KNOWLEDGE-BASED SIMULATION

Philip Klahr and William S. Faught
The Rand Corporation
Santa Monica, California 90406

ABSTRACT

Knowledge engineering has been successfully applied in many domains to create knowledge-based "expert" systems. We have applied this technology to the area of large-scale simulation and have implemented ROSS, a Rule-Oriented Simulation System, that simulates military air battles. Alternative decision-making behaviors have been extracted from experts and encoded as object-oriented rules. Browsing of the knowledge and explanation of events occur at various levels of abstraction.

I. INTRODUCTION

Large-scale simulators have been plagued with problems of intelligibility (hidden embedded assumptions, limited descriptive power), modifiability (behaviors and rules buried in code), credibility (minimal explanation facilities), and speed (slow to build, to run, to interpret). The area of large-scale simulation provides a rich environment for the application and development of artificial intelligence techniques, as well as for the discovery of new ones.

The field of knowledge engineering [2] is developing tools for use in building intelligent knowledge-based expert systems. A human expert communicates his expertise about a particular domain in terms of simple English-like IF-THEN rules which are then incorporated into a computer-based expert system. The rules are understandable, modifiable, and self-documenting. Knowledge-based systems provide explanation facilities, efficient knowledge structuring and sharing, and interfaces that are amiable for system building and knowledge refinement.

Our approach to simulation views a decision-based simulator as a knowledge-based system. The behaviors and interactions of objects, the decision-making rules, the communiciation channels are all pieces of knowledge that can be made explicit, understandable, modifiable, and can be used to explain simulation results.

For our work in simulation, we chose the domain of military air battles. Current large-scale simulators in this domain exhibit exactly the simulation problems we discussed above and thus provide a good area in which to demonstrate the feasibility and potential of knowledge-based simulation.

II. KNOWLEDGE REPRESENTATION

Our domain experts typically centered their discussions of military knowledge around the domain objects. For example, a particular type of aircraft has certain attributes associated with it such as maximum velocity, altitude ranges, time needed to refuel, etc. Similarly, individual planes have given positions, speeds, altitudes, routes, etc. In addition, experts defined the behaviors of objects relative to object types or catagories. For example, they defined what actions aircraft take when they enter radar coverages, what ground radars do when they detect new aircraft (who they notify, what they communicate), etc. It became clear that an object-oriented (Simula-like [1]) programming language would provide a natural environment in which to encode such descriptions.

We chose Director [6] for our initial programming language. Director is an object-oriented message-passing system that has been used primarily for computer graphics and animation. It offered considerable promise for our use in simulation, both in how knowledge is structured and how it is executed. Each object (class or individual) has its own data base containing its properties and behaviors. Objects are defined and organized hierarchically, allowing knowledge to be inherited, i.e., offsprings of objects automatically assume (unless otherwise modified) the properties and behaviors of their parents.

In Director, as in other message-passing systems (e.g., Smalltalk [3] and Plasma [5]), objects communicate with each other by sending messages. The Director format for defining behaviors is

```
(ask <object> do when receiving <message-pattern>
                        <actions>),
```

i.e., when the object receives a message matching the pattern, it performs the associated actions. In ROSS, we have added the capability of specifying IF-THEN rules of the form

```
(IF <conditions> THEN <actions> ELSE <actions>)
```

as part of an object's behavior. The conditions typically test for values (numeric, boolean) of data items while actions change data items or send messages to objects. Since Director is written in Maclisp, one may insert any Lisp s-expression as a condition or action. The following behavioral rule contains all of these options:

```
(ask RADAR do when receiving (IN RADAR RANGE ?AC)
(SCRIPT
(IF
 (lessp
  (length (ask MYSELF recall your OBJECTS-IN-RANGE))
  (ask MYSELF recall your CAPACITY))
THEN
 (ask HISTORIAN at ,STIME MYSELF detects ,AC)
 (ask MYSELF add ,AC to your list of
        OBJECTS-IN-RANGE)
 (ask ,(ask MYSELF recall your SUPERIOR)
        MYSELF detects ,AC)
 (ask MYSELF monitor ,AC while in coverage)
ELSE
 (ask HISTORIAN at ,STIME MYSELF doesn't detect ,AC)
)))
```

This rule is activated when a radar receives a message that an aircraft (AC) is in its radar range. The radar tests whether the number of objects currently in its radar coverage is less that its capacity. If its capacity is not full, then the radar tells the historian that it detects the aircraft at time STIME (the current simulation time), it records the aircraft in its log, it notifies its superior that it detected the aircraft, and it continues to monitor the aircraft through its radar coverage.

