Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

OBJECT-ORIENTED SIMULATION: WHERE DO WE GO FROM HERE?

Jeff Rothenberg
The Rand Corporation
1700 Main Street
Santa Monica, CA 90406, U.S.A.

1. OVERVIEW

Object-oriented simulation provides a rich and lucid paradigm for building computerized models of real-world phenomena. Its strength lies in its ability to represent objects and their behaviors and interactions in a cogent form that can be designed, evolved and comprehended by domain experts as well as system analysts. It allows encapsulating objects (to hide irrelevant details of their implementation) and viewing the behavior of a model at a meaningful level. It represents special relations among objects (class-subclass hierarchies) and provides "inheritance" of attributes and behaviors along with limited taxonomic inference over these relations. It represents interactions among objects by "messages" sent between them, which provides a natural way of modeling many interactions. Despite these achievements, however, there remain several largely unexplored areas of need, requiring advances in the power and flexibility of modeling, in the representation of knowledge, in the integration of different modeling paradigms, and in the comprehensibility, scalability and reusability of models.

The Knowledge-Based Simulation project at Rand is working in several of these areas. In this paper, we will elaborate the existing limitations of object-oriented simulation and discuss some of the ways we believe the paradigm can be extended to surmount these limitations.

2. BACKGROUND

Modeling is a way of dealing with things or situations that are too costly to deal with directly. Any model is characterized by three salient features:

1) It is *of* something (its "referent")

2) It has some intended *purpose* with respect to this referent

3) It is more *cost-effective* for this purpose than using the referent itself

The purpose of a model may be comprehension, prediction, communication, appreciation, etc. It must be more cost-effective to use a model for its given purpose than to use its referent, either because it is impossible to use the referent directly (e.g., modeling the birth of a star) or because it is safer, more convenient or cheaper (in some relevant coin) to use the model. The criteria of *purpose* and *cost-effectiveness for that purpose* determine the appropriate level of fidelity of a model: they define which features of the referent must be modeled (and with what accuracy) and which features can be ignored. The referent of a model must be well-defined; otherwise the criteria of reference, purpose and cost-effectiveness become arbitrary and meaningless. (Although it is not strictly necessary that the referent actually exist in order to be well-defined, it must nevertheless serve as "reality" from the perspective of the model.)

There are many ways of modeling a given thing or phenomenon; computer simulation forms a sub-class of these that includes analytic and discrete-state approaches. The analytic approach brings the power of mathematical analysis to bear on problems that can be understood or approximated analytically, but there remains a large class of problems that are not well enough understood to be handled this way. Such problems typically involve collections of interacting entities each of whose behaviors are understood reasonably well in isolation and whose low-level pairwise interactions with each other are known but whose high-level group interactions cannot be grasped. The strategy of discrete-state simulation is to encode these low-level interactions and "run" them in the hope that the overall behavior of the system will approximate that of its referent, and (perhaps) that higher-level interactions will reveal themselves. Such models are normally used for comprehension or prediction.

There are several requirements for the discrete-state approach to work. The real-world entities being modeled (along with their behaviors) must be represented comprehensibly: the correspondence between these entities and their representations in the model must be obvious, and the representations of their behaviors must be easily verifiable as faithful to reality. The low-level interactions among entities in the model must also be apparent: it must be clear which entities interact with each other, under what conditions they interact, and what their interactions are. Finally, the dynamics of the simulation must be comprehensible: it must facilitate perceiving and understanding patterns of high-level interaction and overall behavior of the collection of entities being modeled. The evolution of discrete-state simulation languages, from SIMSCRIPT (Kiviat 1968) and SIMULA (Dahl 1966) to the present, has attempted to satisfy these requirements with increasing success, leading to the current object-oriented paradigm.

Object-oriented simulation traces its roots to SIMULA. Its modern form is represented by such languages as SMALLTALK (Goldberg 1976), LOOPS (Stefik 1983), and ROSS (McArthur 1985). These languages provide many improvements over their predecessors, notably in the areas of lucid representation; however, they still have many shortcomings. The following sections discuss the major limitations of the current paradigm and possible ways of extending it.

