tool (as does the Nottingham Functional Interaction Architecture and ACT-R/PM). A functional rather than a complete model may be more appropriate here as a first step. This functional approach has been already created in Tcl/Tk (Lonsdale & Ritter, 2000), Garnet and Common Lisp (Ritter et al., 2000), Visual Basic (Ritter, 2000), Windows bitmaps (St. Amant & Riedl, 2001), Windows 98 objects (Misker, Taatgen, & Aasman, 2001), and most recently in JAVA. They could be created in Amulet, X-windows, Delphi, or a variety of similar systems, each of which allows models to interact with synthetic environments through a better programming interface. A functional model would then provide the necessary basis for improving the accuracy and psychological plausibility of interaction.

This approach to providing models access to information in simulations could also support creating cognitive models in general, such as for problem solving, working memory, and the effect of visual interaction. These could be later assimilated back into models and architectures in the synthetic environments.

An excellent programmer very familiar with their language can now create an initial system in about two weeks. Integrating and applying these models takes several months to a year.

## 6.2.3 Ongoing Review of Existing Simulations

To provide for reuse and to understand the current situation, a review of simulation systems used (for as broad a geographic region as possible, working with allied nations if possible) should be created that is similar to the listing in Pew and Mavor (1998, chap. 2

Annex). This listing, for example, could initially be created by an intercalated year (co-op) student and then maintained as part of standing infrastructure. This listing could provide an initial basis for understanding what the total needs were and the totality of current simulation efforts. While the U.S. Defense Modeling and Simulation Office may do this in the United States, we do not know of similar efforts in the United Kingdom.

### 6.2.4 Focus on a Flagship Task

Supporting all the uses of synthetic forces as shown earlier in Table 1.1 with a single model of behavior is probably impossible in the short term. The uses of simulations in operations research, training individual group behavior, and examining new materials or doctrine are too disparate to be met by a single approach. While the various levels and uses of simulations mentioned here are related by the real world they all represent, it does not appear to be possible in the next 5 to 10 years to integrate them to the extent to which the real world is integrated.

While there may be some systems that allow multiple use, and there will certainly be some reuse between these areas, a focus for work must be selected. Therefore, a more narrow focus on the most important uses should be adopted by funding agencies. Taking a more focused approach appears to be happening in several places already. A selective focus on the most approachable or natural set of uses is more likely to be successful in the short term and may provide a better foundation upon which to build in the long term. Discussion of these issues should be grounded, if possible, with a set of potential uses with possible systems and domains that will be used in the next 5 to 10 years. Complete unification is not likely in that time period, nevertheless significant reuse should be sought.

Having a focus would also support the choice of a specific application. Applications can then be chosen with a user audience in mind. Having a specific audience will help the application to be useful and seen as useful by a well-defined user community.

Work that attempts to serve too many needs will serve all of them poorly. Projects and research programs will have to pick a domain and an application (or two), and work with them. This application could be an existing use or application or it could be a new use. Work with simulations for training often have high payoffs. Augmenting existing training would be a natural place to consider starting.

The students being trained could also be used to help test the simulation. Apocryphal tales from MIT suggest that building computer-based tutors to deliver instruction is as useful for learning as using the resulting tutors. Creating and validating these models would be good training for such students as well.

### 6.2.5 A Framework for Integrating Models With Simulations

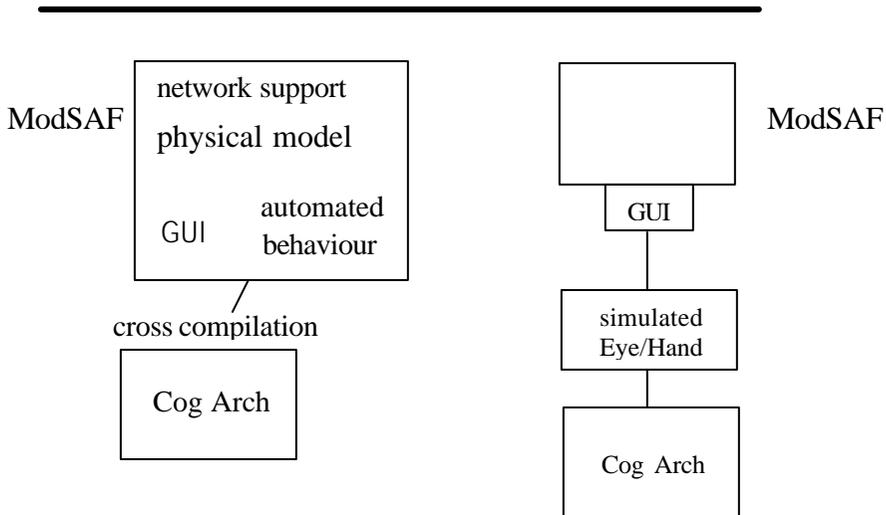
Perhaps the most significant current requirement is a way to integrate multiple cognitive and behavioral architectures into synthetic environments. Currently, it takes a large amount of effort to introduce new models of behavior and connect them directly to simulations via the Distributed Interactive Simulation (DIS) protocol. Coupling cognitive architectures to a simulation via ModSAF is probably marginally easier because ModSAF, while difficult to use, provides physical models and an interface to the network. The left-hand side of Figure

