

RL-TR-97-113
Final Technical Report
October 1997



INTEGRATING STOCHASTIC AND SIMULATION-BASED MODELS INTO C3I AUTOMATED PLANNING TASKS

University of Florida

Paul A. Fishwick

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

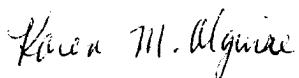
19980209 078

DTIC QUALITY INSPECTED

**Rome Laboratory
Air Force Materiel Command
Rome, New York**

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-113 has been reviewed and is approved for publication.

APPROVED: 
KAREN M. ALGUIRE
Project Engineer

FOR THE DIRECTOR: 
JOHN A. GRANIERO, Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3CA, 525 Brooks Rd, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE Oct 97		3. REPORT TYPE AND DATES COVERED Final Aug 95 - Dec 96
4. TITLE AND SUBTITLE INTEGRATING STOCHASTIC AND SIMULATION-BASED MODELS INTO C3I AUTOMATED PLANNING TASKS			5. FUNDING NUMBERS C - F30602-95-1-0031 PE - 62702F PR - 4600 TA - AA WU- 02	
6. AUTHOR(S) Paul A. Fishwick				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Dept of Computer & Information Science and Engineering University of Florida Bldg. CSE, Room 301, PO Box 116120 Gainesville, FL 32611-6120			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory/C3CA 525 Brooks Road Rome, NY 13441-4505			10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-97-113	
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Karen M. Alguire, C3CA, 315-330-4833				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE N/A	
13. ABSTRACT (Maximum 200 words) This project focused on research involving a simulation-based planning methodology and toolkit to help in real-time planning and decision making. A new methodology called OOPM (Object Oriented Physical Modeling) was developed. Development of the MOOSE (Multimodeling Object Oriented Simulation Environment) system, based on the OOPM methodology, has also been underway including modeling windows for finite state automata, differential equations, and functional models. The object-oriented multimodel framework of MOOSE is composed of three parts consisting of a graphical user interface, BLOCKS models based on the BLOCKS modeling language, and the SimPack Toolkit. MOOSE represents a software prototype for constructing simulation-based planning scenarios. The current scenario involves an interdiction mission. The goal is to achieve the mission goals with the fewest casualties. The best plan to achieve this goal is determined with a combination of a "qualitative" rule-based model and lower level "quantitative" block-structured models. The MOOSE architecture permits a model abstraction capability to allow users to realize time constraints on the simulation-based planning by simulating models at different levels. The use of an iterative deepening approach to simulation is being investigated.				
14. SUBJECT TERMS simulation-based planning, object oriented simulation, multimodeling, physical modeling			15. NUMBER OF PAGES 180	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Contents

1	INTRODUCTION	3
2	ACCOMPLISHMENTS	3
3	MOOSE	3
3.0.1	Object-Oriented Multimodel Framework	4
4	APPENDICES	5

1 INTRODUCTION

This is the Final Technical Report prepared under the Contract F30602-95-1-0031, entitled "Integrating Stochastic and Simulation-Based Models into C³I Automated Planning Tasks." It summarizes progress on this contract for 1996.

The most comprehensive list of online (with full text and graphics) publications can be found in the PIs web page. Many of these publications acknowledge grant F30602-95-1-0031, however, we are including only those publications with direct relevance to the grant's focus on simulation-based planning.

2 ACCOMPLISHMENTS

Several publications were written in the area of simulation-based planning to disseminate the results of the funded work:

- Ph.D. dissertation of Dr. Jin Joo Lee (San Jose State University)
- Journal article in Simulation, November 1996
- Journal article in IEEE Transactions on Systems, Man and Cybernetics, July 1996, Vol. 26, No. 4
- Journal article submitted to ACM Transactions on Modeling and Computer Simulation, August 1996.
- Magazine article in Phalanx, published by the Military Operations Research Society

We developed a new methodology called OOPM (Object Oriented Physical Modeling) and published this in a manuscript to ACM Transactions on Modeling and Computer Simulation. A C++ based system called MOOSE, based on OOPM, is under construction. Modeling windows are underway for finite state automata, differential equations and functional models. These are the key model types used in the simulation-based planning work. More information on MOOSE can be obtained by visiting the web site: <http://www.cise.ufl.edu/~fishwick/moose.html>.

We published a paper on model abstraction in the recent Winter Simulation Conference held in San Diego, CA, December 1996. Our intent is to include model abstraction capability in MOOSE to allow users to realize time constraints on the simulation-based planning by simulating models at different levels. We are investigating the use of an iterative deepening approach to simulation that is similar to that found in the computer gaming literature where a breadth-first strategy is used. Using this method, highly aggregate models are simulated and refinement proceeds to more detailed models as time permits.

3 MOOSE

MOOSE (Multimodeling Object Oriented Simulation Environment) represents our software prototype for constructing simulation-based planning scenarios.

3.0.1 Object-Oriented Multimodel Framework

The design framework is complete and is composed of three parts:

1. Graphical User Interface
2. *BLOCKS* models
3. SimPack

These are defined in more detail below:

1. *Graphical User Interface (GUI)*: The Graphical User Interface allows users to both construct visually- oriented models, as well as view the output from these models in the 2D color scenario window. In many ways, MOOSE will appear similar to commercial simulation packages, such as SES Workbench, when it is complete. Multi-level, graphical object-oriented model building will be facilitated. The new aspects of MOOSE, over previously done work, are: (1) support for abstraction and multimodeling; and (2) support for planning using simulation. The software is also very portable and freely available to other researchers for further extension. This philosophy is built upon our previous SimPack software started in 1990. SimPack currently has 170 users worldwide and is documented in the PIs home page <http://www.cis.ufl.edu/~fishwick>. The first aspect, multimodeling, allows users to construct multi-level models where each level represents an abstraction level for the system being designed or modeled. Multimodeling affords two types of abstraction: (1) structural and (2) behavioral. The structural abstraction is a byproduct of the hierarchical design process performed using the GUI and the use of the *BLOCKS* language to ensure that all system components are coupled correctly (interlevel and intralevel). The behavioral abstraction allows a user to remove lower levels of abstraction through system identification procedures (using a linear systems and neural network approaches).

We are using the Tk/Tcl toolkit for our GUI development work. Tk/Tcl permits sophisticated GUIs to be developed on top of *BLOCKS* and C++. Three Tk/Tcl Windows are planned. Each window will have control icons (buttons with an icon instead of text). This means each window can accept input and produce output in that window.

- (a) *Window 1 (Experiment Window)*: Specify how many replications to run. Specify the goal. Stopping criterion and time duration of simulation. Specify both an analog and digital clock (like X windows) for both real-time and simulated-time. Terminating or Non-Terminating simulations supported. Specify one of five possible criteria for optimization. This window controls what is to be done during execution.
- (b) *Window 2 (Scenario Window)*: This represents the output of the simulation. What object behaviors should be displayed and how should the objects be represented? There should be a base map and overlay maps. Also, it should be possible to view graphs of one variable versus another (such as state vs. time). Graphs could just end up being pop-up windows which come up over the scenario

window (but which can be moved around). Icons can move around over the maps (and overlays).

For multiple simulations (see Experiment Window), we can opt to view only the most recent replication, or possibly watch all of them as they trace out paths over the map.

- (c) *Window 3 (Modeling Window)*: Build and display visual models. All models are visual with a model component being 1) text, 2) pixmap or 3) vector. The Model structure is stored in a flat file.

2. *BLOCKS Modeling Language* The *BLOCKS* modeling language provides an assembly language for the different types of models supported in MOOSE. The primary types of initially supported models are: 1) FSA, 2) functional, 3) Petri net and 4) Equational. In many ways, the *BLOCKS* language resembles a modeling language for digital circuits. All supported model types are translated into *BLOCKS* models and then simulated using the SimPack toolkit.
3. *SimPack Toolkit* SimPack is a collection of C++ programs and libraries to support simulation. Currently, SimPack supports individual model types by allow the user access to libraries for event scheduling, queuing, and equation solving. Also, we are adding tools for experimental design and analysis to SimPack, which supports the simulation-based planning and decision making.

4 APPENDICES

The following manuscripts are included as appendices.

1. Paul A. Fishwick. "Extending Object Oriented Design for Physical Modeling". Submitted to ACM Transactions on Modeling and Computer Simulation, August 1996.
2. Yi-Bing Lin and Paul A. Fishwick. "Asynchronous Parallel Discrete Event Simulation". IEEE Transactions on Systems, Man and Cybernetics, Volume 26, Number 4, July 1996, pp. 397-412.
3. Paul A. Fishwick, Gyooseok Kim and Jin Joo Lee. "Improved Decision Making through Simulation Based Planning" Simulation, November 1996.
4. Jin Joo Lee. "A Simulation-Based Approach for Decision Making and Route Planning" Ph.D. Dissertation, July 1996.

SUBMISSION FOR SPECIAL ISSUE ON MODEL SPECIFICATION &
REPRESENTATION
FOR ACM TRANSACTIONS ON MODELING AND COMPUTER SIMULATION

Extending Object-Oriented Design for Physical Modeling

Paul A. Fishwick
Dept. of Computer & Information Science and Engineering
University of Florida
Bldg. CSE, Room 301
Gainesville, FL 32611
E-mail: fishwick@cise.ufl.edu
Phone and FAX: (352) 392-1414
WWW: <http://www.cise.ufl.edu/~fishwick>

July 12, 1996

Extending Object-Oriented Design for Physical Modeling

Paul A. Fishwick

Dept. of Computer & Information Science and Engineering
University of Florida
Bldg. CSE, Room 301
Gainesville, FL 32611

October 10, 1996

Abstract

When we build simulation models and construct dynamical models for physical systems, we often do not do so using a clear overall framework that organizes our geometry, dynamics and models. How do geometry and dynamics intertwine to effect system change over multiple abstraction levels? We present a methodology, called *object-oriented physical modeling* (OOPM), which builds on the currently accepted computer science approach in object-oriented program design. This type of modeling injects a way of incorporating geometry and dynamics into general object-oriented design. Moreover, we present an approach to dynamical modeling that mirrors major categories of computer programming languages, thereby achieving a definition of system modeling that reinforces the relation of *model* to *program*.