3. LIMITATIONS AND NEW DIRECTIONS

The current object-oriented simulation paradigm has a number of serious limitations. Although these involve considerable overlap, they can be grouped roughly into the areas: Modeling Power, Control, Representation, Comprehensibility, and Model-Building.

3.1 Modeling Power

Discrete-state simulations are severely limited in the types of questions they can answer. Users typically specify initial states of the simulated world and then "run" the simulation to see what happens. This corresponds to asking questions of the form "what if?". As has been aptly pointed out in the literature (Davis 1982, Erickson 1985), there are many other kinds of questions that are of at least as much importance in many situations. These include "why?" questions ("why did Q happen?" or "why did object R take action S?"), "when?" questions ("under what conditions will T happen?"), "how?" questions ("how can result U be achieved?"), "ever/never" questions ("can X ever collide with Y?"), and optimization or goal-directed questions ("what is the highest value Z will ever reach?" or "what initial conditions will produce the highest value of Z?").

The inability of current discrete-state simulation systems to answer such questions derives from basic limitations in their representational and inferential capabilities. Representational limits include the difficulty of modeling goals, intentions, plans, beliefs and constraints (e.g., invariants). To the extent that these can be represented at all in most systems, they are often encoded implicitly in behaviors (i.e., specified procedurally), and are therefore not amenable to inference. For example, if an object's intentions are only implicit in the code it executes, it is theoretically difficult -- and practically impossible -- to answer questions about those intentions except by understanding the code. Answering such questions requires more powerful methods of representing knowledge.

If "inference" is defined in the most general terms as the ability to draw implicit information from explicit information, then running a traditional discrete-state simulation (or indeed any computational process) can be seen as one form of inference: an implicit conclusion (the computational result) is drawn from an explicit representation of the problem (the program and its data). The object-oriented paradigm also provides a degree of "taxonomic" inference based on its representation of class-subclass hierarchies: if x is a SmallTruck and SmallTruck is a subclass of Truck which is a subclass of MovingObject, then it can be inferred that x is a MovingObject. However, the range of conclusions that can be drawn by this means (the "inferential closure" of the class-subclass taxonomy under inheritance) is not very great. To answer the kinds of questions outlined above, it is necessary to supply a simulation system with a richer set of inference mechanisms which in turn access a richer set of knowledge representations.

The kinds of inference and explanation techniques that have been applied in expert systems define one axis of extending the capabilities of simulation systems. Such mechanisms (when coupled with appropriate representations of behaviors, constraints, etc.) allow the simulation to answer many of the questions posed above. In addition, however, it is important to avoid "discrete-state tunnel vision" which excludes analytic and optimization techniques. In cases where suitable analytic representations exist, it should be possible to integrate them into a discrete-state model, though current systems provide little or no support for such integration. As is pointed out in (Davis 1982) there are many cases where relationships can be communicated more effectively by analytic statements like "X increases when Y/Z decreases" than by showing computational results. Discrete-state techniques should be reserved for situations where such analytic insight is not available or is inappropriate for the intended application.

3.2 Control

In addition to the power of a system per se, there is the issue of how easy it is for a user to exercise that power. These issues are not entirely disjoint, since many control limitations imply underlying limitations of power or flexibility, as well. Control is also intimately linked to the interface through which it is exercised, and therefore involves issues of interface design.

One of the major shortcomings of most simulation systems is their inability to represent models with different degrees of aggregation. It is often unclear at the inception of a modeling effort what level of aggregation will be most appropriate or useful; the difficulty of changing this level after a simulation has already been implemented is often insurmountable. Beyond this, it is desirable to be able to build models whose degree of aggregation can be varied *dynamically* (either for efficiency or for conceptual purposes). This requires the ability to define objects that represent aggregates of other objects, where behaviors and interactions are allowed at various levels in the object hierarchy. Furthermore, this requires addressing the problem of insuring consistency among these levels so that (for example) a simulation can be run at an aggregated level up to a given point and then continued at a more detailed, disaggregated level in a meaningful way.