6.1 shows the organization of systems like Tac-Air Soar that interact with ModSAF to generate behavior.

A worthwhile medium- to long-range goal would be to develop utilities to support making a tool like ModSAF even more modular. The core activities of supporting communication across the network for simulation and supporting the physical model need to be provided, but are not of particular interest for modeling behavior.

Efforts have attempted to provide similar interfaces for Soar; however, they have never been fully successful. They have made hooking up Soar easier but have not yet made it easy (e.g., Ong, 1995; Ong & Ritter, 1995; but also see the most recent work by Jones, 2001, and Wallace, 2001). Work on the Tank-Soar simulator (provided as a demo in the latest release of Soar, Soar 8.3) might provide a path for this.

The right-hand side of Figure 6.1 shows how future systems might interact with ModSAF using the same interface that users see through a simulated eye and hand designed to allow models to interact with synthetic environments (Ritter, Jones, Baxter, & Young, 1998a). The interface to the physical simulation could no doubt be made more regular and easier to use so that other architectures, such as Sim\_Agent, could be hooked up to it. We suspect this project might take a good programmer familiar with ModSAF about half-time over a year because we had a similar system built in 2 weeks by someone who was an expert in their graphic programming language. A much longer time should be allowed. This system requires knowing ModSAF very well because it will make use of all of ModSAF and may require extending ModSAF.



**Figure 6.1: On the left, a functional description of Tac-Air Soar and how it uses ModSAF. On the right, a perceptual interface to ModSAF.**

### 6.2.6 A Framework for Integrating Knowledge

Currently, there are multiple knowledge sets (as models) in different simulations that exist in multiple formats. It would be useful to create a framework for integrating multiple knowledge sets, allowing the knowledge to be reused in different simulations.

One way to create a framework for integrating knowledge is to create a task editor that could take a knowledge set and compile it for different architectures. The editor would have to be based on a high-level description of knowledge, such as generic tasks (Wielinga, Schreiber, & Breuker, 1992). These generic tasks would then be compiled into things such as an ACT-R or Soar rule-set.

There are potentially huge payoffs from this very high-risk project. First, this project would provide a way to reuse knowledge in multiple simulations. Second, the reuse that would arise would help validate models and might provide a way forward for validating architectures. Third, this project would provide another way of documenting behavior models. The (presumably) graphic representation would allow others to browse and understand the model on a high level. Fourth, it would assist in writing models. In most cases, there are a lot of low-level details in creating these models that are not of theoretical interest but require attention, such as using the same attribute name consistently (recent Soar interfaces now support this). A high-level compiler for knowledge like this would bring with it all the advantages traditionally associated with high-level languages. When done for Soar, the higher-level language allowed models to be built two to three times faster (Yost, 1992, 1993).

PC-Pack ([www.epistemics.co.uk](http://www.epistemics.co.uk)) is a potential tool to start building upon. Implementing an initial, demonstration version of this approach would take a good programmer 6 to 12 months. Putting it to use would take longer.

### 6.2.7 Methods for Comparing Modeling Approaches

We find ourselves in a position where a number of different approaches to simulating human behavior are available. Some of these approaches, at least, are based on datasets close enough to see themselves as rivals, and make competing claims about their suitability and quality. How can we assess and compare them?

There can, of course, be no one method that answers such a question. Earlier chapters of this report have discussed how practical considerations such as usability and communicability of models come into play as well as scientific qualities such as agreement with data. Thus, a wide range of comments about a model or architecture can be relevant to choosing between them.