1 Introduction

Simulation is divided into three areas: 1) *model design*, 2) *model execution*, and 3) *execution analysis* shown in Fig. 1. Modeling is the process of abstracting real world behavior into a more economical form for purposes of experimentation and learning. Our chief interest is in efficiently capturing and organizing the knowledge necessary to simulate physical systems, both artificial and natural. Simulation requires that we have a way of designing and executing models. Models can represent geometric shapes of real objects or the dynamics of those objects. The shape of an object is captured by its geometry, which is used by computer graphics to display the object. The dynamics of an object allows us to view a computer animation of the object undergoing time-dependent change. Our purpose is to specify a method for modeling a physical system, while claiming that our method provides benefits such as model and model component re-use. We provide a way of organizing physical knowledge and a methodology for those wanting to model systems at many levels of detail. Our techniques build on top of object-oriented design principles espoused in both computer science as well as computer simulation. The method surfaces the importance of an integrated object-process method where both *objects* and *processes* are made visible in model design. We present

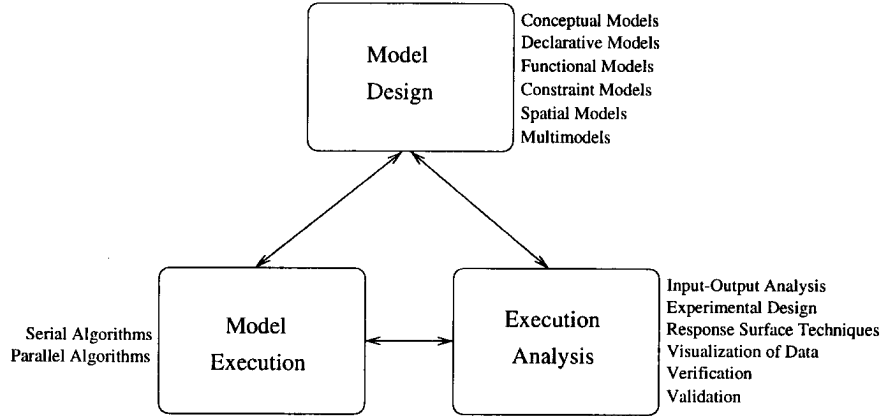


Figure 1: The study of computer simulation: three subfields.

a uniform method that extends previous work by specifying how shape and dynamics are integrated in the same framework, and by defining methods for modeling that permit an integration of existing modeling methods without recommending an all-encompassing new modeling technique. In this sense, the contribution is one where we devise a specific *model integration* approach.

The problems facing most model integration situations are analogous to the carpenter, plumber, and electrician constructing a house. Each has separately created a working network: the carpenter has built the frame, the electrician has soldered wires and boxes for a completely wired house circuit, and the plumber has assembled the plumbing and heating. No blueprint has been used to guide each of the three, and so arguments arise and fast but inelegant fixes are made just to “make it all work.” The three work in isolation in performing their functions. The absurdity of this situation is clear. Without a blueprint to serve as a design of house integration, the house cannot be constructed. To model the large-scale systems properly, many models must be written and assembled. We must find a way for the people to communicate by specifying their models using a common framework. After-the-fact glue and paste methods may be necessary initially, but a knowledge representation and design framework must be created. Only then, should computer code be written to execute the models.

In Sec. 2, we discuss our motivation for this work as well as a literature search, specifying what groups have performed similar studies to our own. We introduce a generic example scenario, in Sec. 3, of a robot moving over a space. This example will be used throughout the paper as a common thread. Then, in Sec. 4, we define our concepts as well as object-oriented design as it is practiced in software engineering and simulation. Since modeling is a process, we present the model design aspects for *model engineering*, which defines how models are created from first principles and a knowledge of the system to be modeled. Sec. 4 supplies the phases needed in engineering the model. Sec. 5 discusses the construction of the conceptual model. The conceptual model provides a kind of skeletal structure or scaffolding which requires attributes and methods to make it complete. Sec. 6 describes the modeling approach for satisfying this completion. There are two types of models: static models (Sec. 6.1) and dynamic models (Sec. 6.2). In Secs. 7 and 8, we close the article with

a description of our current implementation of this model engineering approach, MOOSE, and then summarize what we’ve learned and our future directions.

2 Motivation and Background

The word *model* is a somewhat overloaded term and can have many meanings depending on context. We proceed to define what we mean by the word *model*. Models are devices used by scientists and engineers to communicate with one another using a concise—often visual—representation of a physical system. Models are visual high-level constructs that we use to communicate system dynamics without the need for frequent communication of low-level formalism, semantics and computer code. In our methodology [13], a *model* is defined as one of the following: 1) a graph consisting of nodes, arcs and labels, 2) a set of rules, or 3) a set of equations. Computer code and programs are not considered to be models since code semantics are specified at too low a level. Likewise, formal methods [36, 56] associate the formal semantics with models but do not focus on representing the kind of high-level form needed for modeling. One of our thrusts in this paper is to discourage readers from thinking that to simulate, they need to choose a programming language and then proceed directly to the coding phase. Even the phrase “object-oriented” is often defined as being synonymous with certain programming languages such as C++, and so one may be lead to begin programming, using polymorphism, virtual base classes and other artifacts, before the design is specified. Without the proper scaffolding for our models, in the form of a conceptual model, we will produce disorganized pieces of code without a good understanding and organization of the physical process we wish to study. The act of coding in an object-oriented language is not a substitute for doing good design. As an example, C++ provides many object oriented capabilities, but does not enforce object oriented design. Norman [35] points out the need for good visual, conceptual models in general design for improved user-interfaces to physical instruments and devices. The importance of design extends to all scientific endeavors with a focus on models. Models need to provide a *map* between the physical world and what we wish to design and subsequently implement either as a program or a physical construction.

Programs and formal specifications [54, 56, 39] are a vital ingredient in the simulation process since, without these methods, modeling approaches lack precision and cohesion. However, formal specifications should not take the place of models since they serve two different purposes. Specifications are needed to disambiguate the semantics, at the lowest level, of what one is modeling. Models exist to allow humans to communicate about the dynamics and geometry of real world objects. Our definition of modeling is described at a level where models are translated into executable programs and formal specifications. Fishwick and Zeigler [16] demonstrated this translation using the DEVS [56] formalism for one particular type of visual multimodel (finite state machine model controlling a set of constraint models). For other types of multimodels, one can devise additional formalisms [39]. Object-oriented methodology in simulation has a long history, as with the introduction of the Simula language [3], which can be considered one of the pioneering ways in which simulation applied itself to “object-oriented thinking.” Simula provided many of the basic primitives for class construction and object oriented principles but was not accompanied by a visually-oriented engineering approach to model building that is found in more recent software engineering

texts [45, 5, 18]. As we shall see in model design, the visual orientation is critical since it represents the way most scientists and engineers reason about physical problems and, therefore, must be made *explicit* in modeling. Other more recent simulation thrusts in the object-oriented arena include SCS conferences [41] as well as numerous Winter Simulation Conference sessions over the past ten years. Also, various simulation groups have adopted the general object-oriented perspective [44, 57, 22, 1].

We specify two contributions: 1) a comprehensive methodology for constructing physical objects that encapsulate both geometric and dynamical models, and 2) a new taxonomy for dynamic models. The motivation for the first contribution is that there currently exists no method that uses object-oriented design and specifies an enhancement of this design to accommodate static and dynamic models. We have taken the existing visual object-oriented design approaches reflected in texts such as Rumbaugh [45] and Booch [4] and extended these approaches. Regarding the motivation for deriving a new method for dynamic modeling, we offer the following reasons:

1. *Object-Oriented Design*: The new taxonomy is one based on object-oriented design methodology since it is developed as an extension to object-oriented design. Existing object-oriented design for software engineering does not include the concept of modeling. Our design extends the design approaches in software engineering to employ both static and dynamic models for physical objects. Furthermore, we include both static and dynamic models, which capture an integrated physical system with geometry and dynamics.
2. *Orientation*: Even though we describe our method as “object-oriented,” the method places equal value on processes and objects. A *process* is surfaced through the use of *dynamic models* (ref. Sec. 6.2), which are stored as method of objects. The problem with traditional object-oriented design is that it tends to bury the process as code while surfacing objects through visually oriented class hierarchies and object relations. It is necessary to bring out both the concepts of process and object by interweaving them: objects contain visual models which, in turn, refer to other objects’ attributes and methods, in a chain-like fashion. So, while the approach we advocate is object-oriented in the sense that we organize all our knowledge by developing classes and objects first, *process* has equal footing in the form of visually defined dynamic models.
3. *Completeness*: The new taxonomy organizes models from several different areas including continuous, discrete and combined models under one umbrella. The traditional modeling taxonomies are currently separate. For example, models that require a continuous time slicing type of model execution, are grouped into a model category based on the need for time slicing, not the form of the model. We focus, instead, on visual model structure as a basis for model design organization.
4. *Multimodels*: There does not exist a good modeling approach to multi-level models where levels are defined using heterogeneous model types. The new taxonomy addresses this problem through the multimodel concept.
5. *Design versus Execution*: A taxonomy for modeling should be based on the design of a model and not how it is executed. The current taxonomy introduces some ambiguity as

to whether design or execution is being used to categorize models. The new taxonomy is one which stresses model form as *graphical structure* where possible.

