2

A RAND NOTE

Knowledge-Based Simulation: An Interim Report

Jeff Rothenberg, Sanjai Narain,
Randall Steeb, Charlene Hefley,
Norman Z. Shapiro

July 1989

RAND

The research described in this report was sponsored by the Defense Advanced Research Projects Agency under RAND's National Defense Research Institute, a Federally Funded Research and Development Center supported by the Office of the Secretary of Defense, Contract No. MDA903-85-C-0030.

Published by The RAND Corporation
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER N-2897-DARPA	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Knowledge-Based Simulation: An Interim Report		5. TYPE OF REPORT & PERIOD COVERED Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Jeff Rothenberg, Sanjai Narain, Randall Steeb, Charlene Hefley, Norman Z. Shapiro		8. CONTRACT OR GRANT NUMBER(s) MDA903-85-C-0030
9. PERFORMING ORGANIZATION NAME AND ADDRESS The RAND Corporation 1700 Main Street Santa Monica, CA 90406		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Department of Defense Arlington, VA 22209		12. REPORT DATE July 1989 13. NUMBER OF PAGES 77
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

No Restrictions

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Simulation,
Artificial Intelligence,
Models .
Reasoning. (JES)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See reverse side

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Simulation is a modeling technique that is widely used in such areas as policymaking, manufacturing, computer system design, and military analysis. In spite of its widespread use, current simulation technology suffers from a number of serious drawbacks that limit its power, applicability, and credibility. The Knowledge-Based Simulation (KBSim) project at RAND is engaged in research aimed at producing a new-generation simulation environment that will greatly extend the capabilities of current simulation technology, particularly as applied to analytic modeling. The KBSim project has focused on reasoning in simulation, → Keywords → representation of multiple relations among simulated entities, highly interactive interfaces, sensitivity analysis, variation of the aggregation level of a model, and the modeling of "soft" concepts such as initiative. This Note describes the KBSim project, discusses its goals, and presents an overview of its progress.

A RAND NOTE

N-2897-DARPA

Knowledge–Based Simulation: An Interim Report

Jeff Rothenberg, Sanjai Narain,
Randall Steeb, Charlene Hefley,
Norman Z. Shapiro

July 1989

Prepared for
The Defense Advanced Research Projects Agency

RAND

PREFACE

Simulation is a modeling technique that is widely used in such areas as policymaking, manufacturing, computer system design, and military analysis. Despite its ubiquity, current simulation technology suffers from a number of serious drawbacks that limit its power, applicability, and credibility. This Note describes the Knowledge-Based Simulation project at The RAND Corporation, which is being conducted for the Information Science and Technology Office of the Defense Advanced Research Projects Agency (DARPA), under RAND's National Defense Research Institute (NDRI). The NDRI is a Federally Funded Research and Development Center sponsored by the Office of the Secretary of Defense. This project is investigating ways of using knowledge-based techniques to produce a new generation of simulation environments that allow building more powerful, comprehensible, and reusable simulation models. The Note discusses the problems with current simulation paradigms, identifies research issues intended to address these problems, and presents interim results and directions for continued research. It should be of value to developers of simulation environments and researchers concerned with fundamental modeling issues. It should also be of interest to modelers who build analytic simulations and to analysts who use such simulations.

The work discussed here has been coordinated with several other ongoing research projects at RAND, notably the Intelligent Database (IDB) project and the Advanced Geographic Environment (AGE) project. Related RAND research is described in McArthur, Klahr, and Narain (1984); Steeb et al. (1986); and Cammarata, Gates, and Rothenberg (1988).

SUMMARY

Discrete-state simulation is a modeling technique that is particularly useful for analyzing domains involving complex temporal phenomena whose interactions defy complete mathematical analysis. Simulation models typically consist of descriptions of real-world entities to be modeled and "first-order" interactions among these entities; running the simulation to observe the effects of these first-order interactions often reveals higher level interactions and effects that were not known in advance, answering questions of the form "*What if . . . ?*" This traditional view of simulation (and the technology that supports it) is severely limited in its ability to answer other kinds of questions of at least equal value; it is also limited in its ability to represent a wide range of phenomena in ways that are comprehensible to both model builders and users.

The lack of explicit representation of domain knowledge in simulations makes it difficult to verify the correctness of their underlying models and makes them hard to comprehend. Recent advances in object-oriented simulation languages, rule-oriented approaches, logic programming, automated inferencing, and interactive graphics are facilitating a new generation of simulation environments. The synergy of these techniques promises to revolutionize simulation, transforming it into something far more powerful, more useful, and more believable. It is this synergy that we call "knowledge-based simulation."

The Knowledge-Based Simulation ("KBSim") project at The RAND Corporation is engaged in research aimed at producing a new-generation simulation environment that will greatly extend the capabilities of current simulation technology, particularly as applied to analytic modeling. This work has focused on the following areas:

- Reasoning in simulation
- Representation of multiple relations among simulated entities
- Highly interactive interfaces
- Sensitivity analysis

- Variation of the aggregation level of a model
- Modeling of "soft" concepts (such as *initiative*)

This Note describes the KBSim project, discusses its goals, and presents an overview of its progress to date.

We have produced a number of papers summarizing our research results in several areas. We have also produced several prototype facilities that demonstrate new capabilities. The following summarizes these accomplishments and discusses them briefly.

Reasoning in Simulation

- ROSS-in-Prolog and DMOD formalisms
  - Proof-of-concept demonstration

Multiple Relations (and Extended Objects)

- Analysis of object-oriented simulation paradigm deficiencies
  - Winter Simulation Conference paper (Rothenberg, 1986)
  - KBSim Progress Report (this RAND Note)

Highly Interactive Interfaces

- Graphics-delta, dependencies, demons
  - Extended ROSS ("XROSS") capability
  - Winter Simulation Conference paper (Cammarata, Gates, and Rothenberg, 1987)
- Graphics environment
  - Graphics-in-LISP (GIL) capability using GKS graphics standard

Sensitivity Analysis

- New *propagative* approach
  - Analytic evaluation of the new approach (LISP)
  - Initial prototype computational environment (LISP)
  - RAND Note (in preparation)

Variation of the Aggregation Level of a Model

- Variable aggregation
  - Demonstration
  - RAND Note (in preparation)

Modeling of "Soft" Concepts

- Identification of key issues and approaches

We have identified a wide range of issues and shortcomings in current object-oriented simulations, based on extensive experience with this approach at RAND and elsewhere. This has led to the initiation of the new language design effort to be undertaken by the Transfer of Simulation Technology project at RAND.

We have implemented a subset of the ROSS language in Prolog as a proof of the feasibility of using logic programming in an object-oriented style. This demonstration formed the basis for the new DMOD language with which we are pursuing these ideas further.

We have designed and implemented solutions to several kinds of artifacts in simulations, having to do with the difficulty of keeping graphic displays up to date without excessive coding effort on the part of the modeler (the "graphics-delta" technique), with representing dependencies among simulation entities, and with representing "autonomous" behavior in a simulation. These have resulted in an interim, extended version of the ROSS language, referred to as XROSS (described in Cammarata, Gates, and Rothenberg, 1987 and 1988).

We have built a graphics environment using the GKS graphics standard and SunWindows, and have integrated this with the XROSS extensions described above.

We have implemented a LISP program that analyzes the expected "payoff" from our new sensitivity analysis approach; this led us to conclude that a computational environment based on this approach would be well worthwhile. We have implemented an initial prototype of such an environment.

We have produced a simplified demonstration that exhibits variable resolution, i.e., varies the aggregation of simulated entities such as companies and platoons. This also uses our graphics environment and XROSS extensions, as well as incorporating the beginnings of a planning facility for modeling decisionmaking.

In addition, we have explored the modeling of "soft concepts" (such as a commander's use of *initiative*) and have identified some promising approaches to integrating such models into battlefield simulations.

We have coordinated our work on multiple relations with RAND's Intelligent Database (IDB) project, and our ideas on extended objects are being coordinated with RAND's new Advanced Geographic Environment (AGE) project; our ideas on aggregation will be further developed in coordination with both of these projects. Ultimately, we expect to use the Intelligent Database facility for storing and retrieving permanent definitions of submodels, objects, events, and cases, and we may be able to use aggregated terrain data derived from the AGE project. We also eventually hope to be able to use the results of the CPAS (Concurrent Processing for Advanced Simulation) project to run our simulation environment in a distributed manner for improved performance.

CONTENTS

FIGURES

TABLE

I. INTRODUCTION

Modeling in its broadest sense is the cost-effective use of something in place of something else for some cognitive purpose. Every model has a referent and a purpose with respect to this referent, and it must be more cost-effective (in some relevant coin) to use the model for this purpose than to use the referent itself (Rothenberg, 1986 and 1989). Modeling therefore makes it possible to study systems that may be too dangerous, too expensive, or even impossible to observe physically.

It has long been possible to build models whose complexity defies understanding. One of the most common complaints among military analysts is the incomprehensibility of the models available to them. The quest for detail, the hunger for performance (to support this detail), and the "fetish of realism" conspire to create huge models, whose correctness as programs (to say nothing of their validity as models) can only be taken on faith. Though there is certainly good reason to explore new techniques for increasing the detail of models and for modeling new phenomena (as well as for improving performance), it is at least as important to explore new ways of improving their comprehensibility.

Incomprehensibility leads to two distinct (though related) problems. The first is essentially a software engineering problem: An incomprehensible program is unlikely to be *correct*, as well as being difficult to modify or maintain. The second is a modeling problem: An incomprehensible model is unlikely to be *valid*. Both the model and the program that implements it must be comprehensible in order to have any confidence that the program correctly implements a valid model. Even if a model can be validated (which is problematic in the military domain), it is clear that an incomprehensible implementation of a model cannot be guaranteed to preserve the model's validity. On the other hand, the most comprehensible implementation of an incomprehensible model (or of any model that has not been validated) can at best achieve "face validity" (the superficial appearance of being valid).

Knowledge-based simulation is a new approach to modeling. Its goal is to facilitate building simulations of greatly increased power and comprehensibility by making use of deeper knowledge about the behavior of the simulated world.

Such knowledge is usually omitted from traditional simulations since they are not able to utilize it; in some cases, the knowledge may be present implicitly, but simulations remain powerless to interpret it or take advantage of it. This lack of explicit knowledge in simulations limits them severely: It makes it difficult to verify the correctness of their underlying models, makes them hard to comprehend, and restricts the kinds of questions they can answer. Recent advances in object-oriented simulation languages, rule-oriented approaches, logic programming, automated inferencing, and interactive graphics are facilitating a new generation of simulation environments. The synergy of these techniques promises to revolutionize simulation, transforming it into something far more powerful, useful, and believable. It is this synergy that we call "knowledge-based simulation."

Simulation is traditionally conceived in a rather narrow sense. There is considerable confusion in the literature about what the term really means (or ought to mean) and how it relates to modeling as a whole (Greenberger, Crenson, and Crissey, 1976; Kiviat, 1967; Quade, 1985; Hughes, 1984). Nevertheless, there is general agreement that simulation involves some kind of behavioral analog of the entity or phenomenon being modeled (Dalkey, 1968; Gass and Sisson, 1974). That is, simulation is concerned with modeling the unfolding of events or processes over time. Though simulation is often spoken of as using a particular *kind* of model, it is more useful to consider it as a *way of using* a model, i.e., the process of using a model to trace and (hopefully) understand the temporal behavior of some system. A "simulation model" is any model designed for use in this way.

Unfortunately, this is often construed in its most limited sense: Simulation is seen as the process of building a behavioral model (which may include probabilistic effects), setting it up in some initial configuration, and then "running" it to see what happens. This can be thought of as the "toy duck" view of simulation ("wind it up and see where it goes"), which corresponds to asking questions of the form "*What if . . . ?*"; i.e., "*What* would happen *if* a system having the given behavior were to proceed from the given initial state?" While this ability to run simulations and ask them "*What if . . . ?*" questions may distinguish them from other kinds of modeling, it is only one of the ways that simulations can be used. Freeing simulation from this limited, traditional view yields a powerful new approach to modeling, with vast, untapped potential.

Developing the potential of this new approach will be of tremendous value to modelers in both military and other domains, particularly when using simulation for analytic purposes. Simulation is often a technique of last resort for understanding systems that are too complex to be modeled in other ways; it has the appealing quality of showing how a system behaves and evolves under certain assumptions and conditions. But this is rarely an end in itself. More often, it is a means of understanding causality or other implicit relationships (whether static or dynamic) among elements of the system. This is particularly true of "discrete-state" approaches to simulation, which are usually motivated by the lack of a more definitive analytic understanding of the system being modeled; the hope is that by running the simulation and observing its behavior, deeper understanding will emerge.

In this pursuit, the traditional approach to simulation is sharply constraining, since it only allows observing sequences (and consequences) of events:

> In principle . . . a simulation is the least desirable of models. It has low insight, since it does not provide explanation of the observed outcomes. . . . Nevertheless, it may be a correct choice as a model, if only because no other choice is open.
>
> —*Bowen (1978)*

The "toy duck" approach to simulation can be thought of as performing a degenerate form of inference, in which a result state is inferred from an initial state.[1] To fulfill its true potential, simulation must be allowed to perform inferences of a much wider variety, especially when used for analysis. Appropriate techniques for performing such inferences have been developed in the expert systems arena; these techniques rely on the explicit representation of domain knowledge in symbolic form.

[1]It is possible to view any computation as a way of making explicit what is initially only implicit, "inferring" the former from the latter. In the degenerate case, this simply involves computing an explicit result that was implicit in a program's input (where "input" is taken to include the program itself). For example, a simple program that adds numbers can be thought of as "inferring" their sum from the input numbers, along with the summation algorithm.

Knowledge-based simulation asserts that any model should take maximum advantage of the knowledge available to it. Building a simulation model is a knowledge-intensive task, yet much of the knowledge accumulated in this process is typically discarded, being represented only implicitly—if at all—in the resulting model. (In this context, *knowledge* can be operationally defined as information that is represented in an explicit form that is understandable and meaningful to humans, i.e., "human-meaningful" information.)

Most simulations specify only what action to take, based on the current situation; they contain no explicit description of why the actions are necessary, no representation of the motivation or intention of an object's actions, no depiction of the reasoning process that leads to an action, and no explicit notions of causality or other relationships among events. As a consequence, these simulations cannot answer questions that require interpreting or reasoning about knowledge—even when such knowledge could have been represented explicitly in the model with relatively little effort.

Capturing this knowledge during the design of the simulation and representing it explicitly in the underlying model allows asking a much wider range of questions of simulations, extending far beyond "*What if . . . ?*" For example, traditional simulations are generally incapable of explaining *why* a given sequence of events occurred, nor can they answer definitive questions (such as "*Can this event ever happen?*") or goal-directed questions (such as "*Which events might lead to this event?*").

In addition to these reasoning limitations, the lack of explicit knowledge limits a simulation's ability to interact intelligently, responding to the user's perceptual and cognitive needs (for example, knowing what constitute "exceptional conditions" in a simulation so that these can be brought to the user's attention when they occur).

One of the central tenets of knowledge-based simulation is that the knowledge required in building a simulation should be represented explicitly whenever possible, in such a way that the resulting model will be amenable to automated inferencing and querying by the user, in addition to simulation *per se*. This approach is at once a broad new view of simulation and a merging of traditional modeling with knowledge-based expert system techniques.

Although analysts have grown accustomed to the limitations of the traditional form of simulation, freeing simulation from these limitations can provide analysts with powerful new tools that will facilitate building, debugging, and using analytic models. For example, traditional "*What if . . . ?*" simulation is most appropriate when applied to situations requiring a choice of one of a small number of alternatives, whereas simulation that goes beyond "*What if . . . ?*" is potentially applicable to more open-ended questions, such as finding optimal solutions or formulating new policy. Similarly, providing explanation facilities for simulations would be of great help in debugging and comprehending the behavior of models.

The Knowledge-Based Simulation (KBSim) project at RAND is exploring new techniques for modeling in the sense described above. To focus our efforts, we are developing several prototype military simulations that provide testbeds for exploring knowledge-based simulation.

The remainder of this Note is organized as follows: Section II discusses the current state of the art of simulation, focusing on modern object-oriented approaches to discrete-state simulation. Section III provides what we feel is a necessary (though rare) characterization of the uses of simulation in the military domain and shows where our work falls within this context; Section IV motivates our work in terms of the needs of simulation builders and users, according to this characterization. Section V discusses our project goals and the approach we have pursued, while Section VI discusses the derivation of our research tasks from these goals and describes our results in detail. Section VII presents some overall conclusions and suggests directions for future research.

II. THE STATE OF THE ART OF SIMULATION

This section provides background on the current state of the art of simulation, focusing on modern object-oriented approaches to discrete-state simulation, and showing how the KBSim project has grown out of previous RAND research in this area.

The KBSim project is the descendant of many years of research at RAND in the areas of modeling, simulation, gaming, object-oriented language design, interactive graphics, and artificial intelligence (AI). RAND's long history of simulation research has included the development of the widely used SIMSCRIPT language (Kiviat, Villanueva, and Markowitz, 1968) and many theoretical and experimental results in game theory, Monte Carlo simulation, and military wargaming (Conway, 1962; Kamins, 1963; Ginsberg, Markowitz, and Oldfather, 1965; Sharpe, 1965; Voosen, 1967). (A well-known text on policy analysis notes that "The Rand Corporation has been the site of more important methodological work on modeling of various kinds than any other institution in the United States" (Greenberger, Crenson, and Crissey, 1976).) Similarly, the work of Newell, Shaw, and Simon at RAND in the 1950s (Newell, Shaw, and Simon, 1957; Klahr and Waterman, 1986) was one of AI's earliest results and defined many continuing focal points for AI research. More recently, RAND's work in expert systems has produced the languages RITA (Anderson and Gillogly, 1976; Anderson et al., 1977) and ROSIE (Sowizral and Kipps, 1985; Kipps, Florman, and Sowizral, 1987) as well as a number of expert system applications such as LDS (Waterman and Peterson, 1981), TATR (Callero, Waterman, and Kipps, 1984), and SAL (Paul, Waterman, and Peterson, 1986). Finally, RAND's research in interactive graphics has produced the RAND tablet (Davis and Ellis, 1964), the GRAIL system (Ellis, Heafner, and Sibley, 1969), and fundamental ideas on the use of interactive map displays (Anderson and Shapiro, 1979).

RAND began applying AI and graphics to simulation in the late 1970s and early 1980s. The development of the object-oriented ROSS language (McArthur and Klahr, 1982; McArthur, Klahr, and Narain, 1984 and 1985) and its use in the simulations SWIRL (Klahr et al., 1982) and TWIRL (Klahr et al., 1984) as well as derivative work elsewhere (Nugent and Wong, 1986; Hilton, 1987) clearly demonstrated the potential benefits for simulation technology. Subsequent work at RAND has continued to build on this foundation, leading to research in areas such as cooperative intelligent systems (Steeb et al., 1986) and tutoring tools for simulations (McArthur, 1987).