However, there are some methods available that are too loose and varied to constitute a “technique” but are useful nonetheless for comparing and contrasting such differing approaches. They take the form of *matrix exercises*, in which a range of modeling approaches are pitted against a battery of concrete scenarios to be modeled. Young and Barnard (1987) provide the basic rationale for such a method and explain how it can be used to judge the fit and scope of a modeling approach. They argue, first, that the modeling approaches need to be applied to concrete scenarios. It is not sufficient to try comparing approaches on the basis of their “features” or “characteristics.” Second, it is important to use

a *range* of scenarios. Taking just a single case will inevitably introduce a bias towards or against certain approaches, and will fail to provide an indication of their scope. Young, Barnard, Simon, and Whittington (1989a) provide a short example of such a matrix exercise, and show how the entries in the matrix can be interpreted.

This kind of matrix exercise derives from the idea of a “bake-off” between rival approaches but also differs in important respects. There is unlikely to be a “winner,” one approach that is regarded as the best in all respects. Moreover, the matrix exercise is fundamentally cooperative rather than competitive. Instead of finding the “best” approach, bake-offs provide a tool for probing the scope of applicability of the different approaches, and investigating their relative strengths and weaknesses, advantages and disadvantages, for later modification and fusion. Pew and Mavor appear to call for this kind of activity (1998, pp. 336-339) as well.

Some exercises of this kind have been performed in public. At the Research Symposia associated with the CHI conferences in 1993 and 1994, Young (in 1993) and Young and C. Lewis (in 1994) organized such matrix exercises on the design of an *undo* facility for a shared editor (1993), and on the analysis of the persistent *unselected window* error and of the design of an automated bank-teller machine as a walk-up-and-use device (in 1994). Furthermore, there are precedents for such an exercise in a military research context. In 1993, NASA funded a comparative study of models of pilot checklist completion. The Office of Naval Research has funded, on a longer time scale, multiple analyses and modeling of several interactive tasks using hybrid architectures (Gigley & Chipman, 1999). The speech recognition community in the United States uses this approach in a quite competitive way as well.

The U.S. Air Force has recently started a similar program called Agent-Based Modeling and Behavior Representation (AMBR) to explore models of complex behavior ([www.williams.af.mil/html/ambr.html](http://www.williams.af.mil/html/ambr.html)). This multi-team project comparing four cognitive architectures was recently reported at the 2001 Computer Generated Forces Conference. For an overview, see Gluck and Pew (2001a; 2001b); Tenney and Spector (2001) provide a summary of the model to data fits in the most recent comparison round. Several more iterations of comparisons across architectures using different types of tasks are planned.

A final but important point about such an exercise is that it cannot be done successfully inexpensively. The exercise requires earmarked and realistic funding to provide useful results. A considerable amount of work is required: first in negotiating, agreeing, and then specifying a set of concrete and clearly described scenarios, ideally with associated empirical data; and then subsequently for applying the modeling approaches to the scenarios, performing the comparisons, and drawing conclusions. Multiple research groups are used, and the funding has been leveraged by the groups’ existing work and multiple funding sources.

## 6.2.8 (Re)Implementing the Battlefield Simulation

There are strong arguments for implementing communicating agents and intra-agent processes in JAVA. These are discussed in Bigus and Bigus (1997), and in the context of JACK Intelligent Agents by Busetta et al. (1995b). In fact, there are powerful arguments for building the entire synthetic agent simulation in JAVA as described below. This is possible

within the Higher-Level Architecture (HLA) framework. Implementing Soar in JAVA has also been mooted (Schwamb, 1998), as well as ACT-R (see [www.jactr.sourceforge.net](http://www.jactr.sourceforge.net) for information on a preliminary JAVA implementation of ACT-R 4, as of May 2001), although the usability of these architectures would suffer for this.

A core system implementation is needed that can then be accessed through Application Programming Interfaces (APIs). Supporting software is available for this, but any software could be developed for the purpose, provided it conformed with the standard. The core system might be written in JAVA or any other language provided only that an API is implemented. Similarly, entities may be written in any language, or several, provided that they set up calls to the API specification. There are a number of arguments for using JAVA as the basis for both individual entity simulation and for building a core system to the HLA specification. These are described below.