6. *Dynamic Models versus Programs*: We draw a clear parallel between the new taxonomy for dynamic models and programming language categories in computer science. The ability to use the same categories (for modeling as well as for programming) lends credibility to the new taxonomy, and allows one to draw direct parallels between programming language and model design constructs without inventing new modeling category types.

While there has been significant coverage in the simulation literature for analysis methods [2, 25], the general area of modeling for simulation has lacked uniformity and in-depth coverage. Two areas of modeling termed “discrete event” and “continuous” are defined in the simulation literature. For discrete event models, the field is sub-divided into *event-oriented*, *process* and *activity-based* modeling. To choose one of these sub-categories, we might ask “What is an event-oriented model?” There is no clear definition [2, 25] other than to state that a discrete event model is one where discrete events predominate. There is no attempt to further categorize or classify the *form* taken on by an event-oriented model. In mentioning form, we need to address the differences between syntax (form) and semantics (execution). A program or model may be of a particular form; however, the semantics of this form may have a variety of possibilities. A Petri net [38] has a particular form regardless of the way in which it is executed. Ideally, then, we would like to create a model category that classifies the form of the Petri net, apart from its potential execution characteristics. By associating integer delay time with Petri net transitions, one can execute the Petri net using time slicing, discrete event simulation or parallel and distributed simulation. By providing an “event-oriented” model category, it is not clear whether this includes only those models which have explicitly surfaced “events” in their forms (as in event graphs [48] or animation scripts [13]) or whether a GPSS or Simscript program [34] could be considered an event-oriented model. Our approach is to clearly separate model design (syntax) from execution (semantics). Moreover, as stated earlier, programs are not considered to be models at least for most textually-based programming languages. One can attach semantics to syntax, but they remain orthogonal concepts.

Some model types that are similar in form are unfortunately separated into different categories using the traditional terminology. An example of this can be found in block models for automatic control and queuing networks. A functional block model and a queuing network model are identical in form, the only differences being in the semantics for the blocks (i.e., transfer functions) and the nature of the signal flowing through the blocks (discrete or continuous). Our taxonomy stresses a difference in model topology and structure instead of separating model types based on time advance or signal processing features. By using the concept of *functional model*, we characterize the syntactical form of the model: functional models are identified by a uni-directional flow through a network of nodes through directed arcs. In this fashion, control networks and queuing networks are of the same model type. Likewise, this flow is directly analogous to functional composition in functional programming languages.

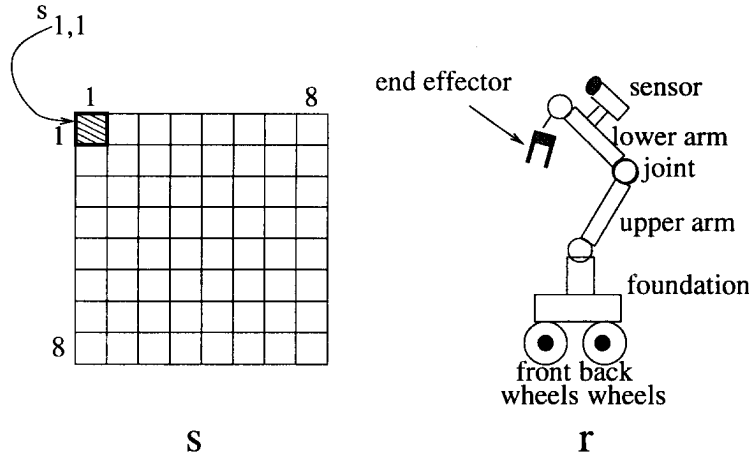


Figure 2: Scenario for generic object behavior.

3 Scenario

A general scenario should be defined so we can develop the concepts of physically-based object-oriented design. We will create a simple example and then provide a table that illustrates how this example can be seen for a wide variety of disciplines. The reason for this choice of scenario is that it captures the essence of physical modeling: the application of dynamics and geometry using particles and fields. Even in the domain of sub-atomic particles, the object orientation is relevant since particles and fields integrate to form objects via Schroedinger's wave equation. The robot serves the role of a particle and the space (or landscape) serves the role of space. Together, they provide for a comprehensive model. Consider a 2D space s that is partitioned using either a quadtree or array (ref. Sec 6.1). A set of mobile robots move around s , undergoing change, as well as changing attributes of s . Fig. 2 illustrates s and a sample robot r . In the remainder of the paper, we will refer to Fig. 2 using classes, objects, attributes and methods defined in Sec. 4. A robot or set of robots move around space s , some staying within the confines of a particular partition of s , such as $s_{i,j}$ for $i, j \in \{1, \dots, 8\}$. Moreover, certain attributes of s may change. For example, there may be water in s or a particular density of matter assigned to s . Our robots will be unusual in that they are capable of changing shape over time if the dynamics demand this of them. Before we embark on a discussion of object-oriented physical design for robots within spaces, we present Table 1 to illustrate how, through mappings from one discipline to another, different areas fit into this general scenario scheme.

4 Model Engineering

Our basis for physical modeling begins with object-oriented design concepts as described in textbooks [5, 45] as well as object-oriented modeling as applied specifically for simulation of discrete event systems [57]. Model engineering is the process of building static and dynamic models for a physical scenario using our extended object-oriented framework. The steps we take in this procedure are shown in Fig. 3.

Table 1: A sample set of applications using a particle-field metaphor.

Application	Particle	Field
Cybernetics (intelligent agents)	robot	space (room, factory floor, terrain)
Military (Air Force)	plane, squadron	air space
Ecology	individuals, species	landscape
Materials	particles, molecules	fluid (air, liquid)
Computer Engineering	chip, module	N/A
Quantum Mechanics	wave function	wave function
Meteorology	hurricane, tornado (finite volumes)	atmosphere

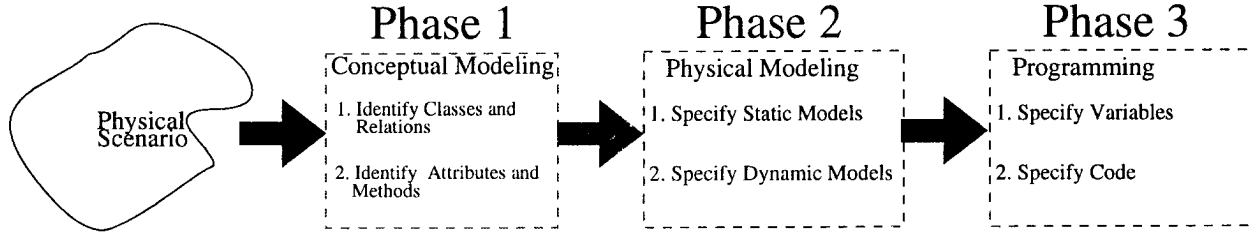
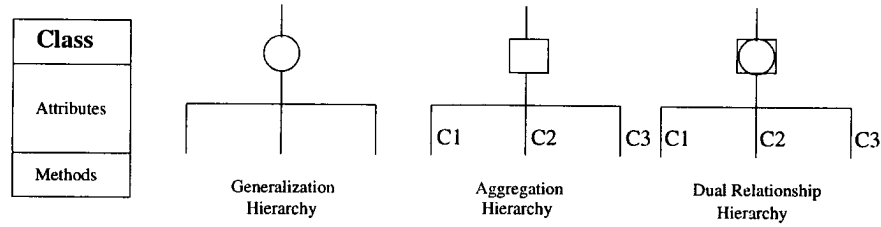


Figure 3: Model engineering.

5 Conceptual Modeling

The first phase is constructing a conceptual model of the physical scenario. To build such a model, we must construct a class graph with relations among the classes. Furthermore, we must identify attributes and methods in those classes. A class is a type. Our treatment of a class is as if it served as a “cookie cutter.” A cookie cutter (class) operates over a sheet of dough to create cookies (objects). We might create several types of robots: *Walking*, *Rotating*, *Fixed-base*. Each of these are classes and they are sub-classes of robot since all of them are types of robots. This particular relation is called *generalization*. Another kind of useful relation is called *aggregation* since it involves a relation among classes where there is a “part of” relationship. For example, a particular robot may be composed of (or aggregated from) wheels, an arm and a camera. The base, arm and camera are part of the robot: the robot is an aggregate of the base, arm and camera. Fig. 4 shows how we illustrate both of these relations: generalization with a circle and aggregation with a square. We also permit an analyst to specify any given relation as both aggregation and generalization. This is delineated with a circle inside a square. The C specified in Fig. 4 can specify cardinality for



Note:
 C_i = cardinality constraint such as " $=4$ " or " <2 "

Figure 4: Structure of a class with three relations.

a class. In general, without such a specification, it is assumed that a class can be composed of any number of objects of the sub-class. However, let's say that we create a class *Room* which will always have four walls, then we can specify $= 4$ on the aggregation relation arc to illustrate this constraint. Without this explicit constraint, a *Room* can be composed of any number of walls. This approach is consistent with several existing OO approaches [45, 22] to aggregation specification.