This work has produced new technology that can aid in building, maintaining, and understanding simulations. Object-oriented simulation as implemented in ROSS provides a rich, 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, comprehended, and modified by domain experts and analysts far more effectively than with previous approaches (Klahr, 1985). It hides irrelevant details of object implementation and allows model behavior to be viewed at a meaningful level. The use of objects allows encapsulating information, associating the behavior of an entity with its state definition, and modeling certain real-world entities (particularly those that are relatively dense and unstructured, such as trucks or aircraft) in a natural way. Similarly, it provides a natural way of modeling static, taxonomic relationships among objects by the use of class-subclass hierarchies, while minimizing the redundancy (and therefore the possible inconsistency) of their definitions through the inheritance of attributes and behaviors over these hierarchies. Finally, it provides a natural way of modeling certain kinds of dynamic interactions among real-world entities by the use of messages passed between objects.

Despite these achievements, however, object-oriented simulation retains a number of critical limitations in the power and flexibility of modeling, the representation of knowledge, the integration of different modeling paradigms, and the comprehensibility, scalability, and reusability of models (Rothenberg, 1986; McArthur, 1987). In particular, the treatment of objects, taxonomies, inheritance, and messages in ROSS (and similar object-oriented languages) is too constraining and provides too little leverage for performing the kinds of inference discussed above. For example, the emphasis on class-subclass relations among objects tends to ignore other, equally important relations, such as part-whole. More fundamentally, the strict object-oriented approach leads to a preoccupation with attributes (i.e., state) and behaviors (i.e., procedures), which produces a limited view of discrete-state simulation.

Discrete-state simulation is an approach to modeling that views time as consisting of discrete states rather than a continuous flow. In cases where the reality to be modeled is continuous and can be described analytically (i.e., by a set of closed-form equations of the form "*f(t)* = . . ."), analytic or numerical solutions of those equations can be used to generate the time-dependent behavior required for simulation. (Note that our definition of simulation as a way of using a model to trace and understand the temporal behavior of a system allows us to consider certain analytic models as simulation models.)

Though closed-form solutions are often mathematically elegant, their elegance may make them cryptic and incomprehensible. There are also cases in which analytic solutions are known but are computationally intractable. Nevertheless, analytic models are appropriate in many situations, particularly when dealing with complex physical phenomena involving vast numbers of relatively small and similar entities whose individual interactions are relatively simple and whose aggregate interactions permit statistical treatment. It is important to allow the use of embedded analytic models in any simulation environment, since such models often represent at least one form of complete understanding.

There remains a large class of problems, however, that involve inherently discrete phenomena or that are not well enough understood to be handled analytically (i.e., which do not have formal mathematical solutions). For these, a discrete-state approach is more effective. These problems usually involve small to large (but not "vast") collections of interacting entities, each of whose behavior is understood reasonably well in isolation and whose low-level, pairwise interactions with each other are known but whose high-level, group interactions are not well understood. The strategy of discrete-state simulation is to encode these known, low-level interactions in the hope that the overall behavior of the resulting simulation will approximate that of the reality being modeled and (ideally) that higher-level interactions will reveal themselves.

The reality modeled by a discrete-state simulation typically consists of a moderate number of entities having relatively distinct—or unique—behaviors that cannot be accurately modeled by a small number of analytic techniques. These entities typically fall into a relatively small number of "classes" of entities with similar attributes and behavior. The ways that individual entities interact with other entities within and across these classes, while often complex, are typically understood in some detail; what is *not* typically known is how to describe the aggregate behavior of collections of these entities.

Note that the behaviors and interactions of these entities may involve probabilistic or stochastic effects; our view of discrete-state simulation is not restricted to purely deterministic cases. Randomization may be used (1) to model intrinsically random phenomena (such as weather), (2) to model deterministic processes that are poorly understood (such as equipment failure), or (3) to eliminate unnecessary detail by approximating a deterministic process (such as the arrival of trucks at a depot) by a probabilistic one.

Time is dealt with in discrete-state simulations as a succession of separate states in which entities interact; time advances by discrete state-transitions, either at fixed "ticks" of a simulated clock (referred to as "time-based" simulation) or whenever something significant happens (referred to as "event-based" simulation).

Discrete-state simulation can therefore be viewed as a last resort for modeling certain kinds of intractable problems. Its power lies in its ability to reveal high-level patterns of interaction that cannot be recognized in other ways. It is often possible to enumerate and describe a collection of entities and their immediate interactions without knowing where these interactions lead; if this knowledge is encoded in a discrete-state simulation and the behavior of the resulting model is observed, deeper understanding will often emerge.

The object-oriented approach to discrete-state simulation uses objects to encapsulate state and represents state transitions by messages between objects. This distributes both the state of the simulation and the conditions for state transitions among the objects in the model. While this serves the software engineering goals of data hiding and modularization, it does not necessarily allow optimal comprehension of the relationships among events (since their definitions are distributed), nor does it enable inferences of the kind discussed above. Although commercial research efforts in simulation are beginning to examine some of these areas (IntelliCorp, 1985; McFall and Klahr, 1986), they have tended to consider domains like manufacturing.

As has been pointed out in the literature (Gilmer, 1986), military simulation is quite different from factory simulation, in which a relatively static collection of objects interact with each other in limited ways in a highly structured world. In contrast, military simulation typically involves a very dynamic collection of objects that are continually changing their composition and are often created or destroyed. These have constantly changing and often adversarial relationships to one another and behave and interact in widely varying and unpredictable ways in a world that is itself in constant flux. Furthermore, a wide range of phenomena may have to be taken into account in a military simulation for it to be credible, including factors such as terrain, weather, command and control, and human decisionmaking. This lack of structure and inherent complexity make it difficult to design military simulations and to comprehend, interpret, or validate them once they have been built. Conventional simulation environments generally do not address these problems. Our work in knowledge-based simulation is an attempt to improve the power and comprehensibility of the object-oriented discrete-state approach, as required for military simulation.

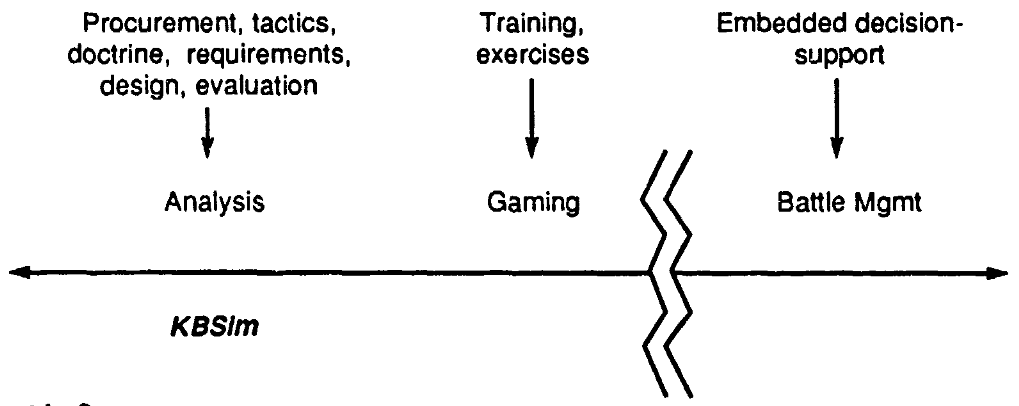
In addition, in order to be comprehensible and usable, a simulation must allow users to query the simulation state graphically, view graphical explanations of the behavior of the simulation (e.g., by animating "causal chains"), specify scenarios graphically, and build or modify simulation objects graphically (for example, defining and exercising new behaviors graphically). Similarly, it should allow users to graphically display and edit diagrams or pictorial representations of relationships among objects. This requires a highly interactive graphics environment that is well beyond the capability of most conventional simulation systems.

In summary, most simulations relate situations to actions with no explicit description of why the actions are necessary, no representation of the motivation or intention of an object's actions, no depiction of the reasoning process that leads to an action, and no explicit notion of causality. This restricts the types of questions simulations can answer to those of the form "*What (happens) if. . . ?*" (i.e., given an initial situation, the simulation is asked to determine future states). Without additional inferencing capabilities, simulations cannot answer questions about what an object should do to accomplish some specified goal, which future states can or cannot possibly exist, which alternative actions could improve an outcome, etc.

These types of questions can only be answered by systems embodying a descriptive model of the domain. Knowledge-based ("expert") systems contain such models as well as the inferencing mechanisms needed for planning and problem solving. Our goal is to extend these knowledge-based reasoning techniques and integrate them with object-oriented discrete-state simulation. In addition, we intend to take advantage of graphical interfaces for making simulations explain their behavior and allowing users to build, modify, and control simulations more directly than is now possible.

III. SIMULATION AND THE KBSIM PROJECT

Although the foregoing discussion and much of what follows applies to simulation in general, the focus of the KBSim effort is in military simulation, which has many distinguishing characteristics, as noted above. However, "military simulation" is not a single entity: There are at least three distinct uses of simulation in the military context, each with its own characteristics and requirements. These can be thought of as lying along a "continuum of stringency," as illustrated in Figure 1. (The following discussion describes the


Procurement, tactics, doctrine, requirements, design, evaluation
Training, exercises
Embedded decision-support
Analysis
Gaming
Battle Mgmt
KBSim

must be?			
interactive	✔	✓	✔
flexible	✔	✓	
realistic	✓	✔	✓
valid	✔	✓	✔
reliable		✔	✔
realtime		✓	✔
portable	✓	✓	
survivable			✔

✓ *important* ✔ *very important*

Fig. 1—Uses of military simulation

details of the figure.) This section characterizes these different uses of simulation and shows where the KBSim project falls within this context.

The left (least stringent and least constrained) end of the continuum shown in Figure 1 represents the use of simulation for analysis, that is, to explore issues and make decisions involving procurement, to evaluate tactics and doctrine, to derive requirements for new systems, etc. The middle of the continuum represents the use of simulation to provide a gaming environment for training or exercises. The right (most stringent and most constrained) end of the continuum represents the use of simulation in battle management, i.e., as an embedded decision support aid for commanders in the field. We discuss each of these briefly, since the distinctions among them are important and are rarely made explicit.

The chart at the bottom of Figure 1 shows the relative qualitative importance of various issues for simulations aimed at analysis, gaming, and battle management; though this list is not exhaustive, it captures some of the major factors involved in designing simulations. Since many of these factors are in competition with one another, it is difficult to satisfy all of them at once (as expressed in the computer science adage, "Fast, cheap, reliable: pick two"). It is therefore necessary to choose a point along the continuum and attempt to satisfy the most important factors at that point. The focus of the KBSim project (or *any* simulation effort) can only be understood and evaluated in these terms.

Though some of the terms in the chart should be self-explanatory (*interactive*, *flexible*, *valid*, and *portable*), some warrant definition. For the purpose of this discussion, we call a simulation *realistic* to the extent that the user sees results as he or she would in the real world; this is not the same as its being *valid*, which means that its results are correct, though they may not be presented as they would appear in the real world. We call a simulation *reliable* to the extent that it can be counted on to keep running without unexpected interruption and to the extent that its results are reproducible from one run to another; this is not the same as calling it "robust," which would mean that it handled all cases, modeled all relevant phenomena, etc. We call a simulation *realtime* if it is fast enough to be used for making decisions in anticipation of real-world events that occur at their natural rates (this is a somewhat specialized definition of the term). Finally, we call a simulation *survivable* if it can be run in a harsh field environment (such as combat).

Analysis uses simulation to answer hypothetical questions in a fairly unhurried and benign setting. This means that while reliability of the simulation environment is desirable, it is not of paramount importance in this setting, whereas realtime and survivability requirements are generally absent. Performance is always important, but the emphasis for analytic simulations is normally on validity (at least some kind of "face validity," such as monotonic behavior); that is, the correctness of results is more important than the speed at which they are produced. However, simulations that support analysis should also be highly interactive to allow users to explore possibilities and understand the behavior of the model (especially when answering questions beyond "*What if . . . ?*"), though this is rarely done in current analytic simulations. This requires that these systems have sufficient speed to be responsive, even though they do not have to satisfy realtime constraints. In addition, analytic systems must be flexible enough to allow users to direct the course of a simulation or modify the model itself as their understanding evolves.

Despite their need for validity, analytic simulations need not necessarily be "realistic" in terms of their presentation of results. Realism (especially graphic realism, such as 3-D perspective, etc.) is often sought for its own sake in simulation, since it is flashy and eye-catching; however, what is important for analysis is the presentation of information to the analyst in an *appropriate* form (i.e., one that is easily comprehended), for which realism may be helpful but is neither necessary nor sufficient. Finally, although portability is useful for analytic simulations (to allow sharing models within the analytic community), this is usually not a primary requirement. However, it is desirable for these simulations to be reusable to minimize the work of building new analytic models that are similar to (or built on top of) existing models; this can be thought of as a special kind of portability, i.e., across time.

In contrast to analysis, a gaming environment uses simulation to provide a surrogate reality in which users can hone their skills or exercise procedures or tactics. Though this has some overlap with analysis, the emphasis is on providing a stable, repeatable environment for training or conducting exercises. This use of simulation is analogous to the use of a flight simulator for training pilots and crews. Of paramount importance for this purpose are the reliability of the environment and the realism of its results. Reliability is dictated by the cost of training and running exercises involving many people and by the need for repeatability of results.

In most current gaming applications, validity is *not* paramount, since the simulation is typically being used to exercise procedural or tactical knowledge rather than to evaluate alternative courses of action. Realism, as in the analytic case, is relative to the needs of the system users. In this case, the users may be trainees or participants in an exercise using the simulation directly, but they are more likely to be intermediate support staff (often called "controllers") who interpret the results of the simulation for the trainees or participants; in exercises involving commanders, for example, these controllers play the role of the staff, digesting and delivering information to the commanders. A realistic simulation minimizes the work required of this staff and improves the fidelity of the simulated world underlying the game. As in analysis, gaming simulations need not be survivable. They also do not need to be especially flexible, except to the extent that they must allow "mid-course corrections" to be made by support staff in order to bring a game or exercise back to its intended track or to align the model with the thinking of the participants (for example, when they feel the model invalidates the game). Finally, portability of gaming systems (as for analytic systems) is desirable, but not mandatory.

Battle management envisions the use of simulation as an embedded decision aid to be used by a commander in making tactical decisions. This is somewhat akin to the analytic use of simulation, but with critical realtime, reliability, and survivability constraints added. Because immediate life-and-death decisions are being made, outcome validity is crucial, as is interactive responsiveness. Realism, as in the analytic case, is subordinate to appropriateness of presentation (unlike the game participant, the commander in the field does not need the simulation to convince him of the reality of the situation). Of all the possible uses of simulation, this is by far the most critical and (so far) speculative. Note that this use may itself be simulated in a game; in this case, there would be two distinct uses of simulation going on in parallel: the game simulation providing the simulated situation and the commander's simulation used as an embedded decision aid. Here the stringency of the constraints on the battle management simulation (particularly survivability) would naturally be relaxed. On the other hand, gaming might be used as an embedded decision aid (i.e., as a "lookahead" to project possible futures), in which case the gaming simulation would inherit the stringency constraints of battle management.

Having produced this characterization of the uses of simulation, the KBSim project has targeted the analytic end of the spectrum. That is, we are concerned with issues of validity, interactive responsiveness, comprehensibility, flexibility, and reusability. In particular, we are not (initially) addressing issues of performance or reliability, except where they are necessary to achieve our goals (for example, our work on sensitivity analysis has focused on performance issues since they pose fundamental problems in this area). While we recognize the significance of performance in simulation, we feel that it is important to avoid introducing performance constraints too early in the process of exploring new techniques.

In light of the above discussion of what users require of simulation environments, KBSim can only attempt to address a small subset of the most important issues. Like a "go" player who selectively places stones at the most strategic points in order to secure the desired territory, we have focused on what we feel are critical tasks that provide the greatest leverage in advancing the state of the art of simulation. These tasks have been chosen on the basis of both their inherent importance and our own research strengths and organ ational context. They are discussed in detail in the following sections.

IV. THE NEEDS OF SIMULATION BUILDERS AND USERS

Simulation models are typically substantial pieces of computer software.[2] As with any software, they can be characterized by the needs and constraints of two distinct (though possibly overlapping) groups of people: their builders and their users. Simulation builders and designers are a combination of modelers and (somewhat less frequently) computer scientists attempting to represent abstract models as concrete computer programs. Simulation users are analysts, students, and decisionmakers attempting to gain insight into some aspect of the real world by using a model, presumably because using the reality itself for this purpose would be impossible, unsafe, unaffordable, or at least inconvenient. Traditional simulation technology suffers many serious limitations from the points of view of both of these groups. Recent advances in software engineering for building simulations, as discussed above, have addressed some of these limitations, but not others. This section discusses the major problems experienced by simulation builders and users in order to motivate the research undertaken by the KBSim project.

Simulation builders require tools for defining new models while taking advantage of existing ones (i.e., reusing old models). Ideally, they would like to be able to validate their models, either against reality or (at least) against a consensus of experts. In the absence of true validation (which can be very difficult in domains like military simulation), model builders must settle for tools that allow making their models comprehensible, credible, and internally consistent. They also require computationally feasible ways of performing sensitivity analysis to verify the stability of a model with respect to its parameters and identify which parameter values must be determined with critical precision.

In most cases, a modeling effort is undertaken to explore or reproduce some relatively small part of a domain, which we refer to as the "model of interest."

[2]While we recognize that simulation does not logically imply the use of a computer, we focus on this case since most practical applications of simulation involve significant computation.

This is typically embedded in a larger world model, which supplies the necessary environment for the model of interest, but is not itself of particular interest to the modeling effort; we refer to this as the "embedding model." For example, a model of electromagnetic propagation might be required to study communication patterns in a command and control setting; here, the communications model might be the model of interest while the propagation model would be the embedding model. The embedding model is itself contained in a simulation environment that allows running the model, displaying results, etc. These relationships are illustrated in Figure 2.