III. BEHAVIORAL DESCRIPTIONS

A difficult problem with large-scale simulators is that they contain complex behaviors that are hard to describe and understand. Each object has many potential actions it can take, and there may be hundreds of objects whose behavior the user may wish to examine and summarize at differing levels of generality and detail. To alleviate this problem, we organized the simulator's behavioral descriptions along two lines:

1. Static descriptions: descriptions of the major events simulated and the rules governing each object's potential actions. The events and rules are organized so that users can quickly peruse and understand the knowledge.

2. Dynamic descriptions: descriptions of each object's behavior as the simulator runs. Events are organized so the user is not overwhelmed by a mass of event reports. Events are reported as patterns with detail eliminated and selected events highlighted.

To organize the descriptions, we constructed scenarios representing sequential event chains. Each major event in ROSS has an associated "event descriptor" (ED). EDs are collected into chains, where each chain is a linear list of EDs. Each ED is causally associated with its immediate neighbors in the list: a preceding ED is necessary to cause its successor in that chain, but not necessarily sufficient. ([7] discusses the use of such chains in constructing proofs.)

ED chains are further organized into "activities," e.g., radar detection of an aircraft. Each activity is a tree of EDs. The root of the tree corresponds to the event that starts the activity. Each path from the root to a leaf is an ED chain. Logically, each ED chain corresponds to one possible scenario of events that could occur in a simulation. The scenario structure is used for both static and dynamic behavior descriptions.

A. BROWSING KNOWLEDGE

Static behavior descriptions are given by ROSS's "browse" function, an on-line interactive facility with which users can examine ROSS's knowledge base. The user is initially given a list of all activities. He then selects an activity, and the browse function prints a list of the names of all EDs in that activity. The user can ask for a more complete description, in which case the browse function prints a tree of the EDs. The user can then select a particular ED to examine, and the browse function prints a simplified description of the event. If the user asks for a more complete description, the browse function prints the actual code for the corresponding behavior (as in the example above). At any point the user can select the next ED, or go up or down activity/event levels.

The composition of ED chains, i.e., which EDs are members of which chains, is selected by the system developers for "naturalness" or appeal to a user's intuitive structure of the simulator. The system itself constructs the actual ED chains from the simulator's source code and a list of start and end points for each chain. The facility appears to be quite useful in our domain where the objects have interdependent yet self-determined behavior.

B. EVENT REPORTING

ROSS contains several objects that have been defined to organize, select, and report events to users. The Historian receives event reports from other objects (as exemplified in the behavioral rule above) and, upon request, supplies a history of any particular object, i.e., a list of events involving the object up to the current simulation time. The Historian also sends event reports to the Reporter, who selectively reports events to the user on his terminal as the simulator is running. The user can request to see all reports or only those involving a particular object or objects of a particular class. In addition, a Statistician accumulates statistics about particular fixed events (e.g., the total number of radar detections).

(We have also interfaced ROSS to a color graphics system which visually displays simulation runs. The graphics facility has been an indispensable tool for understanding ROSS and the simulations it produces and for debugging.)

C. EXPLANATION USING SCENARIOS

To explain "why" certain results occur, traditional rule-based systems typically show the rules that were used, one by one, backchaining from the conclusions. We have taken a different approach to explanation in ROSS. Rather than displaying individual rules to explain events, ROSS presents higher-level behavioral descriptions of what happened, akin to the activity and event descriptions used for browsing.

It is often the case that an event occurring in a simulaton run can be explained by specifying the chain of events leading up to the current event. This is accomplished simply by comparing event histories (gathered by the Historian) to the ED trees described above. The user can then browse the events and activities specified to obtain the applicable rules.