The single most attractive advantage of developing a synthetic battlefield simulation within a JAVA environment lies in the capabilities available within a Remote Method Invocation (RMI) that forms part of the JAVA run-time environment. This is a distributed object model with some similarities to Microsoft's Distributed Component Model (DCOM)<sup>®</sup> but with the advantage that it is effective on any platform that supports a JAVA run-time environment. It goes well beyond traditional remote procedure calls being entirely object-based, even allowing objects to be passed as parameters. Object behavior as well as data can be passed to a remote object in a seamless and transparent way. A mortar weapon being passed as an argument to an individual infantry man entity and arriving complete with its complement of munitions and ability to be fired gives a picture of this capability. The JAVA run-time environment also supports a naming and directory service API (JAVA JNDI) that allows the objects of RMI calls to be found. (For more details of this see [www.javasoft.com/products/jndi/index.html](http://www.javasoft.com/products/jndi/index.html).)

To show how such a service might be used, suppose that a simulation of an individual paratrooper has been developed. This simulation is a uniquely named JAVA object that can be invoked on any machine on the network used for the simulation. The JAVA Naming and Directory Interface (JNDI) service will inform a process about which machines have a suitable simulation available. To take this an important stage further, we use a class-factory object to produce the individual paratrooper objects. This class factory might use randomized parameters to make each entity distinct but fitting a known distribution (like Cabbage-Patch Dolls<sup>®</sup>). To introduce these entities into the simulation, a process would ask the naming service for a suitable class-factory object. This might be on one of any number of machines and is therefore extremely robust against damage to the network. The class factory can then be asked to produce any number of paratrooper entities, each of which (in JAVA) is capable of serializing itself to any other machine on the network, and running there. Indeed, the simulation can be moved from machine to machine at will, perhaps in response to a condition such as imminent power failure.

This approach would also support testing new platforms. A manufacturer might develop an improved simulation of a Tornado fighter-bomber. They then could introduce a new machine with a suitably registered class-factory object. Once this was connected to the network, the new simulation would be immediately available even if this were done while a simulation was running. No relinking, recompilation, or even pause in the simulation would be needed. The objects could be defined in conformity with the HLA standard.

JAVA also supports secure communications and has well-developed APIs for database connectivity and for driving graphics devices. An attractive user interface is very much easier to develop using the JAVA Foundation Classes (JFC) than, for example, using X Motif. In addition, if a Just In Time (JIT) compiler is available to the RTE, programs developed in JAVA show little performance degradation in comparison with C++.

A synthetic environment could be developed using facilities offered by the JAVA run-time environment and existing APIs that would come much closer than existing simulations in meeting the design goals of maintainability, versatility, and robustness. This approach would have to be agreed upon by multiple communities and requires a large amount of resources to be applied uniformly.

### 6.3 Projects Improving Usability

The projects presented here roughly address the issues raised in Chapter 4. This section reviews several possible projects for making model building more routine. For practical reasons, it is useful to make the model-building process more routine. It is also important for theoretical reasons. If the models cannot be created within a time commensurate with gathering data, the majority of the work will continue to be data gathering because theory development will be seen as too difficult.

#### 6.3.1 Defining the Modeling Methodology

There is not yet a definitive approach or handbook for building models that can also be used for teaching and practicing modeling cognitive behavior. Newell and Simon's (1972) book is too long and mostly teaches by example. Ericsson and Simon's (1993) book on verbal protocol analysis has comments on how to create models; although useful, the comments are short. VanSomeren, Barnard, and Sandberg (1994) provide a useful text, although it is slightly short and some of the details of going from model to data are not specified (if indeed they can be). Baxter's (1997) report and Yost and Newell's (1989) article are useful examples of the process, but both are tied to a single architecture and not widely available. There are other useful papers worth noting, but they are short and not comprehensive (e.g., Kieras, 1985; Ritter & Larkin, 1994; Sun & Ling, 1998).

Rouse (1980) has also made an attempt at describing the modeling process. He identifies the following steps as forming an important part of the modeling process: (1) definition, (2) representation, (3) calculation, (4) experimentation, (5) comparison, and (6) iteration. Rouse mainly focuses on the representation and calculation aspects of modeling, particularly from an engineering point of view. He describes several methodologies, including control theory, queuing theory, and rule-based production systems. He also provides a short tutorial on several of these modeling methods together with practical examples of systems engineering models. The examples are taken from a wide variety of domains including aviation, air traffic control, and industrial process control. It is not a complete treatise on human behavior, but does provide suggestions for methods that may be useful in modeling certain aspects of human behavior.