From the viewpoint of both the model builder and the model user, the work and code required to provide the embedding model and the simulation environment are mere overhead and distraction. The mechanisms required to provide anything other than the model of interest are "artifacts" of the modeling technology in that they complicate and obscure the code that implements the model of interest. (Note that this quality of being "artifactual" is relative: Something that may be part of the model of interest in one case may be an artifact of the embedding model or even the simulation environment in another case.) One of

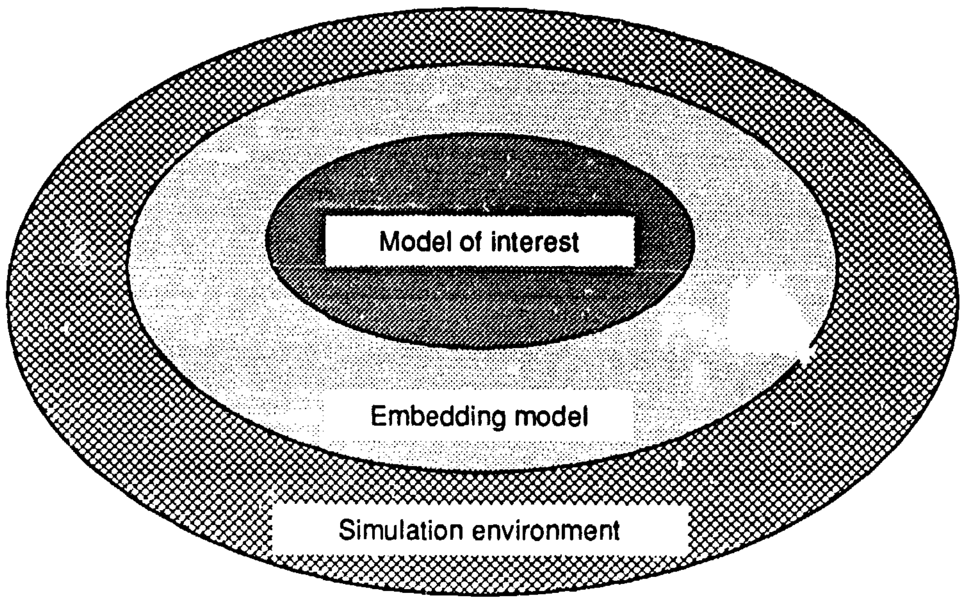


Fig. 2—Model of interest

the simulation builder's primary goals, therefore, can be viewed as the elimination of such artifacts. Even though it may not be logically possible to eliminate the embedding model and the simulation environment, their implementation should be made invisible to the greatest degree possible. This implies that the simulation environment must provide mechanisms for hiding these artifactual aspects of the simulation. Similarly, graphic display, user interaction, planning mechanisms, etc., must be provided transparently by the simulation environment since they are required by most simulation projects.

For example, a fundamental notion of "autonomy" is missing from most simulations. Objects in a discrete-state simulation interact with each other essentially by reacting to stimuli (generally represented by messages in the object-oriented paradigm). This does not capture the idea of an object that moves of its own accord: To simulate this requires crude circumlocution in most simulation environments (e.g., having a scheduler ask each movable object to move each time the simulation clock advances). Similarly, interaction events like collisions, intersections, and detection by sensors are difficult to simulate without additional artifactual devices.

As more and more complex models are developed, it becomes increasingly important to allow model builders to "stand on the shoulders of giants" by building on previous, existing models. This is particularly appropriate in the case of embedding models, which are often similar across many simulation projects; for example, the basic physics of movement, electromagnetic propagation, weather, etc., are phenomena that are required in many embedding models. Simulation languages and environments must provide better mechanisms for defining and storing models in ways that allow sharing and reusing them in later efforts.

Simulation builders also need to be able to define, examine, and modify entities in their models and relationships among these entities. Although object-oriented simulation languages provide a natural way of representing certain kinds of real world objects (such as tanks and airplanes), there are other kinds of entities and phenomena that stubbornly resist most current attempts at representation. For example, entities like terrain and weather, phenomena like human decisionmaking, and "soft" factors like "initiative" are difficult to represent in current paradigms.

Many of the needs of simulation users are analogous to those of simulation builders, particularly when the builder of a simulation is also its user or when users continue to develop and modify a simulation after it is built. Users need to be able to develop confidence in the validity of a model and comprehend what it is doing and why. This requires that a simulation be able to *explain* its behavior in meaningful terms (for example, by showing chains of causality) and that users be able to view the attributes of entities and the relationships among entities in the model in appropriate ways. Users also need to be able to define new relationships among entities in order to define their own "views" of a model, and they need to be able to specify associated inferences to be performed across these relationships. Though most existing object-oriented simulation environments provide *class-subclass* ("IS-A") relations with the ability to infer attributes and behaviors via *inheritance*, other relationships are usually unsupported. For example, a user might want to define a *connectivity* relation—such as that between entities that are in communication with one another—along with a *transitive closure* inference to determine which entities can receive messages from which others through the communication net. Implementing such relations typically requires extensive programming.

Users also need to be able to *explore* alternative assumptions and run "excursions" to test the model and to apply it for different purposes. Similarly, users often need to view the behavior of a model at different levels of aggregation for different purposes. While this can be done after the fact (by "abstracting" the results of a disaggregated model), it is preferable to allow interaction with the model at various levels of aggregation while it is running, which requires that the model be able to change its level of aggregation dynamically. Dynamic aggregation (sometimes referred to as "variable resolution") would also allow the user to focus attention on some aspect or area of a simulation or to run "broad brush" (highly aggregated) simulations to identify interesting cases that can later be rerun in more detail (disaggregated).

Users need to be able to ask a broad range of questions of a model, including not only "*What if . . . ?*" questions but also explanatory questions, questions about the causality of events, questions about the goals and plans of entities, definitive questions about which things can ever happen, and goal-directed questions about how to achieve certain objectives. They also need to be able to analyze results, analyze the sensitivity of results to variations in parameters, and analyze the stability of the conclusions reached by the model.

To summarize, current simulations and simulation environments are limited in their power, comprehensibility, interactive responsiveness, and ability to answer questions beyond "*What if . . . ?*" Both simulation builders and users need environments that provide richer representations for real-world entities and the relationships among them, greater transparency for greater credibility, more intimate interaction for controlling and modifying simulations, and automated inferencing techniques to answer a range of questions beyond the capabilities of traditional simulation.

V. KBSIM GOALS AND APPROACH

The purpose of the KBSim project is to perform research toward the development of the next generation of military simulation environments. This requires making simulations easier to build, understand, believe, use, and reuse.

These goals are in harmony with several other efforts at RAND, including the Intelligent Database project (IDB), the Simulation Technology Transfer project, and exploratory work in new representations for terrain, weather, and other distributed phenomena. The ultimate production of a next-generation simulation environment will involve the integration of these efforts. KBSim has therefore focused on the following specific (overlapping) goals:

- Comprehensibility / believability
- Extended modeling power
- Intelligent exploration and explanation
- Model development aids

The quest for comprehensibility and believability involves such fundamental issues as rethinking the discrete-state paradigm and specifying the behavior of simulation entities declaratively rather than procedurally. It involves representing entities and relationships among entities, events, and causality in more natural ways; eliminating or hiding artifacts (both those of the simulation environment and those of embedding models); and developing new techniques for tracing the behavior of simulations and performing sensitivity analysis.

Extending the modeling power of simulations involves developing new representations for real-world entities that have traditionally proven hard to capture in simulation, developing new forms of inference for answering new kinds of questions, and providing "inferential support" for relations among entities in a model.

Intelligent exploration and explanation is our name for the need to allow users to explore simulations by stopping, backing up, making changes, and trying excursions, along with the ability to trace how or why a simulation behaves the way it does. This has implications for the way a simulation is structured (to allow its being stopped and backed up) and for the interface used to control and interrogate the model.

New model development aids include developing simulation language constructs that highlight the model of interest in a simulation by minimizing artifacts; integrating object-oriented simulation languages with object-oriented databases to store "permanent" simulation entities and behaviors; and providing new interface techniques for specifying relationships and defining behaviors of entities.

To achieve these goals, we have adopted an approach that involves:

- Developing new knowledge-based techniques to increase the power, comprehensibility, and flexibility of simulations
- Developing advanced user interface techniques for improved interaction and the elimination of artifacts
- Integrating these in an extended object-oriented framework

Within the limits of our resources, we are attempting to redefine and extend the object-oriented paradigm, rethink some of the most basic assumptions of discrete-state simulation, integrate this with the best ideas from expert systems and logic programming, and embed the result in a highly interactive environment. (Note that we are specifically *not* targeting a number of other important research topics, such as modeling stochastic phenomena and improving simulation performance.)

In order to reduce risk, we have adopted the approach of prototyping key techniques separately, developing "sublanguages" or stand-alone environments to demonstrate the feasibility of each technique and to allow experimenting with it unfettered by the need to integrate it prematurely with other equally experimental techniques. Once a given technique has proved feasible in isolation, we will integrate it with other techniques that have been similarly proven. The lessons learned from this process are providing direction to the language development effort of RAND's Simulation Technology Transfer project, which is producing a new integrated language to support the next generation of military simulations.

Integrating these new techniques in a testbed simulation environment involves combining many new capabilities. The target external architecture of this environment is shown in Figure 3. The combined model-building environment and model-execution facility (center ellipse) emphasizes the fact that model development and use are an integrated process, though the former logically results in a model to be executed by the latter. Similarly, the model builder and

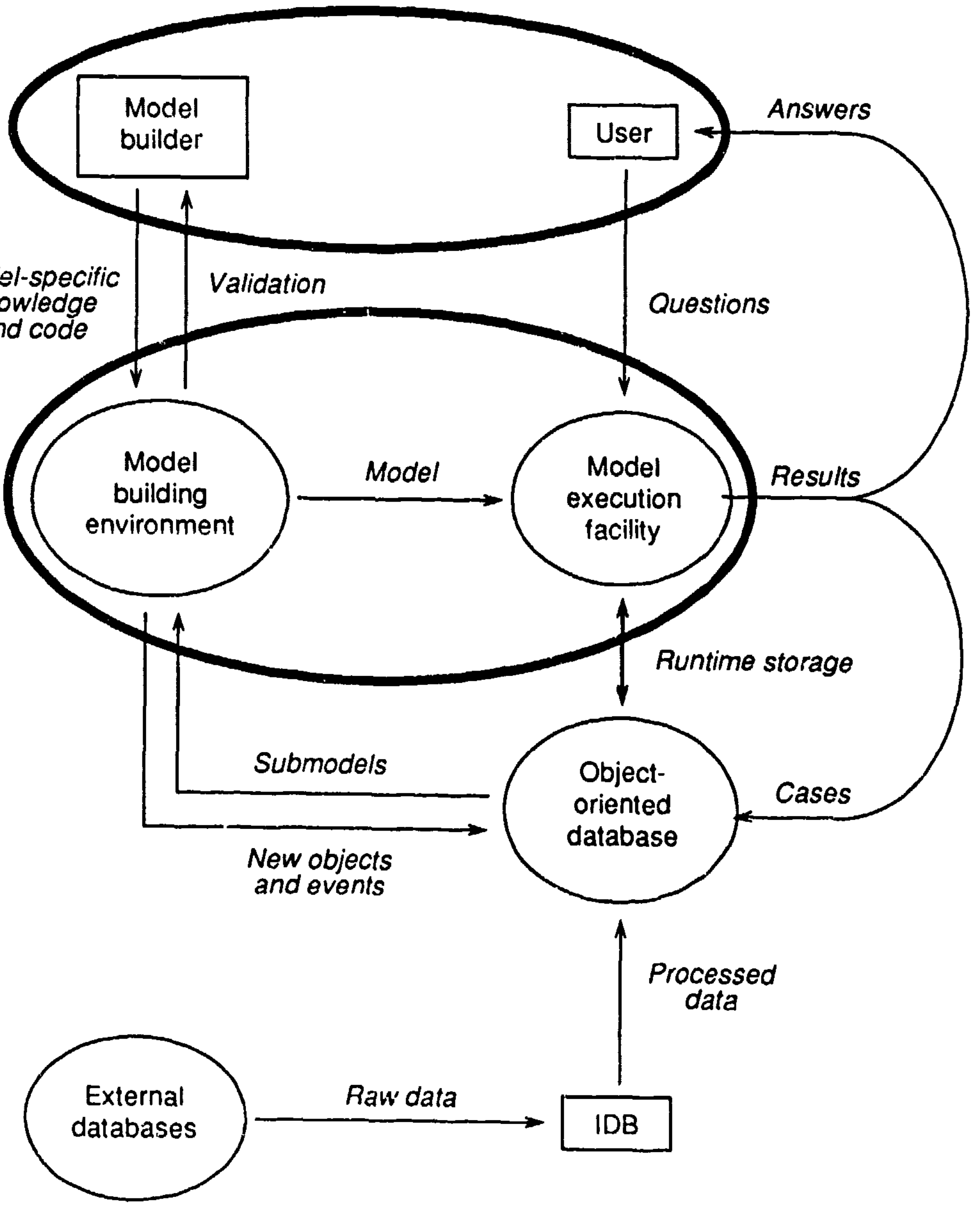


Fig. 3—Eventual external architecture of KBSim

user may be a single person, but logically the model builder supplies knowledge and code (dataflows are shown in italics) and interacts with the model building environment to validate the model being constructed, whereas the user asks questions of the model execution facility and (ideally) gets answers. The object-oriented database (to be supplied by the IDB project) will serve as a permanent repository for submodels, object and event definitions, and cases stored by the user for later analysis.

The internal layered architecture is shown in Figure 4.

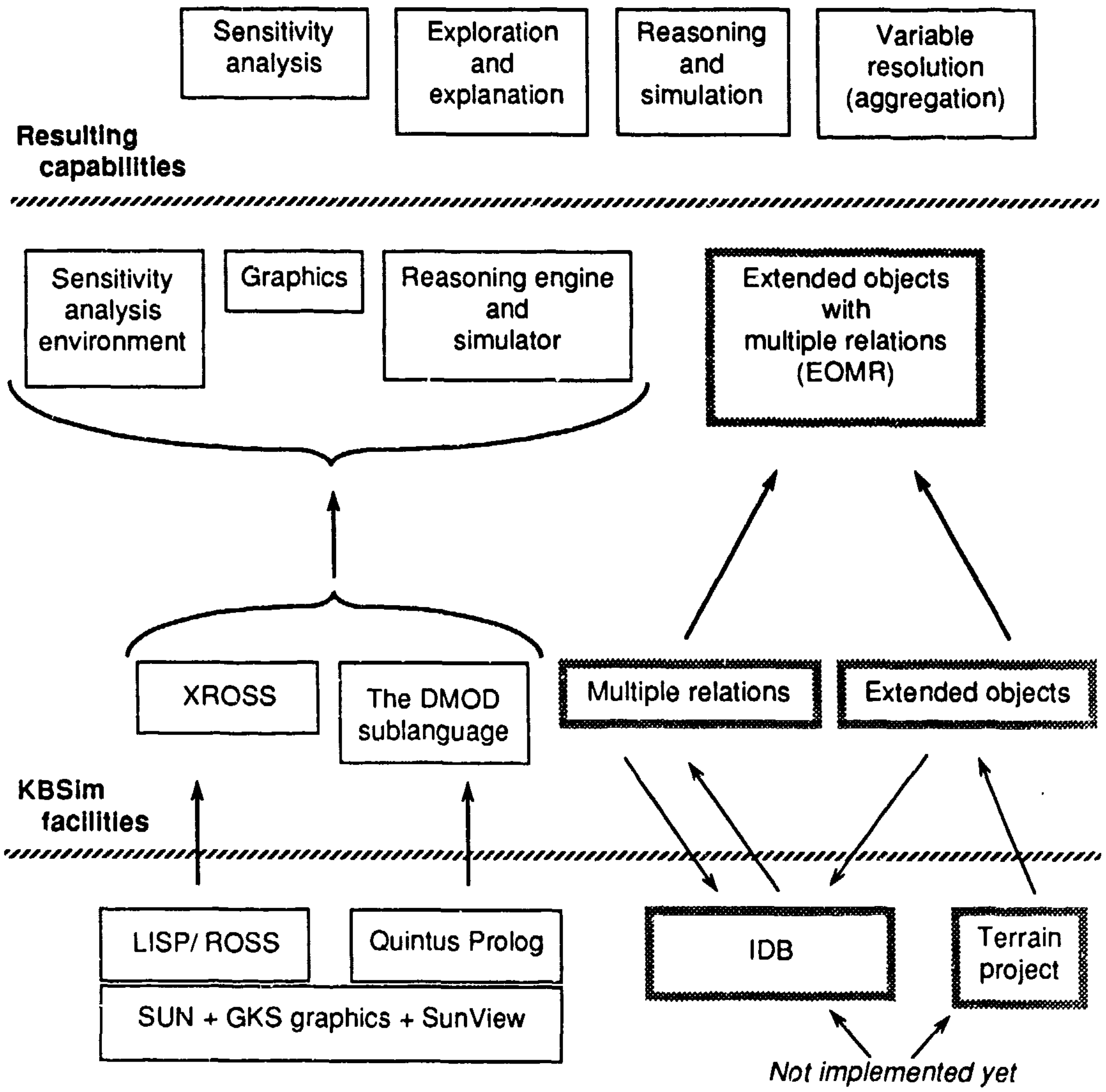


Fig. 4—KBSim internal layered architecture

The lowest layer represents facilities that already exist or that are expected to be provided by other projects at RAND. The middle layer ("KBSim facilities") represents those that are being implemented by the KBSim project itself in order to provide the "resulting capabilities" shown in the top layer. The top layer capabilities are all closely interrelated; they are shown separately as a graphic convention. Arrows pointing upward indicate direct support from underlying facilities. In the case of the Intelligent Database and the proposed Advanced Geographic Environment (AGE) projects, the upward arrows indicate that we expect to develop solutions with the help of input from these projects; in addition, we will use tools or code developed by these projects if appropriate. The arrows pointing downward to the Intelligent Database project indicate that we eventually expect to store objects and relations in the database at runtime.

In the middle layer, the *simulator* facility—which is responsible for running actual simulations—is combined with the *reasoning engine*: These are inseparable in our view of simulation and reasoning as integrated aspects of modeling. This joint facility produces a combination of *reasoning and simulation* capabilities for implementing models. The *exploration and explanation* capabilities provide the enhanced interaction and control that allow the user to manipulate and query the model during a simulation. We show *sensitivity analysis* and *variable resolution (aggregation)* as separate capabilities to emphasize the fact that they add new dimensions to traditional simulation.

To date, KBSim has implemented initial, stand-alone versions of the facilities shown in Figure 4. The DMOD sublanguage (implemented in Quintus Prolog) has been used to develop an integrated reasoning and simulation facility, as well as initial exploration and explanation facilities. The XROSS sublanguage (initially implemented in Franz LISP and currently being ported to Common LISP) has been used to develop the graphics-delta display technique and variable aggregation. An initial sensitivity analysis facility has been implemented as a stand-alone computational environment in Franz LISP. To produce the target architecture shown, further research is required (in coordination with the IDB project) to define multiple relations and to design appropriate runtime interactions between the simulation environment and the object-oriented database. Additional research is also required to explore the relationship between object and event views (discussed above) in order to elaborate the multi-view paradigm. These results will feed into the design of the new simulation language discussed below.