A related limitation is the difficulty of displaying results in usable form. Most object-oriented simulation environments provide some degree of graphic output to help visualize the workings of their simulations, but the user rarely has much control over the level of visual summarization or "abstraction". Abstraction is the presentational dual of aggregation, in that presenting the simulation in an abstracted form can make it appear (at least for some purposes) to be running at an aggregated level; however, here the underlying behavior of the simulation can be kept disaggregated, thereby avoiding consistency problems between the levels.

Ideally, a combination of dynamic aggregation and abstraction should provide the user with control over focus of interest in the model, so that as the user's attention is shifted or "zoomed" to different areas or levels of the model, the desired degree of detail is always available. This requires the ability to perform "localized simulation", where the objects of interest are simulated in full detail, but all other objects are simulated at an aggregated or abstracted level. To allow the user to "zoom" in on a simulation, it must automatically change its level of aggregation and abstraction to produce the required degree of detail as needed, giving rise to what we have called "fractal simulation".

The user of a simulation often needs to run it to a certain point and then try excursions, changing various aspects of the simulation, backing up to previous points and trying different paths. We have called this kind of control "intelligent exploration". This requires that the system save its state incrementally to allow backing up without recomputing; efficient techniques must be developed for doing this without inordinate overhead.

3.3 Representation

In addition to the need to represent goals, intentions, plans, beliefs, constraints and behaviors in ways that are amenable to inference (as noted above), there are a number of other serious representational limits that confront current systems. These have more to do with the ease of constructing simulations than with providing power to the user, but they are nonetheless crucial for creating more capable modeling environments.

The object-oriented paradigm focuses on the definitions of objects, each having various attributes and behaviors and communicating with other objects via messages. Objects are organized into one or more hierarchies which are conceived as representing class-subclass relations on the objects. This organization focuses exclusively on the class-subclass relation, with systems typically providing a built-in "inheritance" mechanism (which can be thought of as a special kind of inference) to support it. The class-subclass relation, however, is only one of many possible important relations. Others include part-whole (both physical and conceptual), proximity, connectivity, etc. The object paradigm must be extended to include a general notion of relation; relations themselves should be represented as having attributes and as participating in relations (i.e., relations among relations). Although supportive inference mechanisms (analogous to inheritance) may be defined for some of these types of relations (e.g., transitive closure for connectivity), the supporting mechanisms for a particular relation should be customizable by the simulation designer.

Most modeling efforts involve building a "model of interest" that is embedded in a surrounding "world model". The model of interest is the focus of concern for the modeler. The world model is the simulated environment of the model of interest: its presence is largely a distraction, but it must be modeled appropriately to provide the correct environment for the model of interest. A common example is the need to model physical phenomena (such as motion, signal propagation, etc.) to provide an environment for the objects of concern (e.g., airplanes or radio transmitters). To produce lucid simulations in such cases requires separating the world model from the model of interest within the simulation so that the details of the world model do not obscure the model of interest. Current simulation environments provide little or no support for such separation. Furthermore, implementing the world model often requires introducing "artifactual" objects (such as propagation channels) whose existence does not correspond to any real object of interest (Klahr 1985) and whose interactions clutter the simulation with artifactual messages. New techniques for encapsulating such objects and behaviors are a necessity for building lucid simulations.

At a more fundamental level, a strict object-oriented paradigm may be inappropriate for modeling certain aspects of the real world. In particular, representing physical and temporal phenomena (such as motion), and autonomous phenomena in general (those phenomena which change independently over time), does not fit the object/message framework very well. Since these phenomena form the core of the world models of a huge class of models of interest, it is vital to pursue alternative representations for them. The use of declarative analytic forms (i.e., mathematical relationships and constraints) must be explored as a way of removing this artifactual burden from the object paradigm.