While we are on the subject of classes, we define an object to be an instance of a class. A particular *wheel* is an instance of the class called *Wheel*. A class is a set of objects, which are related through the class definition. A class is composed of its name, a set of attributes and a set of methods. Aggregation among classes requires some clarification. If The set of *Wheels* for a robot is aggregation from the classes *FrontW* and *BackW*, there may be any number of front wheels and any number of back wheels. The aggregation just shows the class aggregation and not the object aggregation. The specific number of wheels is something that is changed as we create instances of the two classes. An actual object called *wheels* can be created from class *Wheels* and then we can create two objects from *FrontW* and two from *BackW*. We can even create a containment model (or data structure) which we locate as an attribute value within *wheels* that shows the composition of this particular *wheels* object.

A key part of conceptual modeling is identifying the classes. For the most part, this procedure is ill-defined but some rules and approaches do exist [13, 18] to help in the model engineering process. Natural language provides one basis on which to base choices for classes, attributes and methods. The following are heuristics to aid in the creation of the conceptual model from a textual description of a physical scenario:

- *Make nouns classes or instances of a class.*
- *Use adjectives to make class attributes, sub-classes or instances.*
- *Make transitive verbs methods which respond to inputs.*
- *Use intransitive verbs to specify attributes.*

In the physical sciences and engineering, we use models to describe the shape of objects as well as their behavior. We call the combination of attributes and methods *structure*. The two types of relations among classes, *generalization* and *aggregation*, are very popular and are frequently used in object-oriented design. The reason for their utility and popularity is that they involve the implicit act of passing structure from one class to another. Structure

passing is powerful and enables us to fragment the world into classes, while designing common structure through aggregation and generalization relations.

The passing of structure for generalization is top to bottom, of a hierarchy of classes related via generalization, and is frequently known as inheritance. Let's consider an orange. An *Orange* class inherits certain attributes and methods (i.e., structure) from the *Fruit* class since an orange is a kind of fruit. A walking robot is a type of general robot, and so inherits structure from the general robot. The uppermost classes in a generalization hierarchy are *base* classes and the child classes are *derived* classes from the particular base class.

We have discussed generalization and its associated structure-passing inheritance capability, but there is another key kind of relation: aggregation. Aggregation enjoys the benefit of structure passing also, but the structure passing in aggregation is bottom-up instead of top-down. A class that is an aggregation of classes underneath it captures the structure of all of its children. A robot contains all attributes and methods associated with each of its sub-components, such as links, cameras and the behaviors of those sub-components. As we use *inheritance* for generalization, we will use *composition* for aggregation. Structure passing is done, therefore, through inheritance and composition. For our discussion of generalization and aggregation, we assume that structure passing is a *logical* operation; however, it may not be directly implemented in a specific programming language or implementation. For example, a large 10^6 square cell space is an aggregate of 10^6 individual cells; an implementation may choose not to cause the explicit passing of structure from children (i.e., cell) to parent (i.e., space), nevertheless, the structure passing is a logical consequence of aggregation, and is *logically* present in our design, if not in our implementation.

Inheritance and composition are further defined as follows:

- Inheritance (or generalization) is the relational property of a generalization hierarchy. Composition (or aggregation) is the relational property of an aggregation hierarchy.
- To differentiate between layers in a hierarchy we use the terms “child” and “parent.” A parent is always above a child regardless of the relation type. Therefore a child class in generalization inherits from its parent, but a parent class aggregates from its children.
- The words “derived” and “base” are *relative* to the type of relation. Base classes in a generalization tree are at the top with lower-level classes deriving structure. For aggregation, it is the opposite, with the base classes being at the leaves of the tree. Structure passing for both relations is derived from “base” to “derived” classes.
- The only classes that can be used for constructing objects are the leaf classes of a generalization hierarchy. Internal tree nodes are used to hold class structure but are not used for object construction.
- Inheritance occurs when a derived leaf class in a generalization class hierarchy is used to construct or create an object. The object “inherits” all attributes and methods from its parent (or *parents* in multiple inheritance).
- Composition occurs when any class in an aggregation class hierarchy is used to construct or create an object. The object passes all structure assigned to it upward to the

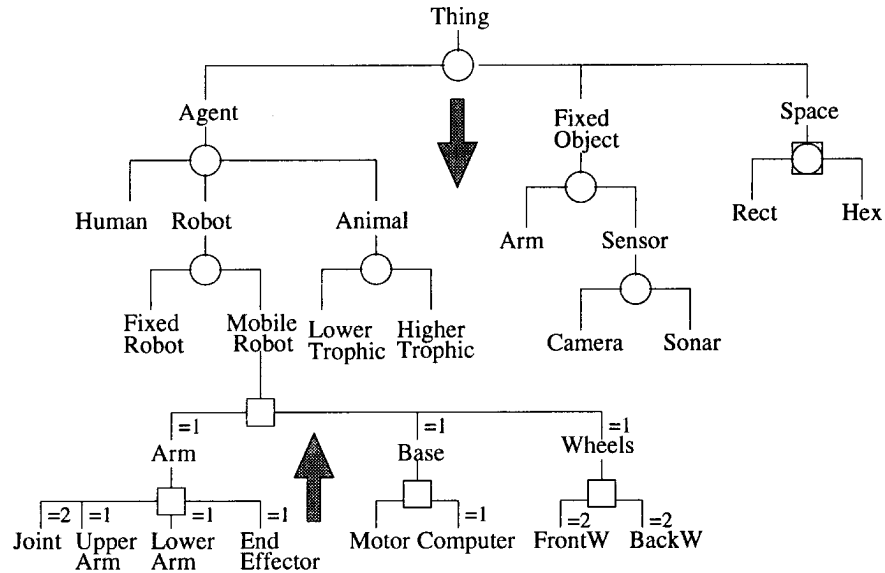


Figure 5: Phase 1, Step 1: Identify classes and relations.

parent (or *parents*) in multiple composition). The derivation of structure moves from the base classes.

Aggregation and containment are two different, but related concepts. Aggregation involves composition, which means that a class is composed of sub-classes. A glass of marbles will contain marbles, but the glass is not composed of marbles and so a *Marble* class, in an aggregation sense, is not a sub-class of *Glass* unless one redefines the meaning of *Glass* to encompass not only the physical structure of a glass, but also of everything within the “scope” or “environment” of the glass.

Generalization and aggregation are the key relations used in building a conceptual model, but they are not the only types of relations. If our physical scenario was such that all robots were attached to wooden boards, then one could form a relation arc between the class robot and the class board. However, there are sometimes better mechanisms for handling such cases. For the robot and the board, one can form an aggregate class called *Robot-env* which aggregates both board and robot classes. Some of these other class relations may or may not involve structure passing, but generalization and aggregation represent the power of a transitive structure-passing relation involving any number of hierarchical levels. In any event, by allowing arbitrary relations among classes, we generalize conceptual models to have similar capabilities in representation to that of semantic networks and certain schemata in databases. That is, one can use logical inference and querying on the conceptual model in addition to using it only for structure passing. Also, the conceptual model need not be static. The conceptual model as it is originally defined represents a physical system at an initial time instant. New classes and relations may be added over time to permit a dynamically changing physical environment.

Fig. 4 illustrates generalization (○) and aggregation (□) relations. It is not necessary to group all relations into one graph or hierarchy—multiple graphs or hierarchies are possible. Fig. 2 provides us with the base classes for Fig. 5. The downward and upward block

arrows in Fig. 5 illustrates the respective directions of structure passing for inheritance and composition.

Inheritance and aggregation have *rules* that they use to perform the movement of structure within a tree. For inheritance, methods and attributes are copied from the tree root to the tree leaves except for when one overrides an inherited attribute or method. However, we need to be explicit about our “inheritance rule” since copies can be made, potentially, not only from the root of the generalization tree, or from the immediate parent of a class, but also from any class which lay in-between the root and leaf class. An attribute *type* within class *Agent* can be set to “organic”. While the classes *Human* and *Animal* automatically inherit this attribute from *Agent*, *Robot* overrides this by setting *type* to “artificial.” Overriding method and attributes permits a kind of heterogeneity in the derived classes so that they need not all be perfect copies of the base class. In addition to overriding, some structure from the base class may not be available for copying to derived classes. A default inheritance rule is one where a derived class inherits structure through *copying* from its parent class.

For aggregation, we have a more complex situation where an aggregation procedure must be specified for all attributes and methods in the base classes. An example of the need for such procedures is when two base-class methods or attributes are identical in name. The aggregation question is framed as “Given a number of base classes, how do we glue the base class attributes and methods to create attributes and methods in the aggregate class?” There is a conflict and a resolution method is required. Consider attribute *contains* in *Arm*, *Base* and *Wheels*. The *contains* attribute points to a data structure of what is contained within an object. Through composition, *Arm* obtains all three of these *contains* structures but what is *Arm* to do with them since they are all of the same attribute name? A logical aggregation procedure here is to say that all sub-classes of *Arm* with a *contains* attribute are grouped together into a record or array which is then placed in *Arm*. However, this sort of aggregation is not always appropriate. If the sub-classes contain an attribute *count* which specifies how many objects there are of this class, then the correct aggregation rule for *count* within *Arm* involves a sum of all *count* attributes in the sub-classes of *Arm*. In aggregation, some sort of “aggregation rule” is *always* necessary. Implicitly, one could define that attributes of different names simply agglomerate into aggregate objects in a set-union fashion. However, there are many instances where this is not so. The *count* attribute is just one example. Other examples include aggregation into a matrix or array, and aggregation via model component “coupling” composing a dynamic model of sub-object methods. Rules can be stored in their respective classes. We make no attempt to formalize the rule structure—only to state that some code or rules should be available within a class to handle all aggregations that occur. A default aggregation rule is one where a derived class aggregates structure through *set union* of the child classes. That is—they just collect structure together without resolving conflicts, merging structure through summation or integration, or performing concatenation of structure.