The KBSim architecture shown here does not map directly onto the modeling contexts shown in Figure 2 (Section IV) above. Though it may appear that such a mapping would be desirable in that it would facilitate the elimination of artifacts, this is prevented by the relativity of what constitutes a modeling artifact. The architecture must not presume to know what the model builder and user will consider to be the model of interest, since this will vary. The only artifacts that can be eliminated at the architectural level are those that belong to the simulation environment itself (such as explicit unscheduling and demand update, discussed below). Beyond this, the most that an architecture can do is provide the capabilities needed to implement a wide range of models of interest and embedding world models. The model builder must use the modeling constructs provided by the architecture to encapsulate appropriate aspects of the embedding model in order to eliminate modeling artifacts.

As noted above, we are developing several of our facilities in coordination with other projects at RAND. Figure 5 summarizes the relationship between KBSim and these other projects (as indicated at the bottom of the figure, thin lines labeled in italics denote the flow of ideas and solutions between projects, whereas thick lines denote the shared use of data or code).

In particular, our ideas on multiple relations are being coordinated with the IDB project, our ideas on extended objects are being coordinated with the proposed AGE project, and our ideas on aggregation will be further developed in coordination with both of these projects. Ultimately, we expect to use the Intelligent Database facility for storing and retrieving permanent definitions of submodels, objects, events, and cases, and we may be able to use aggregated terrain data derived from the AGE project. We also eventually hope to be able to use the results of the CPAS (Concurrent Processing for Advanced Simulation) project to run our simulation environment in a distributed manner for improved performance.

In addition, the New Simulation Language project will transfer our ideas—as they yield solutions—into a new, general-purpose simulation language, allowing modelers to apply our ideas to build their models without the need to use our research development environment. This will involve a reimplementation of the KBSim environment that will improve efficiency and integration and support

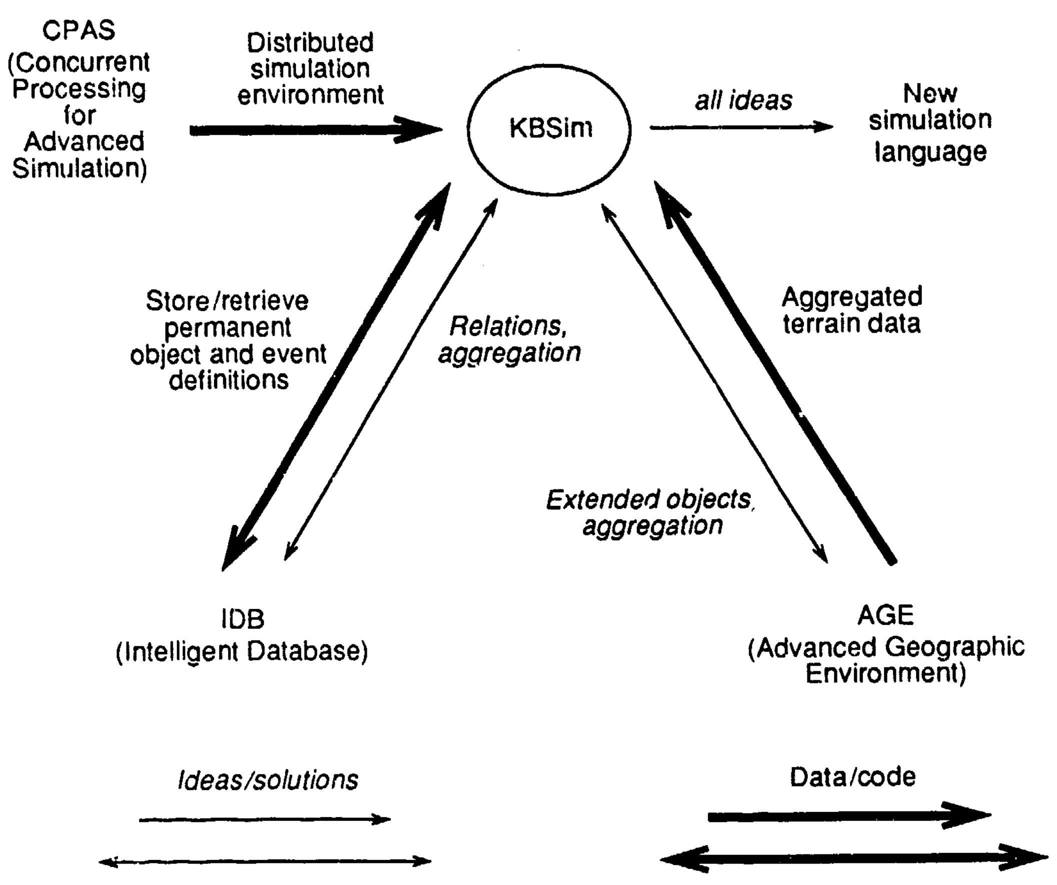

CPAS (Concurrent Processing for Advanced Simulation)
Distributed simulation environment
KBSim
all ideas
New simulation language
Store/retrieve permanent object and event definitions
Relations, aggregation
Aggregated terrain data
Extended objects, aggregation
IDB (Intelligent Database)
AGE (Advanced Geographic Environment)
Ideas/solutions
Data/code

Fig. 5—KBSim and other projects at RAND

the multi-view paradigm that we are currently developing in DMOD. KBSim will reimplement its own facilities in this new language as it evolves, yielding the eventual internal architecture shown in Figure 6.

At the present time, many of the facilities discussed above exist in stand-alone form; their full integration will produce a new environment for powerfu., comprehensible, and reusable modeling by means of knowledge-based simulation.

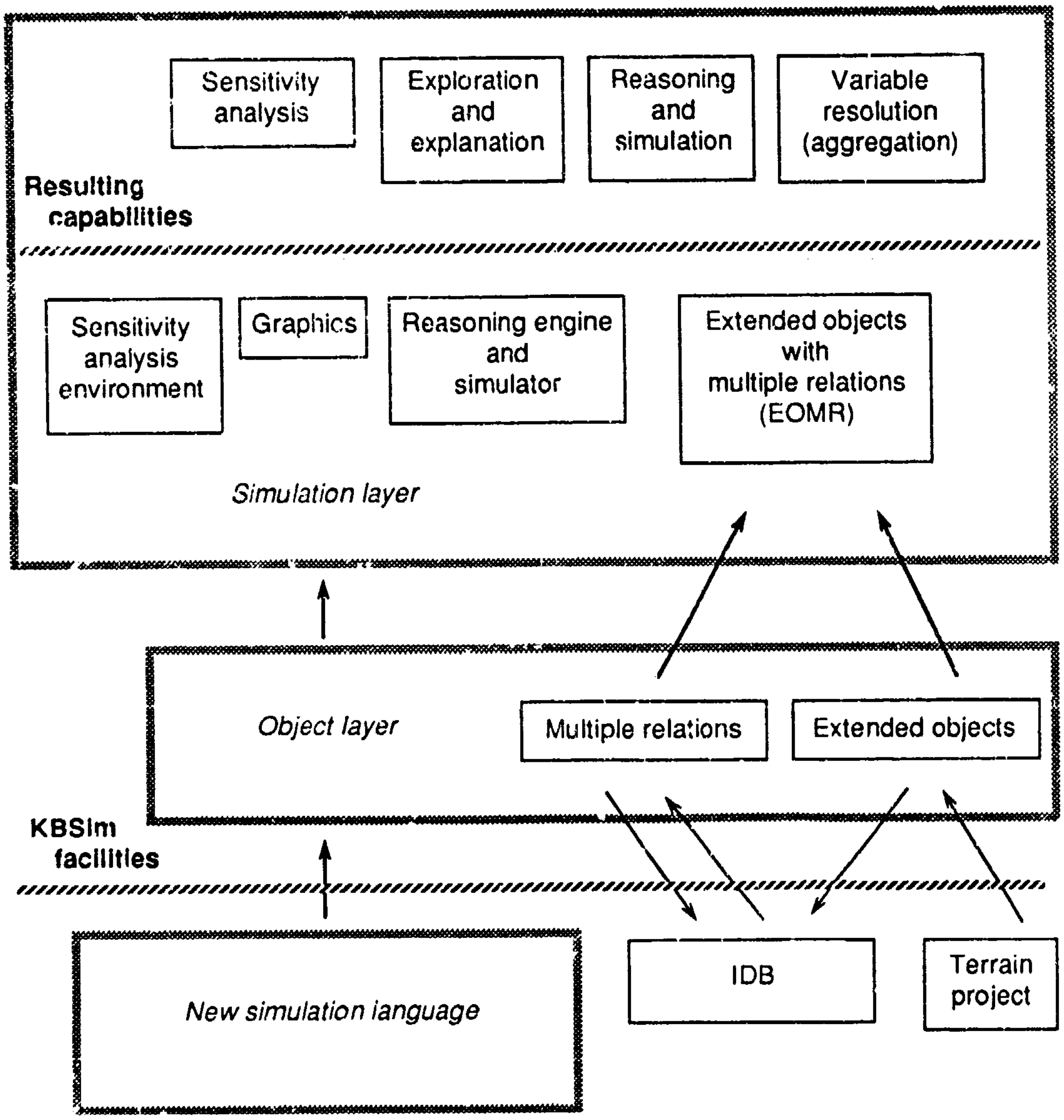


Fig. 6—Eventual internal architecture of KBSim

VI. KBSIM RESEARCH PROGRESS

OVERVIEW

Given the broad objectives of our research, we spent some initial effort defining specific tasks that we felt would have the greatest return and would make the best use of our research skills and interests. We defined the following six tasks:

- Reasoning in simulation
- Multiple relations (and extended objects)
- Highly interactive interfaces
- Sensitivity analysis
- Variation of the aggregation level of a model
- Modeling of soft concepts

Though there is some overlap among these tasks (for example between reasoning and sensitivity analysis and between multiple relations and aggregation), distinguishing them in this way allows us to explore related issues separately, while minimizing the risk of premature integration, as discussed above. In reviewing the literature and interacting with other researchers, we have become convinced that these tasks are central to the production of a next-generation simulation capability. Furthermore, we feel they define a sufficient set of capabilities to produce a meaningful prototype simulation environment; the synergy among them will result in capabilities that provide decisive improvement over available simulation environments. These tasks are derived from our goals as indicated in Figure 7.

The following subsections summarize our progress on each of the above tasks. In addition, we have devoted some effort to examining alternative scenarios for use in demonstrations; this has led to the definition of a river crossing scenario, which is also discussed below.

	Comprehensibility/believability	Extended modeling power	Intelligent exploration and explanation	Model development aids
Reasoning	X	X	X	X
Multiple relations	X	X	X	X
Interactive interfaces	X		X	X
Sensitivity analysis	X	X	X	X
Variable aggregation	X	X	X	X
Soft concepts	X	X		

Fig. 7—KBSim tasks and goals

REASONING IN SIMULATION

As discussed above, traditional 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, effectively asking questions of the form "*What (happens) if . . . ?*" However, it is widely recognized (Davis, Rosenschein, and Shapiro, 1982; Erickson, 1985) that many other kinds of questions are of at least as much importance in many situations. These are questions that might be asked of a human expert in the domain that is being modeled. They include *why* questions ("Why did X happen?" or "Why did object X take action Z?"), *why not* questions ("Why didn't X happen?"), *when* questions ("Under what conditions will event Y happen?"), *how* questions ("How can result R 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?").

Similarly, it is important to be able to ask questions about the simulation state ("At what points in time did condition X hold?") and about explicit causality ("What events can cause event X?", "What events are caused by event Y?", or "Why did event Z *not* occur?"). Finally, there are questions about the model itself ("What constraints govern the occurrence of event X?" or "Under what circumstances can plan Z fail to achieve its purpose?"). We refer to the capabilities needed to answer such questions as **Beyond *"What if... ?"***

For example, the following questions might be asked of an air penetration model:

What caused the penetrator to change its flight plan?
(Why did an event occur?)

Why did the radar *not* detect the penetrator?
*(Why did an event **not** occur?)*

What was the fighter doing when the penetrator crossed the border?
(Treating simulation state as a database and querying it.)

What factors determine whether a radar detects a penetrator?
(Analysis of conditions in causality rules.)

What is the shortest runway a penetrator can land on?
(Analysis of the knowledge associated with a particular object.)

What does a radar do after it has detected a penetrator?
(What event is caused by a given event?)

What can make a penetrator alter its flight plan?
(Which events can cause a given event?)

Can a fighter-base command fighters from another base?
(Analysis of conditions in causality rules.)

If a penetrator has been detected will it necessarily be intercepted?
(Analysis of conditions in causality rules.)

How many times has a given penetrator been detected?
(Analysis of history.)

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. To the extent that these can be represented at all in most systems, they are usually encoded implicitly in behaviors (i.e., specified procedurally) and are therefore not amenable to inference.

The kinds of inference and explanation techniques that have been applied in expert systems provide one way of extending the capabilities of simulation systems. However, simulation imposes additional requirements on these techniques: It requires the ability to represent temporal and spatial constraints and relationships as part of behavior so that inferences can be made on the basis of this information. For example, a given type of airplane may only be suitable for day fighting, and so its behavior in a simulation may be constrained by time of day. Similarly, two units in a simulation may communicate using different media depending on how close they are to each other (spatial constraint), and the choice of medium may determine the delay inherent in their communication (temporal constraint).

Solving these problems requires an ability to reason not only with simulation output, but also with the model itself. This implies that the building blocks of the model must be small and well-defined, and that there must be a powerful deductive mechanism to manipulate them. This requires formalizing precise definitions of certain primitive notions in dynamic systems, including time-varying attributes, events, and causality (i.e., scheduling and unscheduling of events). Our approach has been to use logic for this formalization: We began with a reimplementation of a subset of ROSS in Prolog and subsequently developed a new sublanguage ("DMOD"), implemented in Quintus Prolog.

The main concept in DMOD is an event. An event is said to occur when an "interesting" condition is satisfied in the world. If the sequence of events in the simulation is given, the state of the simulation at any time can be computed. A model therefore consists of two sets of rules:

(a) a set of "causality" rules specifying what other events occur, given that a particular event has occurred (and the sequence of events leading up to it);
(b) a set of "value" rules specifying how the value of a parameter changes after an event, given its value before the event occurred.

The main object being computed in DMOD is the history, or the sequence of events which occur. History is represented as an explicit list of events that have occurred, from which the value rules allow computing the simulation state at any point, on demand. Since many of the computable attributes in a simulation are required only occasionally in any given computation, this "demand processing" (sometimes called "lazy evaluation") is quite efficient; in cases where each computed value of an attribute is accessed relatively often, caching can be used to avoid unnecessary recomputation.

Moreover, histories can be accessed at any point in the simulation. (In contrast, ROSS computes state, from which histories cannot be inferred; objects that need to refer to the past must save past state for themselves, causing a proliferation of attributes.) Since events represent meaningful changes in behavior, histories are a compact representation of behavior, analogous to run-length encoding in data transmission. In addition, chains of causality can easily be inferred from history. Given an event E, it is possible to trace not only what events E led to, but also what events led up to it. This is both a form of explanation and a valuable debugging tool.

DMOD is a formalization of discrete-event simulation, which is something of a departure from the object-oriented approach. However, the declarative nature of DMOD allows a given model to be viewed in different ways for different purposes; for example, object-oriented views can be used wherever they are appropriate, as illustrated in Figure 8. The object view encapsulates those aspects of the model that adhere to objects (i.e., their states and behaviors, their relations to other objects, and the fact that they participate in certain events). The event view encapsulates those aspects of the model that adhere to events (i.e., which objects and attributes they affect, which other events they depend on, and which other events they cause). It should be possible for the designer or user of a model to use whichever view is most appropriate for a given purpose, deriving the information necessary to update the model and produce the complementary view on demand. This ability to derive views (as well as the underlying model itself) from other views and the full integration of object and event views deserve further research.

Discrete-event simulation is based on the observation that to compute the state of a system at any point in time, it is not necessary to keep a record of its state at regular intervals of time. Instead, it is sufficient to keep a record of the *events* that occur in the system; from these and the initial state, the state of the system can be computed at any point in time.

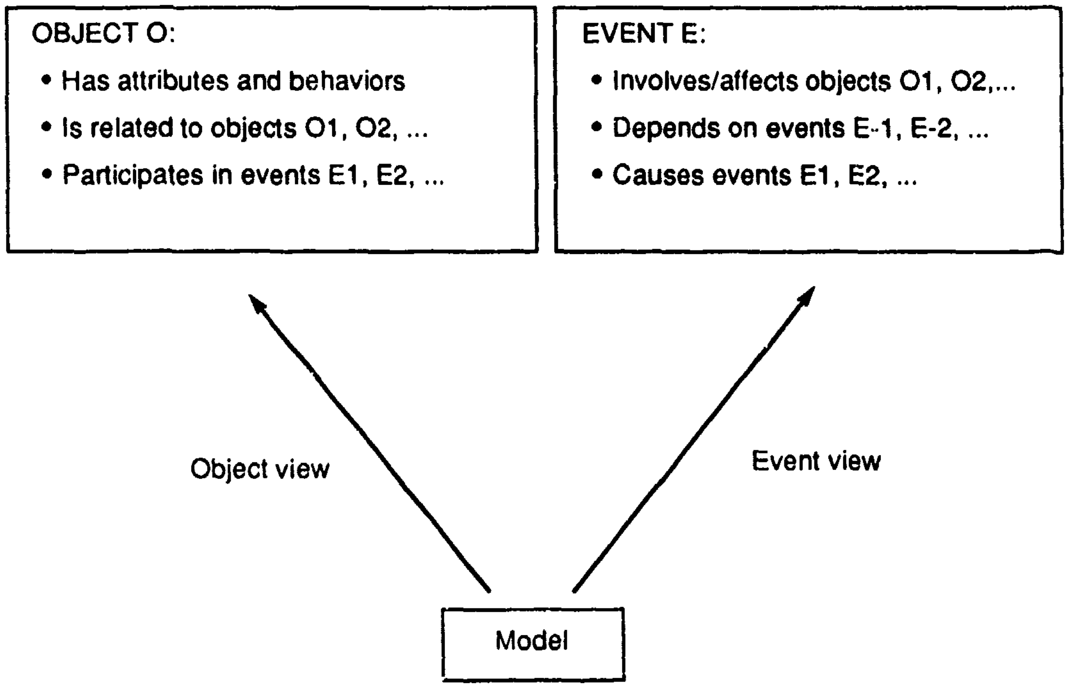

OBJECT O:
• Has attributes and behaviors
• Is related to objects O1, O2, ...
• Participates in events E1, E2, ...
EVENT E:
• Involves/affects objects O1, O2,...
• Depends on events E-1, E-2, ...
• Causes events E1, E2, ...
Object view
Event view
Model

Fig. 8—Multiple views of a model

Models (or programs) in DMOD are statements of logic, so they have a declarative interpretation. However, they can also be executed as Prolog programs, thereby capturing the dynamic aspects of a discrete-event simulation. DMOD currently allows answering certain types of **Beyond *"What if. . . ?"*** questions in a simple air-land model. We are currently expanding this model and are also refining DMOD itself to enable us to answer other, more complex types of questions.