An additional source of artifacts is the maintenance of the simulation environment itself. One such artifact involves the "automatic unplanning" of planned events which become invalidated by subsequent events. For example, an object's plan should be automatically negated if that object is destroyed. The integration of a graphic interface with a simulation also tends to introduce artifacts. Objects and behaviors that have nothing to do with the actual model may be needed to handle problems such as when to update the display, how to do so with minimal overhead and visual disruption, how to insure consistency of the simulation state at the moment of display (e.g., avoiding "ghost" images of objects in invalid states), and how to synchronize user input (for example, interrupting the simulation to query some object) with the state of the simulator (which may have changed the state since the last graphic update). Solutions to such problems often require such artifacts as synchronization messages, objects and attributes that represent the graphic state itself, and behaviors for objects to display themselves or to update their images. Whereas the object-oriented approach may make a graphic interface relatively easy to implement (Klahr 1985, Stefik86), it seems to do so at the cost of introducing artifacts which plague the designer and implementor, if not the user. Artifacts like these deserve special treatment via tailored mechanisms in the simulation environment.

3.4 Comprehensibility

Much of the advantage of the object-oriented paradigm over other approaches derives from its having improved the comprehensibility of simulations, both for the designer/implementor and for the ultimate user. This in turn is a direct result of what is by now a well-accepted tenet of good software engineering: the encapsulation of data and behavior. While this is decidedly worthwhile, it does not by itself eliminate artifacts from simulations (as discussed above), nor does it address a number of broader issues of comprehensibility by users.

The user of a modern simulation typically interacts with the system through a graphics interface. Although some of these interfaces are quite impressive, many of them allow minimal interaction. It is still difficult to provide a simulation builder with the tools to create highly interactive graphics interfaces without considerable effort. Along with "intelligent exploration", as described above, a simulation must provide "intelligent explanation" of its behavior and its results. The user should be able to query the states of visible objects, obtain graphical representations of relations among objects and use the display to explain the behavior of the simulation wherever appropriate. A graphics interface of this kind straddles the two capabilities we have called intelligent explanation and exploration: two-way graphic interaction is assumed in both cases (for example, there should be no sharp demarcation between querying an object's state and changing it). Intelligent explanation should take advantage of the

representational and inferential power of the simulator to attempt to display meaningful results to the user by means of all available graphic techniques, including animation, color, visual abstraction, etc.

In many cases a user may want to see different aspects or levels of a simulation at different times or for different purposes. The control of aggregation and abstraction discussed above provides one degree of freedom in this direction, but by no means the only one. It is often important for the user to be able to specify the desired scope, level or (generally) "view" of the simulation to be displayed, in terms of the structure of the underlying model. This might be done by dynamically creating an object hierarchy at the user's request to represent the desired view. Such a facility would allow showing different "cuts" of a simulation while it was running.

One final area of need along this axis is that of sensitivity analysis. Understanding how a model depends on variations in its parameters may provide vital insight. Unfortunately, in all but the simplest cases, sensitivity analysis has traditionally been prohibitively expensive, even in analytic simulations where closed-form solutions for partial derivatives are available; in discrete-state simulations where closed-form solutions are rarely available, the situation is usually considered a lost cause. Although reasonable techniques have been developed for displaying the results of sensitivity analysis (Clemons 1985), much remains to be done to make it computationally feasible. Nevertheless, without sensitivity analysis it is often nearly impossible to understand which relationships are meaningful in a complicated model. In Section 4.4, we will describe some techniques we are exploring to simplify the computations.

3.5 Model-Building Issues

The above discussion has touched on a number of issues related to the implementation of simulations (such as representational techniques for eliminating artifacts). However, there are additional problems facing model builders, ranging from software engineering to conceptual issues.

The desire to build ever larger and more realistic models of complex environments inevitably raises questions of scalability. Though the object-oriented approach itself can be found in languages like Ada (for which scalability was a major design goal), object-oriented simulation environments have tended to be implemented in LISP, and therefore suffer from LISP's well-known scaling and integration problems. The most obvious dimension of scalability involves efficiency and execution speed, which become problematic when scaling from dozens of objects to thousands. The solution to this may lie in parallel computing, though this involves problems of partitioning an object-oriented solution into subsets that can be effectively distributed among processors (Jefferson 1982).