As to how we might refer to populations or groups versus individuals, we consider the motor example. The set of motors can be called *motor* which points to a data structure specifying motor objects, while an individual motor requires an index such as *motor*[2]. When an object is created that uses the same root name for an object that already exists, such as when one created object *motor*[2] after having created *motor*, then the a hierarchy is assumed and aggregation occurs as a result. This mechanism allows one to attach recursive,

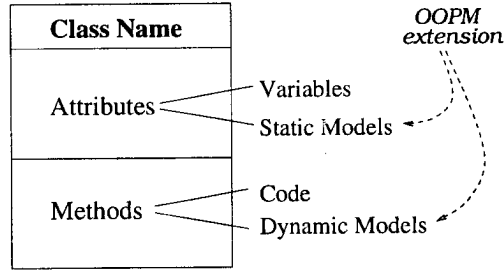


Figure 6: Structure of a Class.

hierarchical properties to any class in the conceptual model without explicitly specifying these properties at the time of conceptual model formation. Instead, this sort of multi-level hierarchy is defined as a static model. An example of this can be seen with the class *Rect* in Fig. 5. This class can be used to create a rectangular space hierarchy of any dimension. Let's first create a space object called *cell* by defining *Rect cell*. If we wish to structure this space into a quadtree, for instance, we can then create four new objects *cell[0]*, *cell[1]*, *cell[2]* and *cell[3]*. Since we used the same name *cell*, there is an automatic aggregation relation with *cell* composed of *cell[0]*, *cell[1]*, *cell[2]* and *cell[3]*. Specifically, aggregation rules come into play as previously described. The actual quadtree would be stored as a static model of *cell*. The explicit creation of aggregation hierarchies within the conceptual model is dictated by a heterogeneous aggregation relation. If a robot arm consists of exactly two links, which are different in nature, then this aggregation relation belongs in the conceptual model. However, the potentially infinite recursion of spatial decomposition suggests a homogeneous aggregate relation, and is relegated to a static model stored within a "space object."

OOPM specifies that an attribute is one of two types: variable or *static model*. Likewise, a method is one of two types: code or *dynamic model*. A method can be of a functional (representing a function) or constraint (representing a relation) nature. Once the conceptual model has been constructed, we identify the attributes and methods for each class. An attribute is a *variable*, whose value is one of the common data types—or a *static model*. A method can be *code*, whose form depends on the programming language, or a *dynamic model*. The structure of a class is seen in Fig. 6. Variables and code are described in OO languages such as C++ [49]. We define a static model as a graph of objects and a dynamic model as a graph of attributes and methods. The model types of interest here are dynamic. However, the concept of static model complements the concept of dynamic model: methods operate on attributes to effect change in an object. Dynamic models operate on static models and variable attributes to effect change. We will use the following notation in discussing object-oriented terms. When we speak of a class, we capitalize the first letter, as in *Robot* or *Arm*. An object is lower case. An attribute that is a variable is lower case, whereas an attribute that is a static model is upper case for the first letter. A similar convention is followed for methods: a code method uses lower case with a parentheses "()" as a suffix; a dynamic model method is the same but with a capitalized first letter. Classes are separated from objects with a double colon "::" whereas objects are separated from attributes and methods using a period "." This convention is similar to the C++ language and is a convenience when communicating conceptual models textually. All classes, objects, attributes, and methods

Agent	Arm	Mobile Robot	Space
Shape plan position	position Geometry	Geometry	Cell_Array
report() Execute()	Rotate()	Move() Reach()	Diffuse()

Figure 7: Phase 1, Step 2: Identify attributes and methods.

will use an italics font to differentiate them from surrounding text.

To provide some examples:

- A four-foot high robot *r1* can be represented as: *Mobile_Robot::r1.height = 48.0*, where *height* is specified in inches. Alternatively, we can simply leave out the class name: *r1.height = 48.0*, since we know that *r1* is an instance of *Mobile_Robot* by reviewing the conceptual model. If there is a static model of a robot, in the form of a computer aided design representation, then we would refer to this as *r1.Geometry*.
- A robot arm with a revolute joint will rotate, so we can create *arm.angle* as an attribute of object arm and also *arm.rotate()* as a code method of arm that changes *arm.angle*. If the dynamics of rotation are captured in a dynamic model Rotate, then we have *arm.Rotate()* as my dynamic model. Object *arm* relates to object *r* in that the robot class is an aggregate class containing sub-classes such as *Arm*. Moreover, an attribute inside *r* called *connectivity*, with a linked list structure, would include *arm*. Should the linked list *connectivity* be a static model or a variable of *r*? This is ambiguous and under the user's control. A general heuristic is that if an attribute contains a data structure specifying geometry or relative position of sub-objects, then we call it a model as opposed to a variable, but a case could be made either way.
- A *landscape* is an object that aggregates an array of cells or patches. The dynamics of *r*, that moves within a cell, may be coded as *r.Move()*, a dynamical model or simply as *r.move()*, a piece of code. A landscape subcell, *landscape.cell[2,3]* will contain some number of robot objects using the appropriate data structure.

Fig. 7 illustrates a subset of the objects shown in Fig. 5. For this subset, we identify some attributes and methods that we feel are necessary in simulating the scenario. At this point, it is not necessary to identify the precise structure of each attribute and method since this is part of the conceptual modeling phase. In drawing the attribute and methods, it is useful to recreate Fig. 5 with the *expanded* class nodes shown in Fig. 7. Since this diagram would be large for all classes in Fig. 5, we are illustrating a subset of all class nodes. We use the scheme previously discussed to differentiate models from variables and code: models begin with a capital letter and methods end with a pair of parenthesis “()”.

Fig. 8 displays another robot-oriented conceptual model to illustrate some of the points we've made about generalization and abstraction. We use the following new acronyms: *DigitalTech* for “digital technology,” *DSPChip* for “digital signal processing chip,” and *Mux* for multiplexer. For each relation, we need to have a procedure. We'll proceed from left to right.

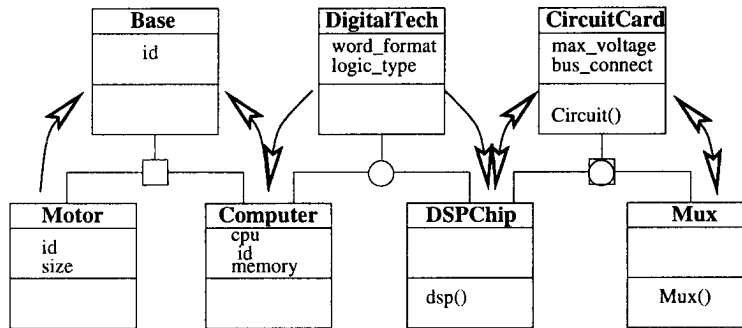


Figure 8: Generic generalization and aggregation scenario.

- Relation 1 (Aggregation):

1. By default, *Base* will aggregate all methods and attributes using the set operation *union* ($\cup$) unless otherwise specified. We'll use this default to pass up all attributes of *Motor* and *Computer* except for *id*.
2. *Base.id* is formed by *Motor.id* and *Computer.id* by concatenating them in a vector $\langle \text{Base.id}, \text{Motor.id} \rangle$.

- Relation 2 (Generalization):

1. By default, *Computer* and *DSPChip* inherit all structure from *DigitalTech* through copying. We'll adopt the default on this one.

- Relation 3 (Dual):

1. The default structure passing for this is the same as for both aggregation and generalization. We are stating that a *CircuitCard* serves as a composition of *DSPChip* and *Mux* as well as stating that *DSPChip* and *Mux* are *types* of *CircuitCard*.
2. For aggregation, we specify model *Circuit()* as being defined by a functional coupling of *dsp()* and *Mux()*.
3. For generalization, we use the default for inheriting *max_voltage*, but override the inheritance for *bus_connect* since the bus connection is an attribute of the circuit card and not relevant to *DSPChip* or *Mux*.

We've seen that generalization and aggregation provide us with power for structure passing, but that we require both default procedures as well as special procedures which either limit the structure passing in a particular way or accurately define it.

In object-oriented design, there are certain key characteristics that one must employ throughout the design process:

- *Coupling*: In class hierarchies and graphs, classes are coupled together via relations, most of which are aggregation or generalization. Coupling also extends to static (ref. Sec. 6.1) and dynamic (ref. Sec. 6.2) models where objects, attributes and methods are coupled together in graph form to create a model. Coupling provides the glue used to bring classes and other object-oriented features together.

- *Hierarchy*: When a relation can be applied transitively (*Camera* is a kind of *Sensor* that is, in turn, a kind of *Fixed.Object*) then this provides a convenient ordering of knowledge. Furthermore, the transitive generalization and aggregation relations permit the passing of class structure down and up hierarchies. Hierarchy plays a key role in modeling as well with components having sub-components. Sub-component children can be of the same type as the parent component (homogeneous hierarchy) or they can be of different types (heterogeneous hierarchy).
- *Encapsulation*: Where does a particular model belong? To help make this decision, we use the following rule: a model (static or dynamic) is encapsulated within the most specific class or object that contains all attributes and methods defined in that model. This is discussed in detail within Sec. 6.2.4.

Coupling provides the basis for sticking object-oriented components (class, object, attribute, method) together, whereas hierarchy and encapsulation provide ways in which the coupled components can be efficiently managed.