Similar tutorials and methodological summaries should be created until they converge. The results will be useful to practitioners and those learning to model; the latter will be an

important audience as this field grows. The output is most likely to require a textbook. A year to several years of support would significantly help create this set of learning materials.

### 6.3.2 Individual Data Modeling: An Approach for Validating Models

What is the best way to make theoretical progress in the study of behavior? Is it to develop micro-theories that explain a small domain or to aim at a higher goal, and develop an overarching theory covering a large number of domains—a unified theory? Modern psychology, as a field, has tended to prefer micro-theories. Unified theories have regularly appeared in psychology—think of Piaget’s (1954) or Skinner’s (1957) theories—but it is generally admitted that such unified theories have failed to offer a rigorous and testable picture of the human mind. Given this relatively unsuccessful history, it was with interest that cognitive science observed Newell’s (1990; see also Newell, 1973) call for a revival of unified theories in psychology.

One of the reasons for the limited success of Newell’s own brand of UTC is that the methodology commonly used in psychology, based on controlling potentially confounding variables by using group data, is not the best way forward for developing UTCs. Instead, Gobet and Ritter (2000) propose an approach, which they call Individual Data Modeling (IDM), where (1) the problems related to group averages are alleviated by analyzing subjects individually on a large set of tasks, (2) there is a close interaction between theory building and experimentation, and (3) computer technology is used to routinely test versions of the theory on a wide range of data. They claim that there are significant advantages here, that this approach will also help traditional approaches progress, and that the main potential disadvantage—lack of generality—may be taken care of by adequate testing procedures.

IDM offers several particular advantages in this area. It does not require as much data because the data will not be averaged but compared on a fine-grained level. Not requiring a large amount of data is attractive when the data are detailed or expensive to acquire, or where the model makes detailed predictions. The other advantage is that it provides a model that produces more accurate behavior on a detailed level. It is this detailed level of behavior that will be necessary to not only allow a model to appear human in a Turing test, but also lead to accurate training results because it performs like a comparable colleague or foe.

Work using IDM is ongoing at the University of Nottingham and at Pennsylvania State University. A full test would require one to two years of work to gather data and compare it with a model. Developing the IDM methodology and applying it could be combined with other projects, however, because it is a methodology and not a feature of behavior to include in a model.

### 6.3.3 Using Genetic Algorithms to Fit Data

There are two potential uses of genetic algorithms worth highlighting. The first is for generating behavior as described above in Section 5.2.1. The second is for optimizing model-fits by adjusting their parameters (Ritter, 1991). Most model-fits have been optimized by hand, which leads to absolute and relative performance problems. In absolute terms, researchers may not be getting optimal performance from their models. In relative terms, comparisons of hand-optimized models may not be fair. (Sometimes even one model

is optimized and the other not.) In the case of models with multiple parameters (with submodels to include), this job is not tractable by hand.

The results obtained by optimizing models with genetic algorithms suggest that optimizations done by hand are likely to be inferior to those done by genetic algorithms (Ritter, 1991) or by other machine-learning techniques (Butler, 2000). Use of genetic algorithms (or similar techniques) would improve performance in absolute terms, provide fairer comparisons between models, and encourage the inclusion of parameter set behavior in model comparisons. Several years of a PhD student working within a project with a model to optimize is probably a good way to progress work in this area.

This optimization should initially be done with an existing model so that the developers of the interface have a ready-made model and audience. The basic approach is simple and robust, and should be straightforward to demonstrate. Making the optimization routine and portable are separate and more advanced steps, so this project could take almost any amount of resources, ranging from a month to several years.

### **6.3.4 Environments for Model Building and Reuse**

There remains a need for better environments for creating models. Few modeling interfaces provide much support for the user to program at the problem-space level or even the knowledge level, although the COGENT interface is interesting as an example of usability.