A further subtle problem of scaling involves issues of "scope" in the design of an object-oriented simulation. Although objects are theoretically intended to encapsulate data and operations, most current environments make object names, message forms, and even attribute names globally available, thereby undermining one of the main advantages of object-orientation. In order to create complex simulations involving more complex object class hierarchies and interactions (not just larger numbers of instances of a small set of object classes), it is imperative to reduce this global name space: software engineering principles decree that the proliferation of globals is inimical to reliability. Some form of scoping of name spaces must be developed to permit such scaling. Ironically, the object paradigm itself seems to provide a natural construct for such scoping, i.e., the object. An object S might define a scope for its subordinate objects by prohibiting them from naming any other objects except each other (and S), and by allowing them to access as globals only the attributes of S. The use of objects to represent scope in this way should be explored as a possible solution to the problem of globals in object-oriented simulations.

As simulations become more complex and attempt to represent additional kinds of knowledge, they will inevitably encounter many of the same problems found in expert systems, having to do with maintaining a consistent knowledge base. Some of these problems can be solved by the use of declarative constraints to enforce invariants on the knowledge base. Expert systems research has developed techniques for "truth maintenance" (Quinlin 1982, Shafer 1985), and "logical support" (Garvey 1981) for assertions in a knowledge base, which should also be applicable to simulations. For example, this approach may offer a solution to the problem of "automatic unplanning" mentioned above: a plan that was unsupported by the current state would be automatically removed by a kind of "logical garbage collection". In addition, it has been noted (Davis 1982) that there tend to be many uncertainties about the assumptions in a model: some of the attributes, behaviors, interactions, and relationships in a simulation are more reliable than others. Ideally, the certainty and logical support of these aspects of the simulation itself should be handled by the same mechanisms that maintain the simulation knowledge base.

Finally, there is the dream of being able to build new models on top of old ones: using existing models, or pieces of models, as modules out of which to construct new ones. This is particularly applicable to those aspects of a model that constitute its "world model", as defined above. Any model that is embedded in a model of the physical world should ideally be able to build on top of existing models of the physical world, since these presumably share many of the same features. This assertion is somewhat glib, since different models may require modeling very different aspects or levels of detail of the physical world. However, if such world models can be built to allow variable levels of aggregation and abstraction (as discussed above), it should be possible to share them across a fairly wide range of applications. Of course, sharing ultimately requires a common language for knowledge representation and simulation behavior, which is not on the horizon. Nevertheless, it should be possible to represent significant portions of models (object descriptions, attributes, relations, etc.) in database form, suitable for use by any simulation environment capable of accessing the database. This capability, in conjunction with the principles stated above, should permit the construction of powerful, flexible models that are modular and lucid enough to be reusable. This would be a significant step forward both because it would reduce the effort of building realistic models and because it would lead to long-lived standard models of widespread interest (e.g., physical models), whose longevity would allow them to evolve, thereby enhancing their sophistication, reliability and completeness.

4. KNOWLEDGE-BASED SIMULATION AT RAND

The Knowledge-based Simulation project at Rand combines simulation and reasoning in an attempt to solve some of the deficiencies discussed above. It combines object-oriented simulation with expert systems techniques, emphasizing hybrid representation, simulation at multiple levels of abstraction, and graphic explanation and exploration. This section provides a brief summary of our current work. Much of the philosophy and strategy of our approach is contained in the foregoing discussion; the following comments serve mainly to identify which problems we are currently addressing.

4.1 Project Background

The seminal work of Newell, Shaw and Simon at Rand in the 1950s dominated much of AI's early research and defined many of its continuing focal points. In the last decade Rand research on expert systems has produced the RITA and ROSIE languages, as well as several expert system applications. Simulation research at Rand produced the SIMSCRIPT language as well as theoretical and experimental research in game theory, monte carlo simulation, and military wargaming.

More recently our simulation research has synthesized ideas and techniques from artificial intelligence, expert system technology, graphics, and distributed computing. Key results include the object-oriented simulation language ROSS that makes simulations easier to build and maintain, the Time Warp technique that reduces the execution time of object-oriented simulations by using a network of processors, and a number of simulations in ROSS (Klahr 1982, Klahr 1984, Klahr 1985).