DMOD can currently answer certain special cases of goal-oriented questions of the form "How can X get from A to B?" For example, given an event E and a history, it is possible to trace causality chains backward and forward from E. That is, DMOD can compute what sequence of events led to E, and what sequence of events E led to. Similarly, DMOD's causality rules can be used to determine which events can lead to (cause) a given event, as well as which events follow a given event. To extend this capability, we envision allowing the user to interact with the goal-directed search to give it "advice" about which paths to search. The general goal-directed simulation problem is at least as difficult as the general planning problem in AI (which remains unsolved), and goal-directed simulation may even be harder due to the presence of time; nevertheless, the formalized approach of DMOD provides a good starting point for attacking this problem, at least in its simpler cases.

In addition, DMOD solves two major problems that plague traditional discrete-state simulations. The first is how to simulate decisionmaking that requires reference to past states and events. For example, if a radar detects more than ten penetrators in a span of five minutes, it infers a major attack and informs the central command, otherwise it simply informs its fighter-bases. In most discrete-event simulations, the state is destructively updated at event boundaries. Consequently, relevant past states and events must be explicitly remembered, i.e., made part of the current state. This approach can very quickly increase the size of the current state, making it extremely unwieldy.

In DMOD the main object being computed is not the state, but the history, or the sequence of events that occur. History is a "first-class" object; that is, it has the status of a real data object that can be passed to procedures participating in the simulation. Since the state of the system at any point in time can be computed on demand from the history and the initial state, the problem of accessing past states or events is easily solved.

The second problem is how to ensure that events are consistently unscheduled. For example, if a penetrator takes off from an air base and flies toward a radar, most discrete-event simulations would schedule an event for the detection of the penetrator by the radar at some time in the future. If, however, the penetrator crashes or is diverted or shot down between the current time and this future time, the detection event must not be allowed to occur. To ensure this, the model requires explicit code to unschedule this event. As models grow larger, it becomes increasingly difficult to ensure that such unscheduling is performed consistently. In languages like ROSS, unscheduling is an artifact that must be handled explicitly by the programmer and is a source of numerous bugs.

In DMOD the notion of unscheduling is absent. Instead, when an event is scheduled, a condition is associated with it. When this becomes the first event in the event queue, the condition is evaluated in the light of the history accumulated so far. If the result is true, the event will occur, otherwise it is discarded. As shown below, this is done declaratively, so the programmer need not even be aware of the procedural notion of unscheduling.

We present examples of causality and value rules below (from a simple air penetration model) and show how they are used to perform simulation and how a model built with this formalism can answer questions beyond *"What if. . . ?"* The examples are shown in (nearly) their true Prolog form for legitimacy. This raw DMOD code would be difficult for a modeler to comprehend without a firm grasp of Prolog. However, it is straightforward to define a high-level modeling language that can be compiled easily into DMOD; the appropriate constructs for such a language will become apparent as we gain experience with DMOD. The code shown here is *not* intended to illustrate the ultimate readability of the approach but rather to show that it can be implemented straightforwardly in Prolog.

For readers who are unfamiliar with Prolog, note that variables always begin with capital letters, whereas all other terms are either the names of rules (or procedures) or are uninterpreted literals; for example, the data object *pen(X)* is used to denote a penetrator aircraft whose name will be bound to the variable X. (An underscore appearing in place of a variable name, as in *pen(_)*, represents an unnamed variable that need not be referred to again.) The notation "[X | Y]" denotes a list whose first element is "X" and whose tail is "Y" (corresponding respectively to the LISP "car" and "cdr" of a list). The distinction between a "rule" and a "procedure" in DMOD—as in Prolog—is purely one of interpretation: Both bind their uninstantiated ("output") arguments and return success or failure.

The general form of a *causality* rule (presented in "then-if" form) is:

```
occurs(EventE, HistoryUptoE, EventF, HistoryUptoF) if
   EventE = <some event>,
   EventF = <some future event>,
        <computation of EventF free of references to HistoryUptoF>
   provided(EventE, HistoryUptoE, EventF, HistoryUptoF).
```

where *EventE* is an event that has occurred; *HistoryUptoE* is the history of events up to (but not including) *EventE*; *EventF* is a variable representing a future event caused by *EventE*; and *HistoryUptoF* is a variable representing the history up to (but not including) *EventF*. (All events have time stamps; histories are lists of events sorted in decreasing order on their time stamps.)

Note that *HistoryUptoF* includes both *EventE* and *HistoryUptoE*. The computation of *EventF* does not refer to *HistoryUptoF*; it is based purely on information about *EventE* and *HistoryUptoE*. The condition *provided(EventE, HistoryUptoE, EventF, HistoryUptoF)* is defined by additional rules; it specifies a condition on the time period between *EventE* and *EventF*.

An example of a causality rule is:

```
occurs(EventE, HistoryUptoE, EventF, HistoryUptoF) if
   EventE = flies(pen(X), [Px,Py], TimeT),
   EventF = detects(pen(X), radar(R), FutureTime),
   someRadar(R),
   entersRange(pen(X), [Px,Py], radar(R), FutureTime,
         [EventE | HistoryUptoE]),
   provided(EventE, HistoryUptoE, EventF, HistoryUptoF).
```

This rule says that *EventE* causes *EventF* under suitable conditions. In particular, if *EventE* consists of penetrator X beginning to fly toward position [Px,Py] at *TimeT* (given the sequence of events that has occurred up to *EventE*, denoted *HistoryUptoE*), then (at some *FutureTime*) *EventF* will occur, consisting of radar R detecting penetrator X (under suitable conditions, as discussed below). The procedure *entersRange* computes the time *FutureTime* at which the penetrator's flight path will intersect the radar's coverage, which is the time at which *EventF* will occur.

Of course, if the penetrator is destroyed or diverted from its path between *TimeT* and *FutureTime*, or if the radar is jammed at *FutureTime*, then *EventF* will not occur. These conditions are specified by the "condition predicate" *provided* (appearing as the final argument in the left-hand side of the above rule), whose definition is:

```
provided(EventE, HistoryUptoE, EventF, HistoryUptoF) if
   EventE = flies(pen(X), [Px,Py], TimeT),
   EventF = detects(pen(X), radar(R), FutureTime),
   not occursAfter(EventE, flies(pen(X), _, _), HistoryUptoF),
   not occursAfter(EventE, destroys(_, pen(X), _), HistoryUptoF).
```

This condition predicate is true if all of its conditions are true, namely if *EventE* and *EventF* are the appropriate events, and if penetrator X does not fly elsewhere and is not destroyed between *EventE* and *EventF* (i.e., between *TimeT* and *FutureTime*). For example, the third condition, *not occursAfter(EventE, flies(pen(X), _, _), HistoryUptoF)*, says that it must not be the case that penetrator X flies to some new (unspecified) position after *EventE* in the *HistoryUptoF* (that is, between *EventE* and *EventF*).

The general form of a value rule is:

value(Attribute, Object, Value, History) **if** *<body>*.

stating that the value of this *Attribute* of *Object* is *Value* immediately after the last event in *History* has occurred, where *<body>* is a set of conditions for computing *Value*.

An example of a value rule is:

```
value(velocity, pen(X), [Vx,Vy], History) if
   History = [flies(pen(X), [Px,Py], TimeT) | _],
   value(position, pen(X), [Mx,My], History),
   value(speed, pen(X), Speed, History),
   distance(Px, Py, Mx, My, Hyp),
   Vx is Speed*(Px-Mx)/Hyp,
   Vy is Speed*(Py-My)/Hyp.
```

This rule computes a penetrator's velocity (a state parameter); it binds the variables Vx and Vy to the x and y components of the velocity vector of some penetrator, X. Note that *History* is a list of events that have occurred in reverse chronological order, so that the most recent event is the first one on this list, namely *flies(pen(X), [Px,Py], TimeT)*. The rule therefore says that if this is the most recent event in the *History* (i.e., the penetrator's beginning to fly toward a destination point [Px,Py] at some *TimeT*), then the velocity can be calculated from the penetrator's speed and the distance between its current position [Mx,My] and its destination [Px,Py] (where *distance* is the appropriate function). The penetrator's current position and speed are in turn computed by additional value rules for these attributes; for example, invoking *value(speed, pen(X), Speed, History)* binds the variable *Speed* to the current value of the speed attribute of penetrator X in the given *History*. Initial values for attributes can be specified by separate value rules; for example, the *InitialSpeed* of penetrator X after some *InitialEvent* could be specified by the rule *value(speed, pen(X), InitialSpeed, [InitialEvent])*.

For each event that does *not* change the value of a given attribute, it is conceptually necessary to provide a rule stating that the value of the attribute is unchanged by the event; this is an instance of the so-called "frame problem" (McCarthy and Hayes, 1969). However, this can be accomplished by a single default rule, stating that attributes never change except in the cases specified by their value rules.

Simulation in DMOD consists simply of computing the sequence of events that follow an initial event. To begin a simulation, a list of one or more events is supplied as an initial *EventQ* (written as "[InitialEvent1, ..., InitialEventN]") along with a null initial History (written as "[]"):

```
simulate([], [InitialEvent1, ..., InitialEventN], FinalHistory).
```

There is one additional subtlety which must be considered: In general, causality rules contain *provided* clauses which cannot be evaluated until the entire history up to the future (caused) event is known. To handle this, the rules shown above are transformed slightly, and the actual event queue is made to consist of a list of pairs of events and conditions of the form:

```
cond(EventF, provided(EventE, HistoryUptoE, EventF, HistoryUptoF))
```

which are sorted in increasing time order so that the next event in the queue is the next one that may occur.

The *simulate* procedure, which performs simulation, consists of three rules, each of which will be described in turn.

The first simulation rule is:

```
simulate(History, EventQ, FinalHistory) if
   EventQ = [],
   FinalHistory = History.
```

This rule says that when the event queue is empty, the *FinalHistory* is just the current *History* of the system.

```
simulate(HistoryUptoF,
            [cond(EventF, ConditionF) | RmdrEventQ], FinalHistory) if
   isTrue(ConditionF, HistoryUptoF),
   bag(cond(EventG, Cond),
         HistoryUptoG^
            occurs(EventF, HistoryUptoF, EventG, HistoryUptoG, Cond),
         EventsCausedByF),
   sortEvents(EventsCausedByF, FutureEvents),
   merge(FutureEvents, RmdrEventQ, NewEventQ),
   simulate([EventF | HistoryUptoF], NewEventQ, FinalHistory).
```

This second rule says that the first item in the event queue is a conditional *EventF* with *ConditionF* specified by *cond(EventF, ConditionF)*. If *ConditionF* is true, then find the collection of all events which it may cause. The procedure *bag* produces a list, *EventsCausedByF*, which is the collection of events caused by *EventF* under suitable conditions, i.e., the collection of events such that for each event *EventG* with the condition *cond(EventG, Cond)* there exists some *HistoryUptoG* in which *EventG* follows *EventF*. The resulting list of events (*EventsCausedByF*) is then sorted in increasing time order and merged with the remainder of the events in the event queue, producing a *NewEventQ*. Finally, *EventF* is added to the *HistoryUptoF* (indicating that it has been "performed"), and simulation continues using this new history and the new event queue. For readability, *ConditionF* is tested by the auxiliary rule:

```
isTrue(provided(EventE,HistoryUptoE,EventF,HistoryUptoF),
          HistoryUptoF)
   if provided(EventE, HistoryUptoE, EventF, HistoryUptoF).
```

The final simulation rule is:

```
simulate(HistoryUptoF, [cond(EventF, Cond) | RmdrEventQ],
        FinalHistory) if
   not(isTrue(ConditionF, HistoryUptoF)),
   simulate(HistoryUptoF, RmdrEventQ, FinalHistory).
```

This says that if *ConditionF* associated with *EventF* does not hold, then *EventF* should be discarded and the simulation should be continued using the current history and the remainder of the event queue.

Note that the sorting and merging of events into the *NewEventQ* in the above rule produces a default form of "conflict resolution" among simultaneous events. If two such events are mutually dependent, then the order in which they occur can be critical (for example, if the occurrence of one precludes the occurrence of the other). In such cases, it may be desirable to perform more intelligent conflict resolution, in which simultaneous events are performed in an order derived from the semantics of their interrelationships. Intelligent conflict resolution of this kind can be implemented straightforwardly using the DMOD event formalism, since the semantics of the interrelationships between events is explicit in the causality rules and their associated condition predicates. This is in contrast to strict object-oriented formalisms (such as ROSS), where analyzing the semantics of the interrelationships between events in the event queue is all but impossible.

Techniques for analyzing a model built using this formalism are straightforward and powerful. The simplest forms of analysis consist of tracing event histories and examining attribute values at any point in time. Events can be read as relations, where the relation name (verb) relates the first argument to the remainder of the arguments; for example, *discovers(pen(1), tank(2), 0.9)* can be read as "penetrator-1 *discovers* tank-2 at time 0.9," while *informs(radar(1), ftb(1), pen(1), 4.7)* can be read as "radar-1 *informs* fighter-base-1 about penetrator-1 at time 4.7." In all cases, the final argument is the time at which the event occurred.

For example, a subsequence of the events produced during a simulation might be:

```
flies(pen(1),[-100,100],0.1)                 ; Penetrator-1 flies toward [-100,100].
flies(pen(2),[200,0],0.1)                    ; Penetrator-2 flies toward [200,0].
discovers(pen(1),tank(2),0.9)                ; Penetrator-1 discovers tank-2, and
detects(pen(1),radar(1),4.7)                 ;          is detected by radar-1.
informs(radar(1),ftb(1),pen(1),4.7)          ; Radar-1 informs fighter-base-1,
scrambles(ftb(1),ftr(1),pen(1),4.7)          ;          which scrambles fighter-1.
engages(ftr(1),pen(1),[113.3,0.0],5.8)       ; Fighter-1 engages penetrator-1,
destroys(ftr(1),pen(1),5.8)                  ;          destroys it, and
returnToBase(ftr(1),6.3)                     ;          returns to fighter-base-1.
```

Given a history of events, the value of a parameter can be computed at any point in time by a query such as *valueAtTime(position, pen(2), V, 3.0)*, which would bind the variable V to the value of the position attribute of penetrator-2 at time 3.0.

Similarly, object-oriented views can be generated by showing the value rules for the attributes of a given object and the causality rules that involve the object. For example, the query *initialAttributesOf(radar(X))* might produce:

```
value(getsJammedAtRange, radar(X), 20, [InitialEvent])
value(jammed, radar(X) ,false, [InitialEvent])
value(jammingRecoveryDelay, radar(X), 0, [InitialEvent])
value(radius, radar(X), 40, [InitialEvent])
```

Beyond this, it is straightforward to trace chains of causality. For example, query *forwardCausality(flies(pen(1), [-100,100], 0.1))* would compute the sequence of events caused by the event *flies(pen(1), [-100,100], 0.1)*, which would be:

```
flies(pen(1),[-100,100],0.1)
discovers(pen(1),tank(2),0.9)
detects(pen(1),radar(1),4.7)
informs(radar(1),ftb(1),pen(1),4.7)
scrambles(ftb(1),ftr(1),pen(1),4.7)
engages(ftr(1),pen(1),[113.3,0.0],5.8)
destroys(ftr(1),pen(1),5.8)
```

On the other hand, the query *backwardCausality(destroys(ftr(1), pen(1), 5.8))* would produce the reverse of the above list of events.

Certain dependencies can be traced abstractly in the model itself. For example, the query *affectedBy(velocity, pen(1), E)* would bind E to successive events that affect the *velocity* attribute of penetrator-1, namely, *flies(pen(1), [Px,Py], T)*, and *destroys(ftr(A), pen(1), T)*.

Building a simulation in DMOD (as illustrated by the above examples) encourages the model builder to focus on the relationships among events and between events and attribute values. Demand-update (lazy evaluation) is the normal mode of computing autonomous (as well as other) attribute values, as discussed above. Explicit unplanning of events is unnecessary. DMOD thereby eliminates two of the more troublesome artifacts of object-oriented simulation, while providing a highly declarative representation that makes it straightforward to answer questions beyond "*What if . . . ?*"

The above examples show how DMOD is being used to demonstrate reasoning in an air-land scenario based on an abstraction of the SWIRL and TWIRL simulations. The scenario includes air units (e.g., penetrators, interceptors, radars, fighter-bases) and ground units (e.g., tanks, artillery units, and mechanized regiment headquarters) with communication and direct interaction between them. A graphical interface to DMOD has been implemented, which enables the user to see the state of the modeled system as it evolves over time. We expect to expand this scenario incrementally as we gain experience with these reasoning techniques. DMOD is already changing our thinking about issues of temporal representation appropriate for military simulation and about issues of autonomy and causality. In particular, an event-oriented view of a simulation model has certain advantages over an object-oriented view, at least for some purposes; for example, encapsulating all the state-changing side effects of an event *as part of the event* makes it much easier to comprehend and maintain them than if they are distributed among the objects that own the state (as they would be in a strict object-oriented approach). We have therefore begun to think of a model as a database that can be *viewed* from different perspectives for different purposes (e.g., event-oriented or object-oriented).

DMOD is only a first step toward the use of reasoning in simulation. Many issues remain to be explored, including the further integration of event-oriented and object-oriented approaches, the handling of stochastic behavior, the implementation of more-intelligent conflict-resolution strategies, and ways of taking advice from the user when attempting to answer goal-oriented queries. Nevertheless, we feel that the leverage already provided by DMOD indicates that a logic-based approach to simulation is an excellent way of answering questions that go beyond "*What if . . . ?*"

MULTIPLE RELATIONS (AND EXTENDED OBJECTS)

Complex simulations require the representation of multi-dimensional relationships among objects. For example a tank *is-a* kind of moving object, *is-a-part-of* a particular tank battalion, may be *under-the-command-of* a particular "crossing area commander," may be *in-communication-with* some set of other objects, and may be *near* a (possibly different) set of objects. It is important for analysts to be able to define such relations freely, examine the state of the simulation in terms of these relations, and modify them dynamically. Traditional object-oriented systems—as well as most semantic nets, frame systems, and expert system shells, with some exceptions (Carnegie Group, Inc., 1986)—provide strong support only for the *class-subclass* relation (also called *IS-A* and *taxonomy*). A corresponding *inheritance* mechanism is usually supplied to maintain taxonomic relationships (serving as a specialized kind of inference), but little or no support is provided for other kinds of relations. In fact, the *IS-A* relation has been pressed into service for many inconsistent purposes (Brachman, 1983), though it is poorly suited to many of them.