To summarize the conceptual modeling phase, we construct classes and relations among classes. Two key hierarchical, recursive, relations that involve structure passing are generalization and aggregation. The conceptual model exists solely as a knowledge representation of a physical system, and to permit operations such as structure passing and logical inference.

6 Physical Modeling

6.1 Static Modeling

The word “static” in static models refers to the inability of the model to cause change of attribute; it does not mean that the model doesn’t change. For physical modeling, our primary type of static model is one that specifies the topology or geometry of a physical object such as *r*. However, a semantic net [53], would be an equally valid static model. Dynamic models (ref. Sec. 6.2) have the ability to change static models over time. The previously discussed conceptual model, composed of classes and relations, can also be seen as dynamically changing with class relations changing over time. If this occurs then it is logical to create an all-encompassing class called *universe* and then make the conceptual model an attribute of *universe* in the form of a static model.

For modeling geometry and space, there are a number of representational techniques, many of which are discussed by Samet in two volumes [47, 46]. We will not create any extensions of static modeling methods. Instead, for our scenario conceptual model in Fig. 2, we’ll discuss our alternatives with an example or two. In Fig. 2 we have two items: a space *s* where robots behave. Space *s* can be modeled a simple square array, which hardly can be classified as a model except that it is a model considering that it is an abstraction of a physical object (i.e., a physical space). Beyond this straightforward model, it is often useful to model space using varying degrees of resolution depending on the area of concern. Areas of space with a sparse density of robots, for example, might be modeled “in the large” whereas dense areas are subdivided hierarchically. A quadtree represents a simple form of four-ply tree data structure that can be used to model the space. For 3D spaces, octrees provide a related structure. Likewise, for the robot depicted in Fig. 2, there are methods

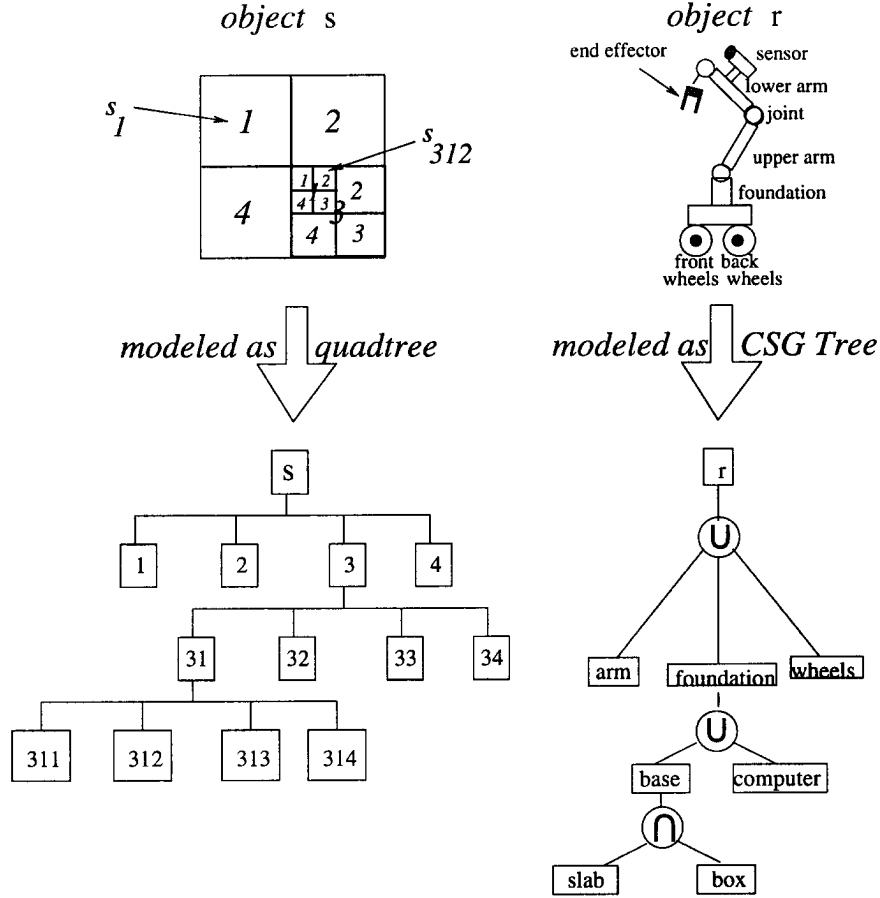
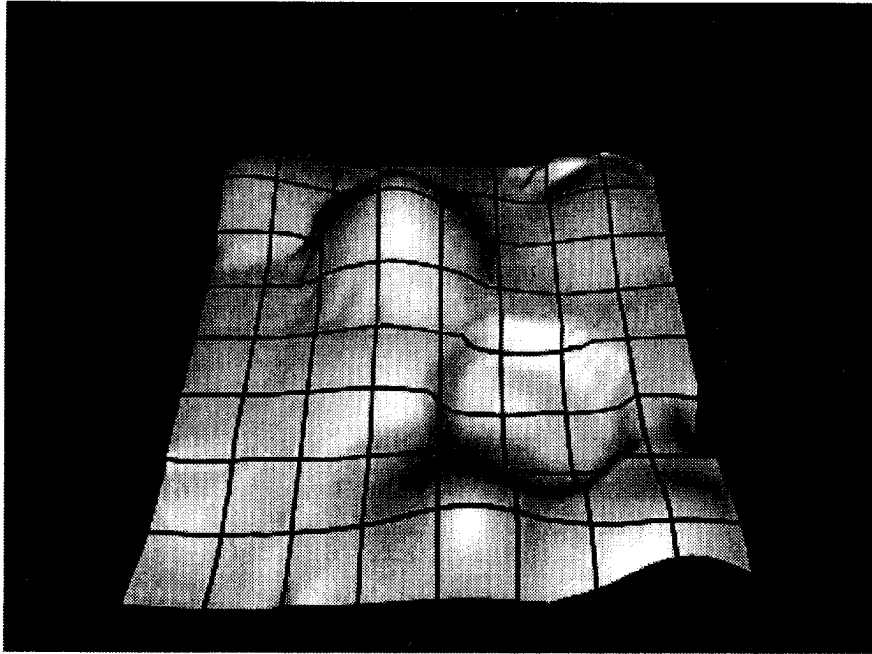


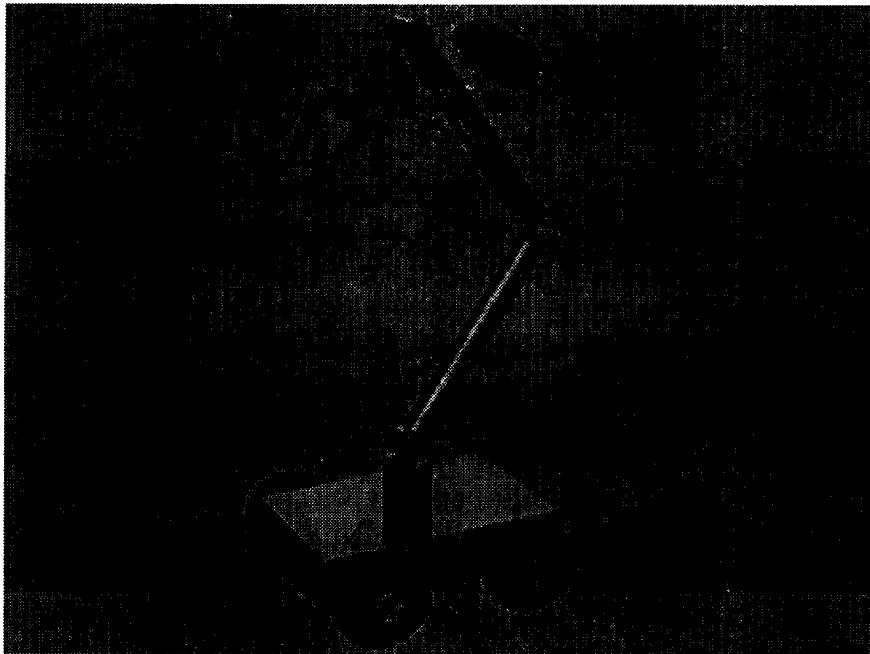
Figure 9: Phase 2, Step 1: Static models for space s and robot r .

described in the literature on computer aided design and computer graphics [17, 20]. Fig. 9 displays two static models: the first model is a quadtree of space s and the second model is a constructive solid geometry model of robot r .

The quadtree is composed entirely of objects, which when aggregated, form s . An object can contain its own static models or an attribute called *contains* that refers to a list structure of robots found in the object. Therefore, s_1 and s_{312} are sample objects that may contain several robots. The reason for sub-partitioning a particular cell is that, depending on the density of robots in a particular part of s , we may want to sub-divide our space. Another reason is that if s represents a landscape, we may wish to focus our modeling resources in a particular sub-cell within s , while still maintaining a partition of s . The CSG tree on the right part of Fig. 9 provides a structure for the topology of r . The symbol $\cup$ denotes union and $\cap$ denotes intersection. A CSG tree contains two types of nodes: operations and objects. For example r is an object composed of the union (operation) of objects *arm*, *foundation* and *wheels*. A rectangular *slab*, when intersected with a cubic *box*, creates the *base* of r . Figs. 10(a) and 10(b) display 3D geometric static models for both s and r , reflecting a more realistic scenario configuration suitable for animation and immersive situations.



(a) Geometry for space s .



(b) Geometry for robot r .

Figure 10: Geometry representing static models for the scenario.