4.2 Extended Modeling Paradigm (Hybrid Simulation)

We are currently augmenting the ROSS language in several ways. Objects will retain most of their current characteristics, including multiple hierarchies for the inheritance of attributes and behaviors. In previous ROSS simulations, generic ("class") objects do not themselves respond to messages; all messages are received and responded to by instance objects. We are experimenting with allowing simulations to vary their levels of abstraction and aggregation by associating behaviors and attributes with generic objects. These generic objects simulate the behavior of their instances and maintain attributes representing aggregations of the attributes of their instances.

We are exploring alternative rule paradigms to represent objects' behaviors, intentions and reasoning processes as well as the behavior of the simulation itself (e.g., selecting appropriate levels of abstraction and aggregation based on the user's stated needs). Our goal is to satisfy the dual criteria of making simulation code easier to understand and amenable to automatic inferencing.

In integrating these ideas, we are using rules and constraints to separate those aspects of a simulation that are really descriptions of the simulated world from the objects of interest in that world. Similarly, we are experimenting with declarative forms and "demand-driven" computation to separate out such aspects of a simulation as automatic unplanning, control of inferencing, control of level of aggregation, and "artifacts" of graphic presentation and interaction, thereby unburdening the model of interest.

4.3 Intelligent Explanation

Explanation requires that a simulation keep track of what it has done and be able to analyze its own execution history and behavior specifications, presenting this analysis to the user in understandable form. The system must maintain an execution history of events that have occurred, rules that have been invoked, messages that have been sent, prior values of attributes and states of databases, magnitudes of changes, etc. We are experimenting with various representations of simulation history for producing "execution trace" style explanation.

The primary task of explanation is to convince the user that a model is behaving reasonably, and to show how it arrived at a particular result. The user must be able to stop the simulation interactively and indicate (either graphically or by means of a query) the result that is to be explained. The user must also be able to back up to a previous point in the simulation, since a key result may not be recognized until after its occurrence.

We are developing a graphics facility for performing such interactions, emphasizing the ability to animate selected portions of a simulation. We plan to give control to the system (via rules) and/or the user (via direct interaction) over the level of graphic abstraction presented, so as to minimize visual clutter and display aggregated results.

4.4 Intelligent Exploration

Exploration allows a user to selectively modify a simulation, pursue excursions, focus attention on selected aspects of the model, perform sensitivity analysis or ask how particular results might be achieved.

The graphic interaction described above allows the user to select objects graphically and edit their attributes or behaviors explicitly. This provides a natural way to specify a scenario and set up initial conditions for a simulation run. We are also experimenting with graphical input for specifying procedural information, for example allowing the user to draw a route for a moving object on the screen. The system must capture (and generalize) the relevant information, representing it as a new behavior for the object. This "graphic behavior modification" should allow non-programmers to specify simulation scenarios more easily.

Similar techniques will allow the user to interrupt a simulation and try alternative excursions, selectively modifying attributes and behaviors. We are experimenting with alternative approaches to relaxing constraints during such explorations, under user control.

Another major concern is to allow analysts to perform sensitivity analysis on a model, to identify important factors. We are examining both static and dynamic approaches to this problem. Static approaches include applying inferencing to rules and logical constraint declarations, or applying analytic methods to closed-form mathematical constraints. Dynamic approaches include automatically generating and running excursions to perturb selected variables. In both static and dynamic cases we are experimenting with a hierarchical representation of sensitivity, as a way of improving the computational feasibility of complicated analyses.

Finally, we are examining the utility of goal-driven simulation, where the user specifies a hypothetical result and the system tries to find a way to achieve it by a combination of static analysis of constraints (to eliminate certain results as impossible), forward chaining from a given set of conditions, and backward chaining from desired goals or hypotheses.

5. CONCLUSION

Modeling is by far the most powerful technique for understanding and predicting the behavior of the real world. Discrete-state simulation is an important form of modeling that allows representation of systems whose behavior is not well understood analytically. The object-oriented paradigm represents the current state-of-the-art in discrete-state simulation; it has greatly enhanced the lucidity and cogency of this form of modeling. Nevertheless, many severe limitations must still be overcome, ranging from fundamental issues of modeling power and representation to pragmatic issues of comprehensibility, scalability and reusability. The above discussion has identified some of the most important limitations in each of these categories, and has suggested solutions in terms of extensions to the paradigm. A number of these extensions are currently being explored by the Knowledge-Based Simulation project at Rand.