Recent work in integrating relations into an object-oriented language (Rumbaugh, 1987) appears to ignore the issue of "inferential support" for relations. Some authors argue that the *IS-A* relation should be thought of as a programming (or implementation) construct rather than a semantic modeling construct, while other relations should be accorded inferior status (Cox, 1988). Our own approach is that, while implementation relations may be important, the primary responsibility of any modeling environment is to provide modeling constructs that allow representing features and phenomena of interest in the real world in natural ways (that is, to allow *modeling*). We consider multiple relations as alternative views of a model (analogous to the object and event views discussed above), which are necessary to provide natural ways of modeling alternative features and phenomena. For this reason, we feel it is important to provide a true multiple relation environment, in which different kinds of relations are supported by appropriate specialized inference mechanisms.

It is also important to note that a number of real-world entities are difficult to represent as traditional objects. For example, terrain, roads, rivers, weather, and electromagnetic fields defy easy representation by conventional object-oriented means. These "extended" objects or phenomena require representations and manipulations that are different from those used for more compact objects, either because they traverse or interpenetrate other objects (without actually being "part" of them) or because they are best described by continuous models (such as partial differential equations).

Though investigating these representation issues is not among our primary research tasks, we expect to confront some of them by necessity and to incorporate the results of research in this area by other projects at RAND. We recognize the need to integrate these issues with those of representing multiple relations among objects; that is, the relations we are considering must be applicable to "extended" objects as well as more traditional ones. We refer to the combination of these ideas as *extended objects with multiple relations (EOMR).* We believe it will be possible to encapsulate extended objects (however they are actually implemented) so that they appear as ordinary objects from the perspective of the multiple relation facility, though the inferential support mechanisms for relations on extended objects will in general depend on the semantics—if not the implementations—of these objects.

We have identified and characterized a number of important types of relations in military simulations, including (among others):

class-subclass (trucks are a subclass of moving objects)
part-whole (a battalion is a part of a brigade)
command (a unit is commanded by a commander)
connectivity (two units are in communication with each other)
proximity (two units are near each other geographically)

We have identified inference mechanisms to support these relations by analogy to the way inheritance supports the class-subclass relation. That is, for each type of relation there is some special kind of inference which is appropriate to it. In the case of a *class-subclass* relation, this inference is inheritance. However, in the case of a *part-whole* relation (for example) the appropriate inference mechanism involves distributing the values of attributes of the whole over its parts, so that the number of troops in a brigade always equals the sum of the troops in its battalions (this special case is discussed under aggregation below). In the case of a connectivity relation, on the other hand, the appropriate "inferential support" involves some form of transitivity so that if A is in communication with B and B is in communication with C, then A is (at least indirectly) in communication with C.

This is only the tip of the multiple relation iceberg: Once we have implemented multiple relations and multiple inference mechanisms to support them, we hope to be in a position to take a more general approach to relations. This should involve defining a framework of general characteristics (i.e., "attributes") of relations to allow defining relations in terms of these characteristics (Carnegie Group, Inc., 1986). For example, a relation can be defined in terms of whether it is one-to-one, one-to-many, many-to-many, onto, transitive, associative, reflexive, invertible, etc. This would allow users to define relations simply by indicating their appropriate attributes; appropriate inferential support mechanisms could then be generated automatically, at least in feasible cases.

Though these issues are important, we have temporarily suspended our work on multiple relations (due to resource limitations) pending the results of work currently being conducted by the IDB project, which is researching the way analysts use relations in real-world databases and the availability of commercial object-oriented database management systems. We are continuing to coordinate with the IDB project to define a representation of multiple relations that will satisfy the needs of both our proposed simulation environment and the IDB project's proposed object-oriented database.

HIGHLY INTERACTIVE INTERFACES

In the analytic military simulation domain, a single analyst is typically the designer, user, and ultimate consumer of the results of a simulation and is constantly evolving and refining the simulation even while it is being run. It is important that the analyst be able to understand what the simulation does and why, be convinced that it is doing the right thing, and be able to modify its behavior. This is the motivation for what we call *intelligent exploration and explanation*. One of the keys to this is the user's interface with the simulation. This must be interactive and lucid, taking full advantage of modern graphics, without falling prey to "the fetish of realism" (i.e., realism for its own sake).

In addition, since we perceive that multiple relations are ubiquitous in simulations, we feel it is important to allow a user to display relations in a perspicuous form (as an analyst would naturally draw them) and to edit them graphically. Previous work on the automated display of diagrams (Vaucher, 1980; Reingold and Tilford, 1981) and on allowing users to define relations graphically (Bryce and Hull, 1986) has largely ignored the problem of inferring the semantics of relations directly from drawn input.

We have defined and begun implementing a highly interactive graphics environment that emphasizes the ease of manipulating simulation objects, minimizes redundant display updating, allows animation of sequences of events (such as causal chains), and eliminates graphic artifacts that have plagued previous simulations (such as ghost images of objects after they have been destroyed in the simulation).

We have currently implemented this environment in Franz LISP as a "Graphics in LISP" package (GIL) that uses our Hose facility (Steeb et al., 1986) for communication between LISP and C to access the GKS graphics routines for our SUN workstations. (SunCORE was chosen initially as the most stable of the graphics packages for the SUN, but we later ported GIL to GKS when it became clear that GKS was a better standard than CORE.) GIL encapsulates the low-level graphics package, so that switching to a different graphics standard in the future (e.g., PHIGS, X-windows, or NeWS) should not be difficult. The use of graphics standards provides device-independence and the flexibility of developing software on both black-and-white and color SUNs. Though there appears to be a heavy performance penalty for using GKS, we are evaluating whether the resulting portability is worth this price and are trying to improve the graphics performance of our current environment. We are also experimenting with Graphics in Prolog, using Quintus Prolog's ProWindows package.

In our attempt to separate the machinery of the simulation environment from the model of interest (as discussed in Section IV above), we have eliminated the need of previous object-oriented ROSS simulations to update positions of moving objects explicitly in simulation code. This is a source of both obfuscation and potential bugs in existing simulations, since it is necessary for the programmer to remember to update the positions of objects whenever they might be affected. We have designed and implemented a *demand update* strategy for displaying the results of a simulation by computing graphic attributes of objects only when necessary. This form of "lazy evaluation" automatically updates the images of objects on the display at appropriate times and minimizes the redundant update of graphic attributes as well as minimizing graphic output. This and a number of other significant extensions to the ROSS language have produced an interim language (referred to informally as "XROSS"), which is described in Cammarata, Gates, and Rothenberg (1988).

We began investigating the display and editing of decision tree diagrams, motivated by the use of such trees by the RSAC project at RAND (Davis, Bankes, and Kahan, 1986). Our intent was to define a generic diagram editor analogous to a syntax-directed program editor. This would allow the definition of transformation commands with which the user could create, delete, or modify logical entities such as boxes, bubbles, and links. Each command would cause both logical and graphical actions that would simultaneously maintain a consistent logical representation and graphic image of the intended diagram. Where a logical structure allowed alternative graphical relationships among pictorial elements, the diagram editor would choose a default, but the user would be allowed to modify this choice to enhance readability (for example, moving "more important" boxes to the top of a picture).

Though we consider this work to be important, we have postponed it in favor of more urgent graphics tasks. We also hope to experiment with direct manipulation of simulation objects, which we have called "graphic behavior modification." For example, an analyst might define an avoidance maneuver for an airplane (e.g., when it is attacked by a missile) by manipulating the simulation picture of the airplane graphically (e.g., with a mouse), moving it through a path that defines the desired avoidance response. The system would generalize this behavior appropriately (taking into account directional symmetries, etc.) so that a missile coming from a different direction would cause the appropriate response.

For the immediate future, we expect to continue developing our current graphics environment, integrating it with the reasoning techniques being explored with DMOD to perform intelligent exploration and explanation. Whereas expert systems typically perform explanation by displaying chains of inference (i.e., which rules fired to produce which results), our emphasis will be on displaying sequences of graphical states in animated form in those cases where the behavior of the simulation is displayable (such as the geographic movement of troops in a river crossing scenario).

SENSITIVITY ANALYSIS

In all but the most trivial situations, it is impossible to run more than a tiny fraction of all the potentially relevant cases of a simulation. Even if cases are generated and run automatically, the computation time required is often prohibitive, and the job of analyzing the results is monumental. It is important to be able to analyze the behavior of a model in more powerful ways, such as by performing sensitivity analysis on its parameters.

A simulation can be viewed as the computation of a single top-level function involving hundreds or even thousands of parameters. Sensitivity analysis attempts to show how sensitive the results of the simulation are to variations in those parameters that are of interest in a given situation. This is especially important for promoting confidence in the stability of the model (i.e., knowing that its results are independent of minor changes to its parameters) and for indicating which parameter values are the most important ones to validate (by real-world means) in order to make the model believable.

The naive approach to sensitivity analysis requires running a simulation many times while perturbing individual parameters to see how the results differ. We refer to this approach as "naive perturbation." This is prohibitive in most cases, which is why sensitivity analysis is rarely performed. The intent of our research is to provide a computationally feasible way of performing sensitivity analysis in a simulation environment.

We have designed a new *propagative* approach to sensitivity analysis that propagates and combines the sensitivities of functions through a computation. This approach is motivated by the chain rule of the differential calculus, which defines the partial derivative of a composite function as a combination of the partial derivatives of its subfunctions[3] (assuming these subfunctions are differentiable with respect to the parameters of interest). Viewing a simulation as a top level function that invokes many levels of nested subfunctions, each of which is called many times (as illustrated in Figure 9), the naive approach to perturbing top level parameters executes the top level function one or more times for each parameter, each time executing each nested function. That is, each nested function is executed a number of times proportional to the number of parameters of the simulation (where the constant of proportionality is the number of times a single parameter must be perturbed in order to approximate a partial derivative). Our propagative approach instead computes a representation of the sensitivity of each nested function (i.e., an approximation to its partial derivatives) the first time it is executed (e.g., by perturbing it once for each of its own parameters) and propagates that sensitivity information through the computation rather than having to recompute it each time it is needed. Since most subfunctions have many fewer parameters than the simulation as a whole, this approach results in their

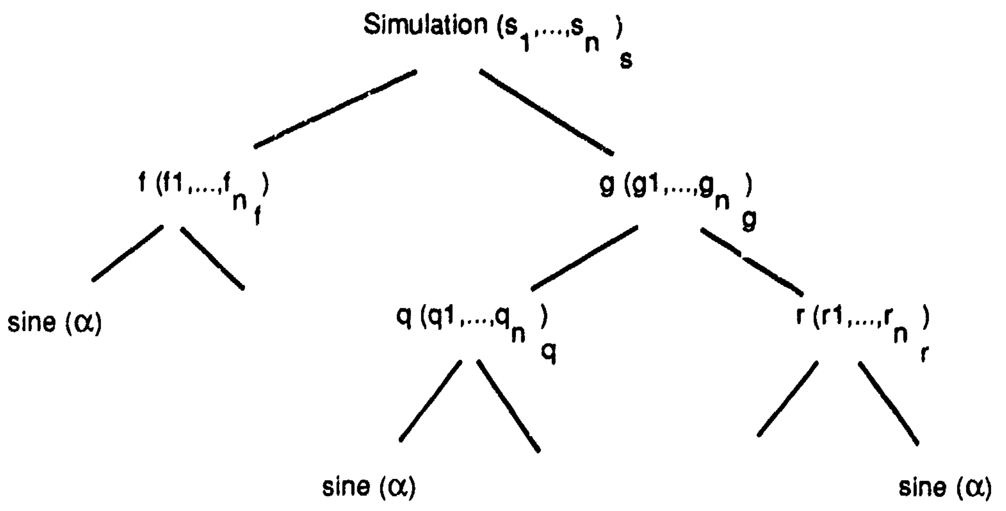


Fig. 9—Simulation as a high-level function

[3]In this discussion, we use the term "subfunction" to mean a function that is called by another function, rather than one that is defined within the lexical scope of another.

being evaluated fewer times, thereby avoiding much of the cost of sensitivity analysis. (Subfunctions that have as many or more parameters than their callers are simply evaluated as in naive perturbation.) Note that, for simplicity, this discussion focuses on sensitivity analysis in which one parameter at a time is varied; however, the propagative approach applies equally well (and has even greater potential "payoff") when combinations of parameters are varied together (i.e., when higher-order partial derivatives are required).

We first implemented a program to analyze the expected payoff from this propagative approach; this program performed a symbolic analysis of a computation in order to see where the new approach would offer an advantage. We encountered performance difficulties in applying this payoff analysis program to any but the simplest functions, and we soon realized that the payoff analysis was even more computationally intensive than the sensitivity analysis it was attempting to analyze! The simple functions that we managed to analyze with this payoff analysis program confirmed our hypothesis about the advantages of the new approach, yet it became clear that it would be more efficient to analyze the payoff by actually implementing a propagative environment and using it to perform sensitivity analysis while collecting performance statistics. We therefore next implemented a novel stand-alone computational environment in LISP to support the propagation and combination of sensitivities. This environment has allowed us to try our approach on a number of computations and to analyze its payoff.

Not all functions in a given computation are of equal interest for sensitivity analysis. Further, it may not make sense to analyze the sensitivity of built-in functions (like the conditional function "cond" in LISP). Our computational environment therefore allows the user to designate certain "candidate" functions as those to be analyzed; these same functions are also instrumented by the environment to keep track of such things as how often they are called. The user further divides candidate functions into two groups depending on whether their return values are discrete or continuous. Each candidate function is then modified by the environment so that (among other things) it returns not only its usual return value but also a representation of its sensitivity to each of its arguments.

The sensitivity of a function is considered to be (an approximation to) the collection of partial derivatives of the function with respect to its arguments. To compute this, whenever a candidate function is called normally during a computation (referred to as a "primary" call), the environment first causes the function to compute its normal return value and then causes it to place recursive "secondary" calls to itself, perturbing each of its arguments in turn. By so doing, the function computes its sensitivity and returns this information to its caller (along with its normal return value). During these secondary invocations, the function places secondary calls to its candidate subfunctions; this approximates the return values of these candidate functions, rather than recomputing them for each secondary call. The approximation technique used is to apply the sensitivity of each called function as a linear approximation of its value (where the sensitivity of each called function is computed recursively by this same process and returned to its caller). This process hinges on the notions of primary and secondary calls, which we illustrate with a simple example, shown in Figure 10.

Consider a top-level function H that calls a candidate function F, which in turn calls another candidate function G. For simplicity, suppose that F calls no other candidate functions besides G and that G calls no candidate functions at all. Further, suppose both F and G are continuous-valued, differentiable functions. Since F is a candidate function, the propagative environment automatically interprets H's call to F as a *primary call* (shown as "p-call" in the figure). Upon receiving this primary call, F initializes its "approximation table" and proceeds to compute its normal return value, in the course of which F calls G. Since G is also a candidate function, the environment interprets this as a primary call as well. Upon receiving this primary call, G proceeds to compute its normal return value (since G calls no other candidate functions, it will have an empty approximation table). Having computed its value (and before returning from its primary call), G must compute its sensitivity (to be returned to F along with G's value, as the result of F's primary call to G).

To compute its sensitivity, G perturbs each of its own arguments in turn, placing a recursive *secondary call* (shown as "s-call") to itself for each perturbation. For example, if G has only a single formal argument, whose supplied value (in F's primary call to G) is X, then G will place a secondary call to itself with an argument of $(X+\Delta)$, where Δ provides the perturbation. This recursive secondary call to G returns the value $G(X+\Delta)$ to the primary invocation of G. Assuming, for simplicity of exposition, that a single perturbation is enough to

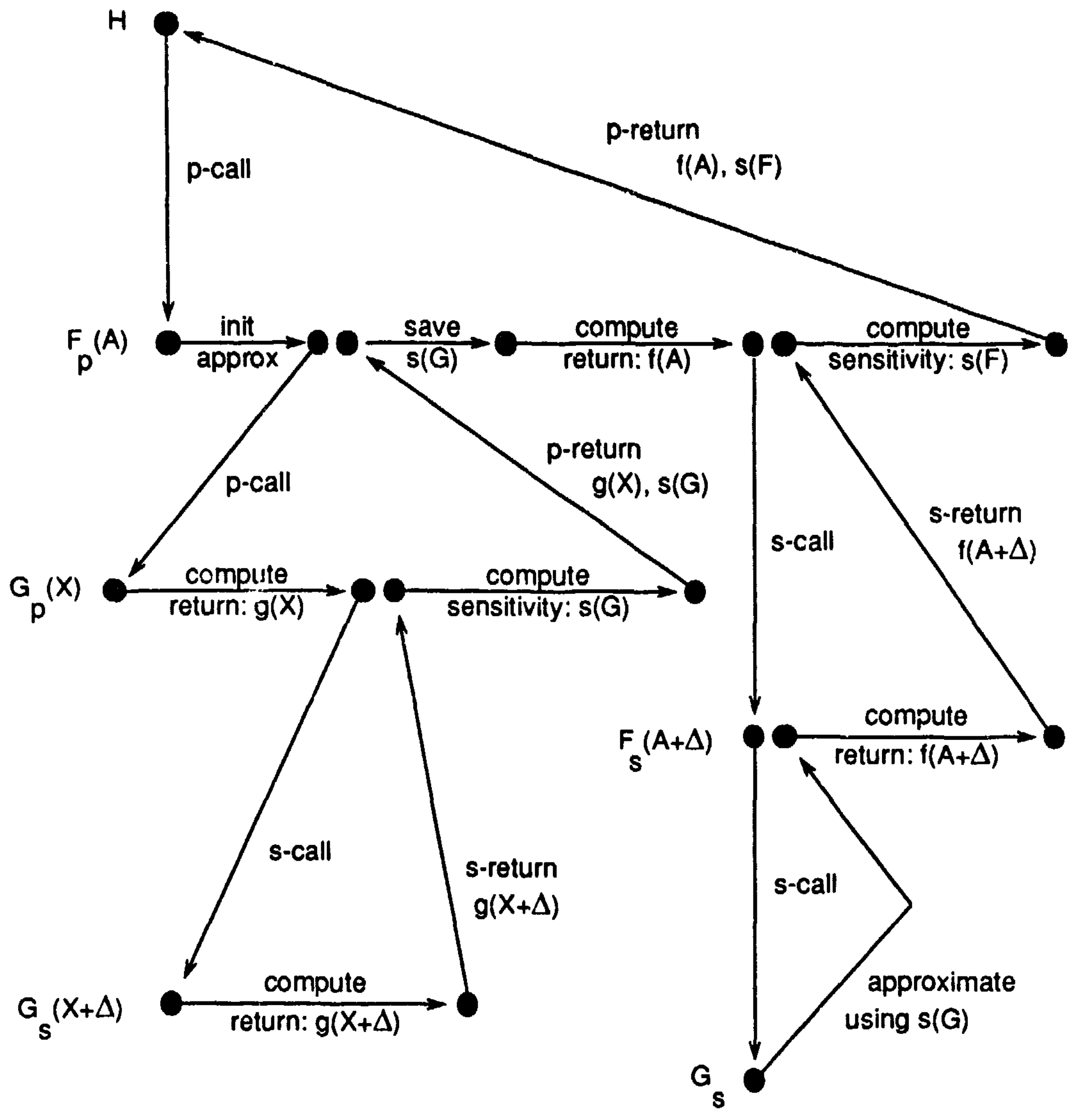


Fig. 10—Propagative sensitivity analysis

produce a linear approximation to its "partial" derivative, G computes the Cauchy ratio (G(X+Δ) - G(X))/Δ as its sensitivity information (shown as "s(G)"), and returns this to F along with G's normal return value (shown as "g(X)").