6.2 Dynamic Modeling

6.2.1 Overview

A dynamic model captures the way in which attributes change over time. At first glance, it may appear that one can create a dynamic object-oriented model by linking together objects in a graph. For example, supposing that robot r_1 gives food to r_2 . A temptation is to draw an arrow from one object (r_1) to the other (r_2) and claim this as a dynamic model. In traditional object oriented design, this *message passing* approach is specified as a way to model the “behavior” of objects interacting with one another via messages passed from an object to another. Unfortunately, while such a graph represents our intention of expressing dynamics, it contains no information as to the underlying dynamics. The primary problem is that we do not know which methods of r_1 or r_2 to use. Suppose that r_1 has many potential dynamic models. Which particular dynamic model should we use in our “object” model? The specification of an object pointing to another object is not sufficiently defined to be of any real use for simulation. In effect, a graph containing the two objects is a static model, not a dynamic one, since the graph depicts a geometric or semantic relation: one robot connected to another. To accurately represent the dynamics of objects, we need a more comprehensive and flexible approach that affords the modeler the ability to use familiar models such as FSAs, System Dynamics graphs, compartmental flow models and block models.

6.2.2 Three types of Dynamic Model

The three model types that have strong ties with programming languages are: declarative, functional and constraint. A declarative simulation model is one where states and event transitions (individually or in groups) are specified in the model directly. Production rule languages and logic-based languages based on Horn clauses (such as Prolog [24]) create a mirror image of the declarative model for simulation. Moreover, declarative semantics are used to define the interpretation of programming language statements. A functional model is one where there is directionality in flow of a signal (whether discrete or continuous). The flow has a source, several possible sinks, and contains coupled components through which material flows. Functional languages, often based on the lambda calculus [31, 40], are similar in principle. If programming language statements are not viewed declaratively, they usually are defined using functional semantics. The languages Lisp [51] and ML [37] are two example functional languages. Lisp has some declarative features (side effects) whereas other functional languages attempt to be “pure.” Finally, with regard to computer science metaphors, constraint languages [6, 29] reflect a way of programming where procedures and declarations are insufficient. The constraint language CLP($\mathcal{R}$) [21] (Constraint Logic Programming) represents this type of language. Also, the next generation Prolog (Prolog III) is constraint oriented. In constraint models, the focus is on a model structure, which involves basic balances of units such as momentum and energy.

Fig. 11 illustrates the dynamic modeling taxonomy. The top level of Fig. 11 refers to the multimodel type (ref. Sec. 6.3) since this type is composed of all sub-types previously discussed by using hierarchical refinement: declarative, functional, constraint and spatial. Conceptual models are generated before multimodels since conceptual models are non-executable and reflect relations among classes. Each of these sub-types has two sub-categories:

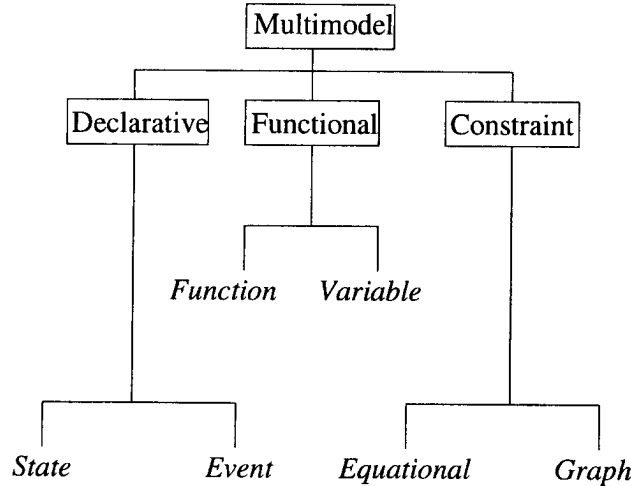


Figure 11: Model taxonomy.

- *Declarative* models focus on patterns associated with states or events. An example declarative model type with a state focus is the finite state automaton. The event graph is an example with the event focus.
- *Functional* models are networks whose main components are either functions (as in block models) or variables (as in the levels found in systems dynamics).
- *Constraint* models are represented as equation sets or as graphs. An example constraint graph is an analog electrical circuit or a bond graph [7].

The extra model type not previously discussed here (but found in Fig. 1 and [13]) is *spatial* model. In our discussion, a spatial model is a static model whose dynamics take on one of three primitive types. So, we do not afford spatial models any special status since in OOPM, they are artifacts of the methodology and of aggregation relations.

6.2.3 Definitions

To define how dynamical systems are embedded within OOPM, we need to address some fundamental systems theoretic concepts. A time invariant system ψ can be abstracted as follows: $\psi = \langle I, O, Q, \Omega, \delta, \lambda \rangle$. The sets I and O represent input and output sets. Q defines the state space for the system. Ω represents the admissible set of input trajectories, δ is the state transition function, and λ is the output function which is generally a function of Q . There are internal and external events. An external event is one from “outside the system” whereas an internal event is one “inside the system” (but from a lower abstraction level). Further explanation and variations of the system formalism can be found in the systems [36] and simulation [55] literature; however, the above definition suffices for our purpose. The first key observation of OOPM is that we are encapsulating behavior (dynamic models) and structure (static models) within objects. This represents a structured representation of a system as opposed to an all encompassing system definition with a multi-dimensional state space spanning many objects. For the aggregate objects, however, this large state space is

accurately captured since these accrue fewer-dimensional state spaces of sub-objects through *composition*. The following definitions are presented to bridge the gap between the formal system components and our components. A class C_i is defined as a sub-class of C in an aggregation relation; likewise, a class C^i is defined as an aggregate object composing C as a sub-class. We let $\hat{C}^i = C \cup C^i$ and $\hat{C}_i = C \cup C_i$ for notational convenience. A similar notation can be constructed for objects O .

- *State: A state of class C is contained within C 's attribute list.* Comments: objects that are leaves in an aggregation hierarchy will have low-dimensional state spaces with internal and root nodes aggregating these low dimensional spaces. The state of object *arm* for the robot may include a *position* for the centroid as well as an orientation θ . The arm contains all states in Arm_i through composition.
- *Event: An event is classified as internal or external.*
 1. *An internal event for C is an input from within $\hat{C}_i$.* Comments: all internal events within C are inputs from C or sub-classes of C . A clock has an internal event when the alarm rings, but the alarm mechanism is a sub-class C_i of C and the event is an output from C_i . The computer for robot r is a sub-object of the robot. The computer may periodically produce internal (relative to r) events that are employed in a dynamical model in r).
 2. *An external event for class C is an input from a class in C^i .* Comments: An external event to an alarm clock comes from the human hand or finger object that presses a button to stop the alarm. All environmental activities in s affect r through external events, since the environment is outside of r .
- *Input: All inputs to C are either internal or external events.*
- *Output: All outputs from C are inputs to some P with the exception of a sink node, for which an output employs a method of C using the C 's state attributes.*

These definitions will help us in formulating common templates for dynamical models. A primitive dynamical model is of three types: declarative, functional or constraint. For each of these model categories, there are some common types that we define with templates.

1. **Declarative:** A finite state automaton (FSA) is defined with nodes and arcs. A node is a state of C , and therefore an attribute of C . The arc contains a boolean-valued method $p()$ with arguments that are internal or external events. An event graph is defined with nodes and arcs, as for an FSA. A node is an event (internal or external). An external event relative to C is an attribute of C^i and an internal event is an attribute of $\hat{C}_i$. The arc in an event graph represents a method that schedules or causes the event on the head of the arc. The robot r may be in one of three states: *active*, *stationary* or *maintenance*. The *maintenance* phase is used for fixing the robot's position via a satellite. Therefore, a method is created in r that points to a dynamic model: an FSA with three states. Arcs from one state to another in an FSA are boolean predicates.

2. **Functional:** A block model has nodes and arcs. A node is a method of C , and the arc represents a *directed* connection from one method to another. Both methods can be found in $\hat{C}_i$. The block model is function-based since functions are made explicit as nodes. Variable-based models such as System Dynamics [42] or compartmental models [23] are the duals of function-based models since variables are placed at the nodes. For a C with this type of method, the variables are attributes of $\hat{C}_i$. Functions are often assumed to be linear, but if they are defined, they are methods found in $\hat{C}_i$. For our scenario we would have to derive a directional activity to use a functional model. One such activity is the movement of water on s over cells in a specific direction, such as via a canal or along a river. Such boundary conditions suggest a functional model, since without these boundaries, we would use a constraint model defined by dynamics incorporating local conservation laws. Another activity is if robots cooperate with each other to form a pipeline task, passing a product or food from one end of the robot “chain” to the other. This activity is common in workstation cells in a manufacturing scenario where robots are often fixed relative to each other in space.
3. **Constraint:** Constraint models are equational in nature, and reflect non-directional constraints among object attributes. A class C with a constraint equational model contains an equation with terms containing attributes of $\hat{C}_i$. Equations can be represented in graph form as well as with bond graphs [7, 43, 50]. Models of non-directional behavior, such as general hydrodynamic models are constraint-based. If all robots interact with each other in ways dictated by nearest-neighbor conditions, for example, this is modeled as a constraint since there is no consistent, time-invariant direction associated with the dynamics.