Soar, in particular, needs a better interface. While there is now a modest interface, even the latest versions of the Soar interface (Kalus & Hirst, 1999; Laird, 1999; Ritter et al., 1998b) are not as advanced as many expert system shells and are just becoming as comprehensive as the previous, Lisp-based version (Ritter & Larkin, 1994). The Soar interface is, however, providing increasing amounts of support at the symbol level (Jones, Bauman, & Laird, 2001; Roytam, 2001) and higher, including model-specific displays (Jones, 1999b). TAQL (Yost & Newell, 1989) and Able (Ritter et al., 1998b) have been moderately successful, but modest attempts to create high-level tools in Soar, for example, Gratch's (1998) planning-level interface should be expanded and disseminated as a modeling interface. Knowledge acquisition tools and techniques (e.g., Cottam & Shadbolt, 1998; O'Hara & Shadbolt, 1998) might be particularly useful bases upon which to build.

Associated with this project would be general support for programming. This includes lists of frequently asked questions, tutorials, and generating models or model libraries designed for reuse. These libraries should either exist in each architecture or in the general task language developed in the previous task. These would serve as a type of default knowledge for use in other applications. We can already envision libraries of interaction knowledge (about how to push buttons and search menus), arithmetic, and simple optimization like the default knowledge in Soar.

Work on improving the modeling interfaces for each architecture should be incorporated as part of another modeling project so that the developers of the interface have a ready-made audience. There are multiple additions that would be useful and multiple approaches that could be explored, so this project could take almost any amount of resources, ranging from a month to several years.

### 6.3.5 Automatic Model Building

Most process models induced from protocols are created by hand. There has been some work to do this automatically or semi-automatically with machine-learning techniques. Semi-automatic model generation is done in the event-structure modeling domain (a sociological level of social events) by a program called Ethno (Heise, 1989; Heise & Lewis, 1991). Ethno iterates through a database of known events finding those without known precursors. It presents these to the modeler, querying for their precursors. As it runs it asks the modeler to create simple qualitative, non-variabilized token-matching rules representing the event's causal relationships based on social and scientific processes. The result at the end of an analysis is a set of 10 to 20 rules that shape sociological behavior in that area. In a sense, the modeler is doing impasse-driven programming (i.e., what is the next precursor for an uncovered event not provided by an already existing rule?). After this step, or in place of it, the modeler can compare the model's predictions with a series of actions on a sociological level (a protocol in the formal sense of the word). The tool notes which actions could follow and queries the modeler based on these. Where mismatches occur, Ethno can present several possible fixes for configuration. Incorporating the model with the analysis tool in an integrated environment makes it more powerful. It would be a short extension to see the social events as cognitive events in a protocol.

Stronger methods for building models from a protocol are also available. Cirrus (VanLehn & Garlick, 1987) and ACM (Langley & Ohlsson, 1984) will induce decision trees for transitions between states that could be turned into production rules given a description of the problem space, including its elements and the coded actions in the protocol. Cirrus and ACM use a variant of the ID3 learning algorithm (Quinlan, 1983). (ID3 induces rules that describe relationships in data.)

These tools look like a useful way to refine process models. Why is automatic creation of process models not done more often? Perhaps it is because these tools do not create complete process models. They take a generalized version of an operator that must be specified as part of a process model. It could also be that finding the conditions of operators is not the difficult problem but that creating the initial process model and operators is. It could also be that it is harder to write process models that can be used by these machine learning algorithms. In any case, these methods should be explored further.

Diligent (Angros, 1998), Instructo-Soar (Huffman & Laird, 1995), and Observe-Soar (van Lent, 1999) are approaches to create models in Soar that learn how to perform new tasks by observing behavior and inferring problem-solving steps to duplicate them. Related models have been used in synthetic environments (Assanie & Laird, 1999; van Lent & Laird, 1999). They have had limited use but suggest that learning through observation may be a way to create models as it is an important way that humans learn. Their lack of use could simply be due to the fact that they are novel software systems. As novel systems they are probably difficult for people other than their developers to use and will have to go through several iterations of improvement (like most pieces of software) before they are ready for outsiders. With a small user base (so far), the need has not forced software development, which has further decreased their potential audience.

Automatic modeling tools need to be developed. Machine-learning algorithms and theories of cognition are developed enough that this could be a very fruitful approach. A several-year effort here could yield large benefits of more routine modeling.

### **6.3.6 Improvements to ModSAF**

A major problem with ModSAF is usability. ModSAF is large and has a complicated syntax. Users report problems learning and using it. One way to improve its usability might be a better interface; better manuals and training aids might also be useful.