The value of the Cauchy ratio represents the sensitivity of G to its argument (in the neighborhood of the point X); it can be interpreted geometrically as the slope of the tangent to the function G at X or algebraically as a coefficient *t* such that G(X+Δ) = G(X) + *t* * Δ. This latter interpretation is used subsequently by F as a linear approximation to the value of G. In the more general case, where G had several arguments, it would perturb each one, generating secondary calls to itself to compute a Cauchy ratio for each partial derivative, and would return the collection of the resulting coefficients as its sensitivity information.

When the primary call to G returns to F, F separates the return value g(X) from the sensitivity information for G, which F stores as its approximation table entry for this invocation of G. F continues executing its own primary call (from H), using the value g(X) as needed to compute its own return value. Having computed its value (and before returning from its primary call), F computes its sensitivity (to be returned to H along with F's value, as the result of H's primary call to F). To compute its sensitivity, F perturbs each of its own arguments in turn, placing a recursive secondary call to itself for each perturbation. The results of these recursive secondary calls are used to compute the Cauchy ratios for F's own partials, as was done for G. However, during these secondary calls, whenever F would normally call G, it now places secondary calls to G, using its approximation table entry for G to approximate the value of G. This is the essence of the computational payoff of the propagative scheme.

To summarize, a *primary call* to a candidate function calculates its normal return value, in the course of which it places primary calls to any candidate functions it calls. The primary call then initiates a series of recursive secondary calls by the function to itself, to compute its sensitivity by perturbing its parameters. In addition, the primary call causes the function to initialize an "approximation table" for approximating the return values of those candidate functions that it calls during its recursive secondary invocations. It is the ability to approximate these return values (rather than recomputing them) that allows the propagative approach to outperform naive perturbation. Each call to a candidate function G by a given function F will generate an entry in F's approximation table. This entry will contain sensitivity information (about the called function G) that was returned to F by G (in response to F's primary call to G). This sensitivity information (which is a representation of G's partial derivatives) serves as a linear approximation for the value of the called function G. When F has completed perturbing its parameters via recursive secondary calls, it will have derived its own sensitivity to its parameters: This information is returned by F to its caller to serve as F's entry in its caller's approximation table.

A *secondary call* by a function F to a candidate function G essentially replaces the evaluation of G with an approximation, using the entry for G in F's approximation table. This entry was returned to F by G itself when F placed its primary call to G.

The environment we have implemented (in LISP) requires that the user declare which functions in a computation are candidate functions and which are discrete. It arranges for primary and secondary calls, as necessary and for the computation and propagation of sensitivity information.

Our initial results indicate that this propagative approach has tremendous potential, reducing a combinatorial process to a linear one; however, additional research is needed before the approach can be integrated into our simulation environment. For example, although Boolean derivatives and a Boolean version of the chain rule can be defined (Blanning, 1987), the general case of symbolic-valued functions requires further thought. Our computational environment allows functions of this sort, but only applies the propagative approach to those functions that are differentiable in the usual sense, performing "naive" perturbation for all others. However, even without such extension, we expect this new approach to have a major impact on the feasibility of sensitivity analysis in simulation.

VARYING THE LEVEL OF AGGREGATION OF A MODEL

Current simulation models cannot vary the level at which they are aggregated: The desired level of aggregation is chosen in advance and the simulation is designed for this level. Changing this level typically requires considerable reprogramming of the simulation; changing it under user control or dynamically is generally unthinkable. Structured (or "composite") objects are poorly supported by traditional object-oriented environments. Only *ad hoc* mechanisms exist for representing part-whole relations (such as the fact that a brigade consists of several battalions), and there are no automatic facilities for maintaining attributes that are the sums of attributes of the parts of an object. Similarly, only the lowest-level objects in the hierarchy are expected to receive and react to messages (i.e., interact with other objects) in a simulation. These limitations make it difficult to represent even static, uniform aggregation (in which all peer objects, such as brigades, are aggregated to the same level throughout the simulation), since objects at various levels of the hierarchy must maintain attributes representing summary information about their parts and must be able to interact with objects at their peer level.

Dynamic aggregation compounds these problems by requiring the ability to switch levels at runtime. For example, an aggregated object (such as a brigade) that is asked to disaggregate itself would be required to generate subordinate objects (such as battalions) obeying reasonable constraints for how the attributes of the whole should be distributed among these parts. Nonuniform aggregation allows the interactions among objects to cross peer levels, so that, for example, an aggregated brigade in one sector can interact with disaggregated battalions in another sector. This allows "focusing the attention" of a simulation by disaggregating only those areas or aspects that are of particular interest. Dynamic, nonuniform aggregation requires the ability to reroute interactions to the appropriate levels dynamically.

We do *not* yet envision simulations that can be automatically aggregated to any arbitrary level (without a prior modeling effort at that level), since that would amount to an "automatic programming" environment. Instead, we restrict our attention to models in which a fixed set of possible levels of aggregation have been defined in advance (when building the model). That is, we expect dynamic aggregation to be performed on a model that has been constructed with aggregation in mind: The model will actually consist of a suite of coordinated models at various potential levels of aggregation.

Note that if a simulation is designed at the most disaggregated level that might ever be desired, aggregated results can be computed after the simulation is run; we refer to this (somewhat arbitrarily) as "abstraction" to distinguish it from aggregation. Unfortunately, running a highly disaggregated, detailed model may be unwarranted (i.e., unaffordable) in many cases. Furthermore, abstraction does not allow the user to interact with the simulation at different levels of aggregation (providing different "views" of the model) and is therefore a poor substitute for dynamically variable aggregation.

The fact that the level of aggregation of a model gets "frozen in" early in its design is a major impediment to the reusability of model d the utility of simulation in general. Users should be able to vary the level of aggregation (or 'resolution") of a simulation and to indicate which aspects of the model are of particular interest, running those aspects of the simulation disaggregated while running peripheral aspects at higher levels of aggregation. This goal has been addressed only in very limited contexts at a theoretical level (Fishwick, 1986). We are developing te hniques for building simulations whose level of aggregation can be varied both statically and dynamically by representing "vertical slices" of objects in an aggregation hierarchy (for example, divisions, brigades, battalions) and allowing interactions between objects at different levels of aggregation.

We have taken an object-oriented approach to representing aggregation, in which objects are used to represent each of the levels of aggregation. We associate behaviors and attributes with *composite* objects that simulate the behavior of their subordinates and maintain attributes representing aggregations of the attributes of those subordinates. We have successfully prototyped a scenario fragment using this approach. The fragment consists of a military company made up of several plat s; the simulation is plan-driven to provide a framework for coordinatin the actions of units at various levels. Various aggregation and disaggregation functions are defined for switching among aggregation le ls.

Our results to date indicate that dynamic aggregation is feasible, so long as the multilevel model is developed with certain consistency criteria held firmly in mind; this can be viewed as a semiformal aid to developing coordinated, consistent multilevel models, which has heretofore been akin to alchemy.

Our prototype was coded in the interim language "XROSS," implemented on top of ROSS in Franz LISP. Unfortunately, the XROSS extensions (as well as those that implemented dynamic aggregation) could only be run interpretively, resulting in a severe performance penalty. We are currently recoding both XROSS and dynamic aggregation in Common LISP to permit compilation. Once this is accomplished, we intend to expand the protot into a more realistic demonstration of our approach.

MODELING "SOFT" CONCEPTS

Policy analysts often use simulations to try to answer qualitative questions involving intangible, "soft" concepts: factors that appear to be important but are hard to quantify. Some examples are troop morale breakdown following heavy losses, the importance of momentum in a hasty attack, or the effects of panic in chemical or biological warfare. Such factors can have major impacts on the course and outcome of a situation, but they are extremely difficult to model with metrics such as communication delays, effective force ratios, or other directly quantitative criteria.

These "soft" concepts (or "constructs") can best be modeled by analyzing their underlying or contributing factors and subjectively aggregating these to arrive at an estimate of the construct itself. For example, morale can be considered to be composed of factors such as sleep deprivation, hunger, casualty level, weather, and bombardment level; these factors can then be aggregated to derive an estimate of morale. Because of the complexity of this process, such qualitative factors are rarely introduced into military simulations (Davis and Winnefeld, 1983), and then only at a surface level.

To investigate building models that incorporate qualitative notions of this kind, we initially focused on the concept of "initiative" as used by military analysts. The following quotations motivate this choice:

> The Army's AirLand Battle Doctrine is based on using initiative, depth, agility, and synchronization to gain the advantage against numerically superior forces. . . . In essence, if a commander can decide and execute more quickly than the opposition, the objective of achieving the initiative will be assured.
>
> — *DARPA (1986)*

> The stereotypical Soviet military leader is seen in the West as being prevented from exercising initiative on the battlefield and thus unable to take full advantage of opportunities that may come his way.
>
> — *Armstrong (1984)*

Quotations like these suggest building a simulation model that reflects what the term "initiative" means to analysts, so that we can submit statements like the above to experimentation within a simulation. Analysts seem to use the term "initiative" to mean the ability to consider a wide range of options and act with great flexibility in the absence of direct instruction. Furthermore, many analysts assume that Blue force commanders possess greater initiative than their Red force counterparts; from this assumption, analysts conclude that Blue forces would be more effective than simple force ratios would suggest. This conclusion (though *not* the assumption on which it is based) could be verified if an appropriate model of initiative were built into the representations of commanders in a simulation.

We have researched this concept in the literature and among RAND analysts. We have also identified a methodology for acquiring analysts' models of such concepts; this involves a blend of expert-systems knowledge engineering and an algebraic modeling technique developed at RAND for use in judgment research (Veit, Callero, and Rose, 1984). (This methodology produces an algebraic model showing the quantitative relationship between a construct such as initiative and the component concepts and attributes that comprise it.) Based on this research, we have proposed a number of approaches to modeling initiative. For example, to the extent that initiative consists of having greater freedom of choice in making decisions, it might be modeled by a simulation of a commander as an inferencing process having variable inference capabilities or variable access to relevant data for making inferences. Additional thoughts on this subject are presented in the appendix.

As we explored these ideas, we became convinced that our initial goal was too narrow. To model "soft" concepts, it is necessary to model commanders; but this requires nothing short of modeling the decisionmaking process as a whole and building simulations that are driven by plans (just as the real-world actions of commanders are based on the plans they formulate or receive from their commanders). Modeling decisionmaking is also central to modeling command and control issues, which are of great current interest in the modeling community. There is no question that this is a worthy area for research; however, we do not feel we currently have the resources to do it justice. We have therefore decided to postpone further work on this task until we can reexamine it in this expanded context.

THE RIVER CROSSING SCENARIO

We originally planned to explore, develop, and integrate knowledge-based simulation in a realistic battle management application. This was to be done in an evolutionary manner, beginning with an expansion of the TWIRL (Tactical Warfare in the ROSS Language) scenario used originally in the development of ROSS. TWIRL is based on a hasty river crossing operation by Red forces against a Blue hasty defense. The crossing operation is a maneuver involving combined arms teams (infantry, artillery, armor, electronic combat, etc.) on both sides, with coordinated actions and critical timing requirements. The maneuvers are primarily the responsibility of the division commander, but many decisions are made at the regiment, battalion, and company levels. This would have facilitated the examination of multiple levels of aggregation, representation of different relations among units, and exploration of issues of initiative.

We expected to add a number of enhancements to the TWIRL scenario, including a detailed terrain database, line-of-sight and mobility calculations, rules for coordination of forces, and details of the actual river crossing itself (the TWIRL simulation concentrates on the advance to the river, and models only high-level actions). Our plan was to integrate this augmented TWIRL scenario into a larger tactical warfare scenario, drawing on work being performed in other projects at RAND. This larger scenario would add complexity, scope, and realism to the effort, as well as providing a bridge between several RAND projects.

However, we modified this initial plan for several reasons. First, we began to feel that the river crossing was too ground-specific and did not reflect current concern with integrated air-land operations. Second, we realized that since detailed scenario development of this kind was already being pursued by other projects at RAND, continued work in this area on our part would be redundant. Finally, we realized that the modeling techniques we are exploring do not require the detail of an elaborate scenario but are more appropriately demonstrated within simplified scenarios that are abstracted from an integrated air-land scenario. We have accordingly focused our attention on building simpler demonstration scenarios (such as the one illustrated above in Section VI) that exhibit the phenomena and degree of complexity relevant to our new techniques. As these techniques are shown to be worthwhile, we will seek to apply them to real, ongoing analysis studies, rather than attempting this prematurely.

VII. CONCLUSIONS AND FUTURE DIRECTIONS

Knowledge-based simulation attempts to bring together the best ideas of object-oriented simulation, AI, and interactive graphics to produce a new kind of modeling environment. The single most important goal of this effort is to improve the *comprehensibility* of the modeling process.

One of the most common complaints among military analysts is the incomprehensibility of the models available to them. Incomprehensibility leads to both a software engineering problem (an incomprehensible program is unlikely to be *correct)* and a modeling problem (an incomprehensible model is unlikely to be *valid*). Both the model and the program that implements it must be comprehensible in order to have any confidence that the program correctly implements a valid model. The KBSim project described in this Note has taken several crucial steps toward improving the comprehensibility of simulations, as summarized here.

The use of explicit, human-meaningful knowledge in the specification of a model makes it directly comprehensible. To the extent that the modeling environment can draw inferences from this knowledge using comprehensible procedures (for example, by running a simulation), the entire process becomes comprehensible to the modeler and user. Our use of DMOD is an attempt to represent knowledge and inference procedures in a way that greatly improves the comprehensibility of a simulation, while still being computationally efficient. The focus on objects in object-oriented simulation has neglected the *event* as an entity in its own right. The event view encapsulates the causes, effects, and side effects of events in much the same way that the object view encapsulates the attributes and behaviors of objects. We provide both views in DMOD, since each has significant advantages for some purposes.

Comprehensibility must not be thought of as a purely static quality: It is equally important that the user of a model be able to comprehend the behavior and dynamics of the phenomena being modeled. We have broadened the traditional ("toy duck") view of simulation to allow the user to ask questions of the model that go beyond "*What if . . . ?*" In addition, providing the user with capabilities for stopping, querying, backing up, and rerunning a simulation—as well as explanatory capabilities such as showing causal chains—adds a new dimension to the comprehensibility of the dynamics of a model.

Our work in XROSS (as well as DMOD) has shown how to eliminate certain key simulation artifacts having to do with autonomous behavior over time, as well as certain simulation environment artifacts having to do with updating the display of simulation state. Our initial experiments suggest that a coordinated suite of multilevel models can be built to allow dynamic aggregation. The consistency criteria that must be developed in the process can be regarded as semi-automated tools for model building. The initial results of our propagative approach to sensitivity analysis suggest that our approach may ultimately make this process cost-effective for a wide range of computations, including much of what goes on in a typical military simulation.

Viewed abstractly, our work to date has:

- Developed new techniques for eliminating artifacts
  - XROSS extensions
  - Elimination of explicit "unplanning" via DMOD
- Developed new techniques for answering **Beyond *"What if. . . ?"*** questions
  - Logic-based simulation/reasoning via DMOD
  - New propagative approach to sensitivity analysis
- Evolved a new "multi-view" paradigm for modeling
  - Object and event views
  - Multiple relations among objects
  - Dynamic aggregation

On the other hand, we have avoided or postponed investigating certain issues; in particular, we have *not*:

- Concentrated on performance issues, *except indirectly through:*
  - Update-on-demand ("lazy evaluation") strategy (XROSS)
  - Elimination of explicit "unplanning" (DMOD)
  - Aggregation to "focus the attention" of a simulation
  - New computationally feasible approach to sensitivity analysis
- Pursued the modeling of "soft concepts"

  Which requires major investment in modeling decisionmaking

- Implemented a detailed scenario demonstration
  - Our original river crossing scenario lacked air-land integration
  - A simple air-land scenario has been evolved in DMOD
  - A fragmentary land scenario has been developed for aggregation

Continued research in knowledge-based simulation must investigate all of these areas in further detail. Though we have made what we feel is significant progress, there are still many unexplored issues. Some of these issues have been discussed in the above sections and will be the subject of future publications; others will only emerge as our research progresses. In particular, we hope to:

- Explore the use of reasoning to provide validation
  - Pursue interactive exploration and explanation
  - Develop techniques for valid composition of models
  - Explore formal specification of models and scenarios
- Pursue multiple views for validation through comprehension
  - Integrate the object and event views of models
  - Integrate multiple relations, aggregation, and extended objects

There seems to be little agreement as to what validation means or ought to mean. We feel that this subject is too vital to ignore, and we intend to interact with other modelers in an attempt to refine an operational definition of validity. This investigation must distinguish between "face" validity (associated with various kinds of "realism") and true validity, and it must question the common, implicit assumption that greater detail in a model necessarily produces greater validity.

We hypothesize that the reasoning techniques we have been exploring will contribute to validation in several ways. First, the ability to ask questions of a model that extend beyond "*What if. . . ?*" should allow a model builder to verify the behavior of a system (i.e., to make sure that it behaves as intended). Though this is not sufficient to ensure the validity of the model itself, it *is* necessary to ensure that the system correctly implements the intended model. Second, we believe that the logic-based approach embodied in DMOD will allow the development of formal techniques for composing submodels, while ensuring that validity (if it is present in the submodels) is preserved in their composition. Finally, we believe that a suitable generalization of DMOD can serve as a formal specification language for models; we believe that this formalism may also provide a means of validating scenarios (as suggested in Builder, 1983). We therefore intend to continue our exploration of these research areas and the development of these techniques.

In addition, we believe that making a model more comprehensible by providing multiple views (objects, events, multiple relations, aggregation, etc.) will make validation easier by making it more apparent what the actual model is. We therefore intend to continue our development of the multi-view paradigm in order to improve the comprehensibility of models.

We believe that these research efforts will ultimately lead to a new generation of simulation environments that are at once more powerful, more flexible, and more comprehensible than those currently available.

Appendix

MODELING INITIATIVE IN COMMANDERS

As discussed in Section VI above, we have decided to postpone our work in the area of modeling soft concepts such as "initiative," since in order to model commanders, it is necessary to model the decisionmaking process as a whole and to build simulations that are driven by plans (just as the real-world actions of commanders are based on the plans they formulate or receive from their commanders). The following summarizes the results of our initial exploration of this area.