REFERENCES

Clemons, E. and Greenfield, A. J. (1985). The SAGE System Architecture: A System for the Rapid Development of Graphics Interfaces for Decision Support. *IEEE CG&A*, 5(11), 38-50.

Dahl, O-J. and Nygaard, K. (1966). Simula--An Algol-Based Simulation Language. *Communications ACM*, 9, 671-678.

Davis, M., Rosenschein, S. and Shapiro, N. (1982). Prospects and Problems for A General Modeling Methodology. N-1801-RC, The Rand Corporation, Santa Monica, California.

Erickson, S. A. (1985). Fusing AI and Simulation in Military Modeling. *AI Applied to Simulation, Proceedings of the European Conference at the University of Ghent*, 140-150.

Garvey, T., Lowrance, J. and Fischler, M. (1981). An Inference Technique for Integrating Knowledge from Disparate Sources. *Proceedings of IJCAI '81*, 7.

Goldberg, A. and Kay, A. (1976). Smalltalk-72 Instruction Manual. Report SSL 76-6, Xerox PARC, Palo Alto, California.

Jefferson, D. and Sowizral, H. (1982). Fast Concurrent Simulation Using the Time Warp Mechanism, Part I: Local Control. N-1906-AF, The Rand Corporation, Santa Monica, California.

Kiviat, P., Vilanueva, R. and Markowitz, H. (1968). The Simscript II Programming Language. Prentice-Hall, Englewood Cliffs, New Jersey.

Klahr, P. (1985). Expressibility in ROSS: An Object-Oriented Simulation System. *AI Applied to Simulation, Proceedings of the European Conference at the University of Ghent*, 136-139.

Klahr, P., Ellis, J., Giarla, W., Narain, S., Cesar, E. and Turner, S. (1984). TWIRL: Tactical Warfare in the ROSS Language. R-3158-AF, The Rand Corporation, Santa Monica, California.

Klahr, P., McArthur, D., Narain, S. and Best, E. (1982). Swirl: Simulating Warfare in the ROSS Language. N-1885-AF, The Rand Corporation, Santa Monica, California.

McArthur, D. and Klahr, P. (1985). The ROSS Language Manual. N-1854-1-AF, The Rand Corporation, Santa Monica, California.

Quinlin, R. (1982). Inferno: A Cautious Approach to Uncertain Inference. N-1898-C, The Rand Corporation, Santa Monica, California.

Shafer, G. and Tversky, A. (1985). Languages and Designs for Probability Judgement. *Cognitive Science*, 9, 309-339.

Steeb, R., Cammarata, S., Narain, S. and Giarla, W. (1984). Distributed Problem Solving for Air Fleet Control: Framework and Implementations. N-2139-ARPA, The Rand Corporation, Santa Monica, California.

Stefik, M., Bobrow, D. and Kahn, K. (1986). Integrating Access-Oriented Programming into a Multi-Paradigm Environment. *EEE Software*, 10-18.

Stefik, M., Bobrow, D. G. and Mittal, S. (1983). Knowledge Programming in LOOPS: Report on an Experimental Course. *The AI Magazine*, 3-13.

AUTHOR'S BIOGRAPHY

Jeff Rothenberg is a Computer Scientist and Project Leader at the Rand Corporation, in charge of the Knowledge-Based Simulation Project. He received a B.A. in mathematics from Williams College in 1968. At the University of Wisconsin, Madison, he received an M.S. in Computer Science in 1969, and then pursued additional graduate work in AI and robotics. He has been involved in various simulation, graphics, and intelligent tutoring applications at USC Information Sciences Institute (1973-79), Clear Systems (1979-83), TRW (1980-81), and Uniform Software (1983-84). His current research interests include graphic user interfaces for expert systems and simulation, concurrent processing, and new techniques for knowledge representation.

Jeff Rothenberg
Information Sciences Department
The Rand Corporation
1700 Main Street
Santa Monica, CA 90406 , U.S.A.
(213) 393-0411