The approach used by models of behavior to interact with basic simulation capabilities such as ModSAF needs to be regularized. A fundamentally better approach might be possible. There exists an interface between ModSAF and Soar that partly provides a model eye and hand. This eye/hand could be improved to provide a more abstract interface to ModSAF, one that might be easier to use (Schwamb et al., 1994).

One thing we have repeatedly noted is that getting models to interact with simulations is more bearable when both are implemented within the same development environment. When they are not, work proceeds much more slowly (Ritter et al., 2000; Ritter & Major, 1995), requiring a mastery of both environments. The situation is exacerbated because the development and use of any communication facility tends to be an ill-defined problem with numerous wild subproblems (i.e., problems where the time to solution can be high and with a large variance, that is, not easily predicted). So, for example, although the ModSAF Tac-Air system (Tambe, Johnson, Jones, Koss, Laird, Rosenbloom, et al., 1995) appears as if it was developed using joint compilation techniques, it was probably difficult to use because it implements communication between ModSAF and the Tac-Air model using sockets. Although informal communication with researchers in the Soar and robotics communities suggest that the use of sockets may be becoming more routine, this has not always been the case.

## **6.4 Other Applications of Behavioral Models in Synthetic Environments**

There are numerous ways that behavioral models could be applied outside the military domain. We will examine four of them here.

The most obvious additional application of the models arising from approaches proposed in this report is in the provision of automated support for system operators. This support can take the form of intelligent decision-support systems or embedded assistants that guide operator behavior. There are some existing applications, most notably the Pilot's Associate (Geddes, 1989), its derivative, Hazard Monitor (Greenberg, Small, Zenyah, & Skidmore, 1995), and CASSY (Wittig & Onken, 1992), all from the aviation domain. In the United Kingdom, the Future Organic Airborne early warning system is attempting to insert a knowledge-based system into the Osprey aircraft and radar simulation to assist users.

These assistants, because they have a model of what the user is likely to do next, should be able to assist the user: if not by performing the task, then by preparing materials or information, or by modifying the display to help distinguish between alternatives or make performing actions easier. In the past, such assistants have had only a limited ability to model users. With increased validity and accuracy, these models may become truly useful.

The second application is in education and training. The uses in education have been fairly well illustrated by Anderson's work with cognitive model-based tutors (Anderson, Corbett, Koedinger, & Pelletier, 1995). In training, behavioral models can be used to provide experts to emulate and the same knowledge can also be used to debrief students' performances (Ritter & Feurzeig, 1988). The knowledge can also be used to populate adversaries and colleagues in the same environment (Bloedorn & Downes-Martin, 1985).

Training needs exist outside the military in several domains where dynamic models are necessary. Mining, for example, is starting to use virtual reality to train simple tasks (Hollands, Denby, & Brooks, 1999). Virtual reality is already being used to train hazard-spotting, avoiding mine machinery as a pedestrian, and driving vehicles underground (Schofield & Denby, 1995). A web search on virtual reality and training will indicate a wide range of other areas of application as well.

The third application is in entertainment. This has been proposed for some time as an application. A recent report by the U.S. National Research Council (Computer Science and Telecommunications Board, 1997) suggests that it is possible to use synthetic environments and the behavioral models in them for entertainment. This is currently being done by the Institute for Creative Technologies at the University of Southern California.

The fourth application is in systems analysis. The behavioral models can be used to examine different system designs to measure errors, processing rates, or emergent strategies. To return to mining again, truck models in a simulation can be used to examine road layouts in mines (Williams, Schofield, & Denby, 1998).

## 6.5 Summary of Projects

We have laid out important objectives for models of behavior in synthetic environments in the important areas of providing more complete performance, increased integration of the models with each other and with synthetic environments, and improved usability of the models. A wide range of funding bodies may be interested in supporting these projects because most of these projects have both engineering and scientific results. They will not only improve engineering models of human behavior, but they will also improve our understanding of behavior and our general scientific ability to predict and model human behavior generally.

These proposals, taken as a whole, call for several broad and general research programs. They suggest several moderating variables that affect cognition, including emotions and behavioral moderators, personality, and interactions with the environment, which should be included in cognitive architectures. They argue for creating or moving towards a more uniform format for data and models and a more clearly defined approach for modeling. There are also several concrete suggestions for making modeling easier and more routine, including providing more usable modeling environments and supporting automatic model generation. Finally, we were able to suggest some further applications of models of behavior in synthetic environments.