The qualitative concept we chose as an example of modeling "soft" concepts is "initiative." This concept is often mentioned in the literature (Peters, 1986; Vorobyov, 1983; Donnelly, 1984; Armstrong, 1984) and appears to differentiate U.S. and Soviet tactics in ground warfare, especially in river crossing operations. Initiative is usually described as *having the freedom and capability to plan and act independently.* A commander exercises little initiative if he simply executes detailed instructions passed down from his superior. Initiative is also absent if the commander responds to situations "by the book," invoking responses according to predetermined rules.

A popular notion is that the Soviets exhibit little initiative in their operations, relying on centralized command and lockstep execution of their plans, while the U.S. forces tend to be more distributed in nature, with lower command levels having more responsibility and initiative. For example, the Soviets tend to drill their units heavily on a few preset plans, while the American forces train reactive decisionmaking skills. To a certain extent, this reflects the Soviet emphasis on coordinated, offensive actions and the West's focus on defensive operations. Even in a defensive operation, though, the Soviets have been known to forgo opportunities, preferring preplanned applications of force over the use of initiative. They typically provide detailed contingency plans to their commanders, and often will not take advantage of an unplanned opportunity, as this might throw off their overall plan. On the other hand, U.S. forces attempt to capitalize on any weaknesses or opportunities that present themselves.

These generalities are not absolutely true, of course, as commanders on both sides have been found to have a wide range of command styles; for example, this was true in the Kabul operation in Afghanistan, where the Soviets undertook a rapid, high-risk attack at night in a strange city (Luttwak, 1985).

Initiative can be useful in some circumstances and detrimental in others. The advantages of initiative are that lower echelons of command can opportunistically respond to threats, without having to rely on vulnerable and slow communication links or having to act in a stereotypical manner in accordance with pre-specified tactics. Many of the goals of AirLand 2000, such as surprise, rapid maneuver, disruption, and quick reaction, should be facilitated through use of greater initiative (Martin, 1983; U.S. Army TRADOC, 1981). Too much initiative, on the other hand, can result in reduced coherence of overall planning and execution. High level commanders cannot confidently rely on units to act in a coordinated manner, and low level commanders can build up few expectations about neighboring forces, unless there is substantial communications traffic. These considerations are especially important in a highly coordinated offensive operation with tight time lines. As noted earlier, the Soviets tend to concentrate on such offensive operations in their war plans, while U.S. and NATO forces typically focus on defensive operations (Veit, Rose, and Callero, 1981).

Another aspect that differentiates Soviet and Western views on initiative is command organization. Soviet units are typically very cohesive and centralized in their organization. A Soviet commander is frequently responsible for all operations in a unit, while a U.S. commander tends to delegate authority. For example, a U.S. commander may use "management by exception," in which the subordinate units will ask for help only if they cannot accomplish their assigned objectives. The Soviets will more often give specific orders, with deviation by the subordinate allowed only after an authorization by the commander. Soviet and U.S. units are also different in composition; U.S. units employ cross attachment of forces with other units, changing the force mix when necessary, while the Soviets tend to fix forces with the commander (even including reserves). The Soviets also use communication networks that overlap as many as three echelons, allowing commanders to contact low level subordinates directly; U.S. communications usually have one network for each echelon (Patrick, 1979). All of these factors presumably lead to a centralization of authority and reduction of initiative in the Soviet system.

Our initial plan (which we have postponed, as discussed in Section VI above) was to model many of the above factors to produce a range of apparent levels of initiative. The specific behaviors would result from interviews with commanders and from studies of Army Field Manuals FM 90-13 and FM 30-102 and the Soviet Military Thought series (Grechko, 1975; Kozlov, 1971; Chuyev and Mikhaylov, 1975). The resulting models would be consistent with these sources but were not expected to be absolutely complete or definitive. We planned to check that the resulting simulation behaviors were plausible and realistic (by interacting with commanders), but we did not intend to spend excessive amounts of time ensuring that the resulting models were internally consistent and valid, since the main objective of our task was to explore, evaluate, and demonstrate techniques for modeling soft concepts such as initiative.

The primary factors we planned to model in our exploration of initiative included command structure, communication links, types of messages sent (e.g., command messages, data, constraints, goals), message delays and degradations, and decision time stress. These are all reasonably well-defined factors. Some important but less well-defined factors include forms of situation assessment and planning, and types of goal structure (e.g., attempting to achieve individual, unit, or high level objectives). For example, we could model different forms of planning present under different command regimes. The key planning forms seem to be condition-response, projection-based planning, and optimization. Condition-response planning is present when the decisionmaker applies a "cookbook" rule to the situation, without extrapolating the consequences of the decision. This involves minimal initiative. Projection-based planning also employs canned responses, but is conditional on the decisionmaker's extrapolation of the consequences. Optimization allows the greatest initiative, as the decisionmaker is unconstrained in his generation and testing of new options.

To complete the simulation, we planned to collect and display performance data sensitive to the presence of initiative. The key performance measures we envisioned were communication load, decision response time, options considered, number of coordination messages, and operational conflicts (firing at the same target, moving into the same assembly area). These are in addition to the "normal" outcome measures of attrition, FEBA (forward edge of battle area) movement, etc.

A river crossing scenario would provide an excellent vehicle for exploring the effects of allowing different degrees of initiative. Organization, tactics, and procedures of both the Soviet and the U.S. forces could be varied. Table A.1 summarizes some key conditions, indicating low and high initiative levels.

Rather than exploring all combinations of the above factors, we planned to examine several plausible combinations corresponding to extremes of Soviet and U.S. doctrine.

The question of modeling initiativ its well with a number of other objectives of the KBSim project. The structure and dynamics of the command hierarchy, for example, are key issues in the modeling of object relations. Communication and decision delays present with different command structures are important to planning and temporal reasoning. Determination of vulnerable communication links is important to explanation and exploration, as well as providing insights about the effects of initiative. Finally, simulation at multiple levels of aggregation is necessary to model the effects of different levels of initiative. Highly detailed operations at the lower levels of command are

Table A.1

INITIATIVE CONDITIONS

Condition	Low Initiative	High Initiative
Type of order:	Explicit instructions	General goals
Planning mode:	Condition response	Projection, optimization
Comm network:	Overlapping echelons	Separate for each echelon
Division structure:	Invariant	Cross attachment
Command post (CP):	Large, stationary CP	Mobile tactical CPs
Troop type:	Unseasoned, low capability	Seasoned, capable
Command locus:	Centralized	Distributed
Mission type:	Offensive, deliberate	Defensive, hasty

important when initiative is high, while top-level command operations are most important during low initiative, highly scripted operations.

An additional interesting aspect of initiative is that of knowledge distribution among units. Modeling different levels of initiative implies the need to display the degree of knowledge required at different command levels. For example, it is necessary to establish what data must be passed to the lower levels to allow them to assess the situation, generate plans, and achieve local and aggregate objectives. At the other end of the scale, it is necessary to establish what data are necessary to execute and monitor a completely specified set of orders.

Although the study of initiative illuminates many interesting modeling issues (as well as verifying practical questions of interest to analysts), we feel it should only be undertaken within the larger context of modeling the human decision process as a whole. We have therefore postponed further work in this area until we can devote an appropriate effort to exploring this larger context.

BIBLIOGRAPHY

Anderson, R. H., and J. J. Gillogly, *RAND Intelligent Terminal Agent (RITA): Design Philosophy*, The RAND Corporation, R-1809-ARPA, February 1976.

Anderson, R. H., and N. Z. Shapiro, *Design Considerations for Computer-Based Interactive Map Display Systems*, The RAND Corporation, R-2382-ARPA, February 1979.

Anderson, R. H., et al., *RITA Reference Manual*, The RAND Corporation, R-1808-ARPA, September 1977.

Armstrong, R. N., "Initiative Soviet Style," *Military Review*, June 1984, pp. 14–27.

Blanning, R. W., "Sensitivity Analysis in Logic-based Models," *Decision Support Systems*, Vol. 3, 1987, pp. 343–349.

Bowen, K. C., "Analysis of Models," University of London, Department of Mathematics (unpublished), 1978.

Brachman, R. J., "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks," *Computer*, Vol. 16, No. 10, 1983.

Bryce, D., and R. Hull, "SNAP: A Graphics-based Schema Manager," *The Proceedings of the International Conference on Data Engineering*, IEEE Cor Society, Los Angeles, 1986.

Builder, C. H., *Toward a Calculus of Scenarios*, The RAND Corporation, N-1855-DNA, 1983

Callero, M., D. A. Waterman, and J. R. Kipps, *TATR: A Pro[illegible]type Expert Syst[illegible] for Tactical Air Targeting*, The RAND Corporation, R-3096 ARPA, August 1984.

Cammarata, S., B. Gates, and J. Rothenberg, "Dependencies and Graphical Interfaces in Object-Oriented Simulation Languages," *Proceedings of the 1987 Winter Simulation Conference* (Atlanta, GA, Dec. 14–16), Society for Computer Simulation, San Diego, CA, 1987, pp. 507–517.

Cammarata, S. J., B. L. Gates, and J. Rothenberg, *Dependencies, [illegible] an Graphical Interfaces in the ROSS Language*, The RAND Corp[illegible], N-2589-DARPA, March 1988.

Carnegie Group, Inc., *Knowledge Cra[illegible] CRL Technical Man[illegible]al*, Pittsburgh, 1986.

Chuyev, Yu. V., and Yu. B. Mikhaylov, "Forecasting in Military Affairs (A Soviet View)," *S[illegible]iet Military Though[illegible] No. 16*, U.S. Government Printing Office, Washington, DC, 1975.

Conway, R. W., *Some Tactical Problems in Simulation Method*, The RAND Corporation, RM-3244-PR, October 1962.

Cox, B. J., "Objective-C: Outlook," *Journal of Object Oriented Programming*, Vol. 1, No. 1, April/May 1988, pp. 54–57.

Dalkey, N. C., "Simulation," in E. S. Quade and W. I. Boucher (eds.), *Systems Analysis and Policy Planning: Applications in Defense*, Elsevier, New York, 1968.

DARPA (Defense Advanced Research Projects Agency) *Strategic Computing 2nd Annual Report*, Washington, DC, February 1986.

Davis, M. R., and T. O. Ellis, "The Rand Tablet: A Man-Machine Graphical Communication Device," *FJCC 1964*, Spartan Books.

Davis, M., S. J. Rosenschein. and N. Z. Shapiro, *Prospects and Problems for a General Modeling Methodology*, The RAND Corporation, N-1801-RC, June 1982.

Davis, P. K., and J. A. Winnefeld, *The RAND Strategy Assessment Center: An Overview and Interim Conclusions about Utility and Development Options*, The RAND Corporation, R-2945-DNA, March 1983.

Davis, P. K., S. C. Bankes, and J. P. Kahan, *A New Methodology for Modeling National Command Level Decisionmaking in War Games and Simulations*, The RAND Corporation, R-3290-NA, July 1986.

Donnelly, C., "Soviet Fighting Doctrine," *NATO's Sixteen Nations*, Vol. 29 (3), May/June 1984.

Ellis, T. O., J. F. Heafner, and W. L. Sibley, *The GRAIL Project: An Experiment in Man-Machine Communications*, The RAND Corporation, RM-5999-ARPA, September 1969.

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

Fishwick, P. A., "Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction," *UF CIS Technical Report TR-86-6*, University of Florida, September 1986.

Gass, S. I., and R. L. Sisson, *A Guide to Models in Governmental Planning and Operations*, U.S. Environmental Protection Agency, Washington, DC, 1974.

Gilmer, J. B., *Parallel Simulation Techniques for Military Problems*, The BDM Corporation, McLean, VA, 1986.

Ginsberg, A. S., H. M. Markowitz, and P. M. Oldfather, *Programming by Questionnaire*, The RAND Corporation, RM-4460-PR, April 1965.

Grechko, M.A.A., "The Armed Forces of the Soviet State (A Soviet View)," *Soviet Military Thought No. 12*, U.S. Government Printing Office, Washington, DC, 1975.

Greenberger, M., M. A. Crenson, and B. L. Crissey, *Models in the Policy Process*, Russell Sage Foundation, New York, 1976.

Hilton, M. L., *ERIC: An Object-oriented Simulation Language*, Rome Air Development Center, RADC-TR-87-103, Rome, NY, 1987.

Hughes, W. P., *Military Modeling*, The Military Operations Research Society, Inc., Alexandria, VA, 1984.

IntelliCorp, *The SimKit System Knowledge-Based Simulation Tools in KEE*, Mountain View, CA, 1985.

Kamins, M., *Two Notes on the Lognormal Distribution*, The RAND Corporation, RM-3781-PR, August 1963.

Kipps, J. R., B. A. Florman, and H. A. Sowizral, *The New ROSIE Reference Manual and User's Guide*, The RAND Corporation, R-3448-DARPA/RC, June 1987.

Kiviat, P. J., *Digital Computer Simulation: Modeling Concepts*, The RAND Corporation, RM-5378-PR, August 1967.

Kiviat, P., R. Villanueva, and H. Markowitz, *The SIMSCRIPT II Programming Language*, Prentice-Hall, Englewood Cliffs, NJ, 1968.

Klahr, P., "Expressibility in ROSS: An Object-oriented Simulation System," *AI APPLIED TO SIMULATION: Proceedings of the European Conference at the University of Ghent*, February 1985, pp. 136–139.

Klahr, P., and D. A. Waterman, "Artificial Intelligence: A Rand Perspective," *Expert Systems Techniques, Tools and Applications*, Addison-Wesley, 1986, pp. 3–23.

Klahr, P., et al., *SWIRL: Simulating Warfare in the ROSS Language*, The RAND Corporation, N-1885-AF, September 1982.

Klahr, P., et al., *TWIRL: Tactical Warfare in the ROSS Language*, The RAND Corporation, R-3158-AF, September 1984.

Kozlov, S. N., "The Officer's Handbook," *Soviet Military Thought No. 13*, U.S. Government Printing Office, Washington, DC, 1971.

Luttwak, E. N., *The Pentagon and the Art of War*, Simon and Schuster, New York, 1985.

Martin, H., "AirLand Battle 2000 Is Being Implemented with a High Technology Light Division," *Military Electronics/Countermeasures*, January 1983, pp. 28–36.

McArthur, D., *Building Learning and Tutoring Tools for Object-Oriented Simulation Systems*, The RAND Corporation, R-3443-DARPA/RC, July 1987.

McArthur, D., and P. Klahr, *The ROSS Language Manual*, The RAND Corporation, N 1854-AF, Septemb[illegible] 1982.

McArthur, D., P. Klahr, and S. Narain, *ROSS: An Object-Oriented Language for Constructing Simulations*, The RAND Corporation, R-3160-AF, December 1984.

McArthur, D., P. Klahr, and S. Narain, *The ROSS Language Manual*, The RAND Corporation, N-1854-1-AF, September 1985.

McCarthy, J., and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in B. Meltzer and D. Michie (eds.), *Machine Intelligence*, Edinburgh University Press, Edinburgh, 1969.

McFall, M. E., and P. Klahr, "Simulation with Rules and Objects," *Proceedings of the 1986 Winter Simulation Conference*, Washington, DC, 1986, pp. 470–473.

Newell, A., J. C. Shaw, and H. Simon, "Empirical Explorations with the Logic Theory Machine," *The Proceedings of the Western Joint Computer Conference*, Institute of Radio Engineers, New York, 1957.

Nugent, R. O., and R. W. Wong, "The Battlefield Environment Model: An Army-Level Object-Oriented Simulation Model," *The Proceedings of the 1986 Summer Simulation Conference*, Society for Computer Simulation, San Diego, CA, 1986.

Paul, J., D. A. Waterman, and M. A. Peterson, "SAL: An Expert System for Evaluating Asbestos Claims," *The Proceedings of the First Australian Artificial Intelligence Congress*, Computerworld, Ltd., Melbourne, November 18–26, 1986.

Peters, R., "Unmatched Spurs: A False Step in Soviet Doctrine?" *Military Intelligence*, Vol. 12, No. 1, January-March 1986, pp. 14–58.

Quade, E. S., "Modeling Techniques," in H. J. Miser and E. S. Quade (eds.), *Handbook of Systems Analysis*, Elsevier, New York, 1985.

Reingold, E. M., and J. S. Tilford, "Tidier Drawings of Trees," *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 2, 1981.

Rothenberg, J., "Object-oriented Simulation: Where Do We Go from Here?" *Proceedings of the 1986 Winter Simulation Conference*, Washington, DC, 1986, pp. 464–469.

Rothenberg, J., "The Nature of Modeling," in L. Widman et al. (eds.), *Artificial Intelligence, Simulation and Modeling*, John Wiley & Sons, Inc., New York, 1989.

Rumbaugh, J., "Relations as Semantic Constructs in an Object-oriented Language," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OPSLA'87)*, Association for Computing Machinery, Orlando, FL, October 4–8, 1987.

Sharpe, W. F., *The Army Deployment Simulator*, The RAND Corporation, RM-4219-ISA, March 1965.

Sowizral, H. A., and J. R. Kipps, *ROSIE: A Programming Environment for Expert Systems*, The RAND Corporation, R-3246-ARPA, October 1985.

Steeb, R., S. J. Cammarata, S. Narain, J. Rothenberg, and W. D. Giarla, *Cooperative Intelligence for Remotely Piloted Vehicle Fleet Control: Analysis and Simulation*, The RAND Corporation, R-3408-ARPA, October 1986.

U.S. Army, *River Crossing Operations*, Field Manual, FM 90-13, November 1978.

U.S. Army, *Assault River Crossing Operations*, Army Field Manual Attachment, FM 30-102, November 1977, pp. 16-6 to 16-13.

U.S. Army TRADOC, *The AirLand Battle and Corps 86*, Pamphlet 525-5, March 1981.

Vaucher, J. G., "Pretty-Printing of Trees," *Software—Practice and Experience*, Vol. 10, 1980.

Veit, C. T., M. Callero, and B. J. Rose, *Introduction to the Subjective Transfer Function Approach to Analyzing Systems*, The RAND Corporation, R-3021-AF, March 1984.

Veit, C. T., B. J. Rose, and M. Callero, *Subjective Measurement of Tactical Air Command and Control—Vol. III: Preliminary Investigation of Enemy Information Components*, The RAND Corporation, N-1671/3-AF, March 1981.

Voosen, B. J., *PLANET: Planned Logistics Analysis and Evaluation Technique*, The RAND Corporation, RM-4950-PR, January 1967.

Vorobyov, I., "Developing the Commander's Initiative," *Soviet Military Review*, May 1983, pp. 18–21.

Waterman, D. A., and M. A. Peterson, *Models of Legal Decisionmaking*, The RAND Corporation, R-2717-ICJ, 1981.