What is perhaps more interesting in simulation is explaining why certain events do _not_ occur. Often times simulations are run with expectations and the user is particularly interested in those cases where the expectations are violated. ([4] describes how such expectations can be used to drive a learning mechanism.) We have developed an initial capability for such "expectation-based" explanation.

Explanations are given relative to specified ED chains. One event chain is designated as the "expected" chain. An "analyzer" reports deviations from this chain, i.e., it determines the point (event) at which the ED chain no lcnger matches the simulation events. Typical responses from the analyzer are: aircraft in radar range but not detected; radar sent message to superior but it was not received; command center requested aircraft assignment but none available. Such analysis can occur at any time within a simulation run to determine the current status of expected event chains.

It is important to note that expectations need not be specifed prior to a simulation run (although this could focus the simulator's reporting activity). Users can analyze events with respect to any number of existing scenarios. Each analysis provides a simplified description and explanation of the simulator's operation from a different point of view (e.g., from radar's view, from aircraft's view, from a decision maker's view). This feature has also been extremely useful in debugging ROSS's knowledge base.

IV. SUMMARY AND FUTURE RESEARCH

ROSS currently embodies approximately 75 behavioral rules, 10 object types, and has been run with up to 250 individual objects. To show ROSS's flexibility, we have developed a set of alternative rule sets which encompass various strategies and tactics. Simulation runs using alternative rules show quite different behaviors and results.

Future research will include scaling ROSS up both in complexity and in numbers of objects. Our goal is to turn ROSS into a realistic, usable tool. A more user-oriented English-like rule-based language will be required for users to express behaviors and strategies. We are looking toward ROSIE [8], or a hybrid ROSIE object-oriented language for this purpose.

Scaling-up will necessitate enhancements in speed. We plan to explore parallel processing, abstraction (e.g., aggregating objects, adaptive precision), sampling, and focusing on user queries to avoid irrelevant processing.

In summary, we have applied knowledge engineering to large-scale simulation and implemented ROSS, an interactive knowledge-based system that simulates the interactions, communications, and decision-making behavior within the domain of military air battles and command and control. We have shown the feasibility and payoff of this approach and hope to apply it to other domains in the future.

ACKNOWLEDGMENTS

We wish to thank Ed Feigenbaum, Rick Hayes-Roth, Ken Kahn, Alan Kay, Doug Lenat, Gary Martins, Raj Reddy, and Stan Rosenschein for their helpful discussions and suggestions during ROSS's development. We thank our domain experts Walter Matyskiela and Carolyn Huber for their continuing help and patience, and William Giarla and Dan Gorlin for their work on graphics and event reporting.

REFERENCES

1. Dahl, O-J. and Nygaard, K. Simula -- an Algol-based simulation language. _Communications ACM_, 9 (9), 1966, 671-678.

2. Feigenbaum, E. A. The art of artificial intelligence: themes and case studies in knowledge engineering. _Proc. IJCAI-77_, MIT, 1977, 1014-1049.

3. Goldberg, A. and Kay, A. Smalltalk-72 Instruction Manual, SSL 76-6, Xerox Palo Alto Research Center, 1976.

4. Hayes-Roth, F., Klahr, P., and Mostow, D. J. Knowledge acquisition, knowledge programming, and knowledge refinement. R-2540-NSF, Rand Corporation, Santa Monica, 1980.

5. Hewitt, C. Viewing control structures as patterns of passing messages. _Artificial Intelligence_, 8 (3), 1977, 323-364.

6. Kahn, K. M. Director Guide. AI Memo 482B, Artificial Intelligence Lab, MIT, 1979.

7. Klahr, P. Planning techniques for rule selection in deductive question-answering. In _Pattern-Directed Inference Systems_, D. A. Waterman and F. Hayes-Roth (Eds.), Academic Press, New York, 1978, 223-239.

8. Waterman, D. A., Anderson, R. H., Hayes-Roth, F., Klahr, P., Martins, G., and Rosenschein, S. J. Design of a rule-oriented system for implementing expertise. N-1158-1-ARPA, Rand Corporation, Santa Monica, 1979.