6.2.4 Representing Dynamic Models

How are dynamic model components represented in the physical modeling methodology? We will illustrate two model types (functional and declarative), each with two model sub-types. For the functional model types, we use a block model and a System Dynamics model. For the declarative model type, we will use an FSA and a Petri net. The following notation, consistent with the previous notations, will be used throughout this discussion:

- **Objects:** An object is represented as obj . obj_i represents a sub-object of obj (in its aggregation hierarchy), and obj^i represents a super-object which is composed, in part, of obj . When indices i and j are used, it is possible that $i = j$ or $i \neq j$. This relation rests with the particular application.
- **Attributes:** $obj.a$ represents a variable attribute and $obj.A$ represents a static model attribute. a is short for any string beginning with a lower case letter; A is short for any string beginning with an upper case letter. Attribute references (i.e. names) and values are relevant: a name is just $obj.a$ whereas the value of attribute a is denoted as $v(obj.a)$. The following special attributes are defined for all objects: $obj.input$, $obj.output$ and $obj.state$ and represent the input, output and state of an object at the current time.

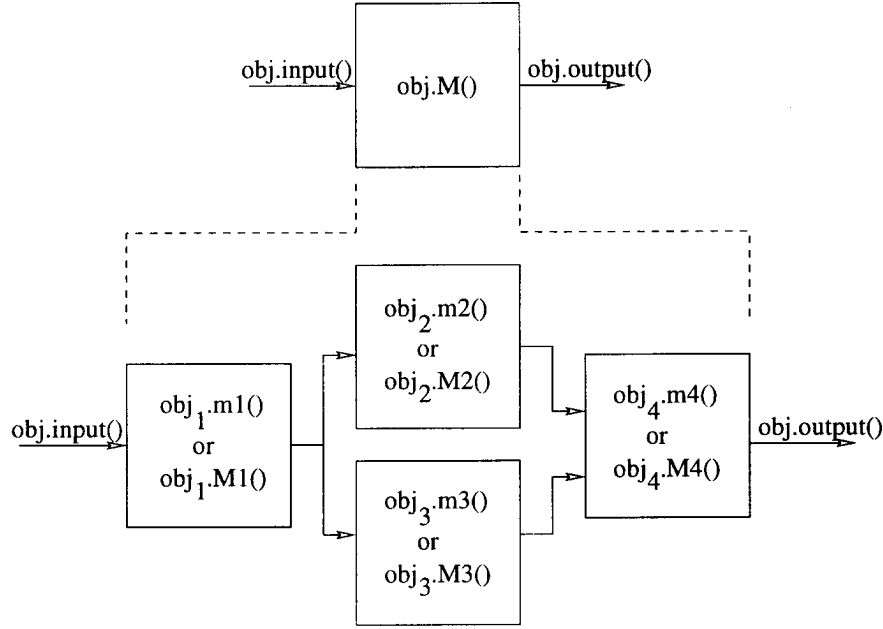


Figure 12: A block multimodel refinement.

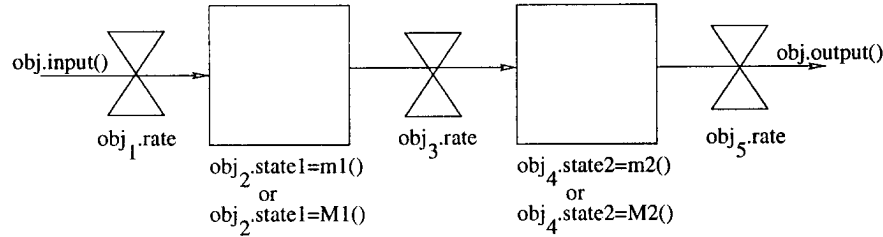


Figure 13: A System Dynamics model.

- *Methods*: $obj.m()$ represents a code method and $obj.M()$ a dynamic model method. m is short for any string beginning with a lower case letter; M is short for any string beginning with an upper case letter. The following special methods are defined for all objects: $obj.input()$ and $obj.output()$ and represent the input and output time trajectories.

The block model contains a collection of blocks coupled together in a network. Each block calculates its input, performs the method representing that block, and produces output. A block is not an object since it represents a method *within an object*. Without any refinement within a block, a function takes no time to simulate. Time is ultimately associated with state change. All obj_i represent sub-objects of obj . Fig. 12 displays two simple block models: the first has one block and the second has 4 coupled blocks. The essence of multimodeling is seen here: a method defined as a dynamic model of the same model sub-type: *block*. Fig. 13 shows a System Dynamics model that is similar to the block model except that instead of methods represented by the network nodes, a node represents an object's state (a variable attribute). The same kind of multimodeling represented in Fig. 12 (refining a block into a dynamic model) can be done for the model in Fig. 13; a block is refined into a System

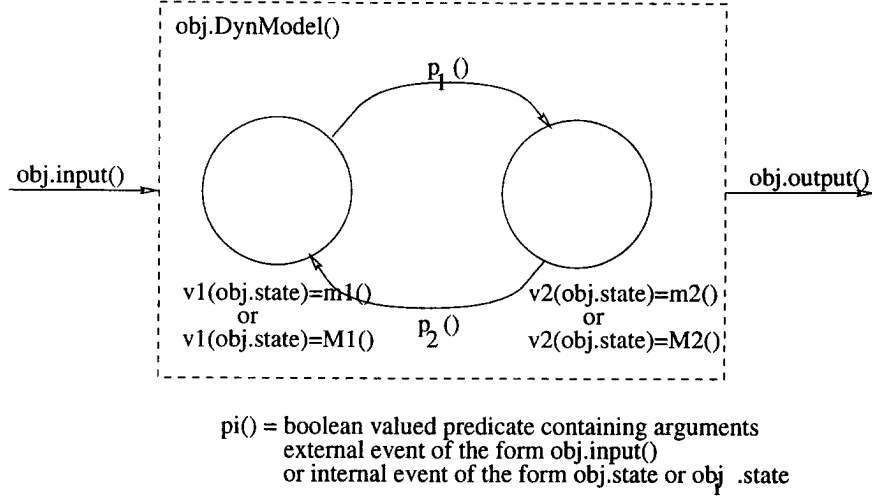


Figure 14: A finite state automaton.

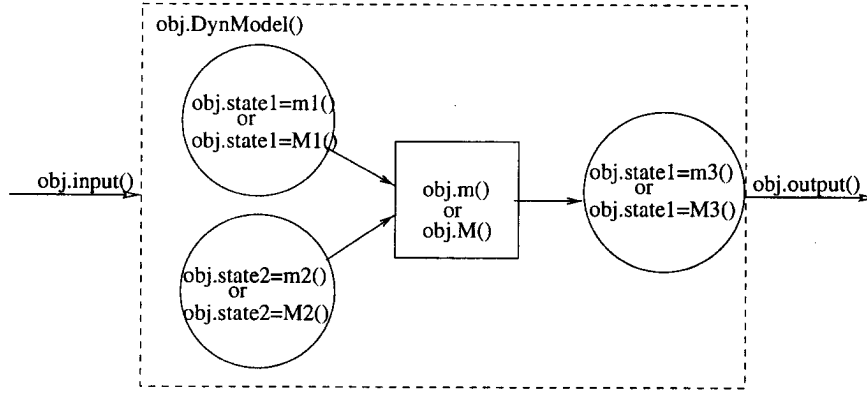


Figure 15: A Petri net.

Dynamics model. This multimodel refinement is particularly useful for our two declarative model types shown in Figs 14 and 15 where *input()* and *output()* are not as obvious unless we capture the model inside of a functional block. Multimodeling is denoted by drawing a dashed functional block around the model, denoting model refinement. The methods *input()* and *output()* are essential to perform *coupling* of models together. An FSA will have an *input()* method and a state variable attribute *obj.state*, but we require coupling information somewhere if we are to decide where the input is coming from and where the output is going to. This coupling information for a method of *obj* is present in some *objⁱ*. For example, if the FSA belongs in a robot *r*, defining the dynamics of state change for *r*, the input must come from a physical object outside of *r* but within a more encapsulating object such as the environment.

The predicates *p₁* and *p₂* in the FSA model in Fig. 14 require further explanation. A predicate is defined as a logical proposition, whose value is boolean, containing internal and external events. External events are of the form *obj.input()* and internal events are of the form *obj.state* or *obj_i.state*. Rules are another convenient dynamic model (of the declarative type) that express changes of state and event. A rule-based model is composed of a collection

of rules within an object *obj*, each rule *i* of the form: IF $p_i(event, state)$ THEN *obj.mi()* or *obj.Mi()*. The phrase $p_i(event, state)$ defines the same logical proposition discussed for the FSA.

6.2.5 Single Object

Single objects are our starting point for defining dynamic models. For example, one may create a simple declarative model in object *r* of class *MobileRobot* by specifying a finite state automaton (FSA) model and linking this to *r.Move()*. We will let the FSA be defined by two states, *S* and *M* for “Stationary” and “Move” respectively. A robot is in state *S* until it receives a cue (an input signal) to move to state *M*. When the robot receives another input signal, it moves back to state *S*. The state variable is located as an attribute *r.position*. Therefore the semantics of the FSA would be such that attribute *r.position* would be modified depending on the state *S* or *M*. For a single object, expressing the dynamical model in terms of object-oriented features turns out to be straightforward. For a number of objects, the procedure is more involved but still reflects a natural method of modeling.

A single object may be an entity such as *r* or a more general space such as *s*. For *s*, we define dynamic models to be those that change attributes of *s*. Models such as partial differential equations (PDEs) and cellular automata (CA) can be defined within *s*. For example, the constraint reaction-diffusion model

$$\frac{\partial \phi}{\partial t} = f(\phi) + \nabla^2 \phi \quad (1)$$

as a method (dynamic model) of *s* describes how ϕ changes over time. For our robot scenario, ϕ is an attribute of the landscape *s* that varies over each cell—such as water depth or vegetation density. Furthermore, *r* can interact with *s* since a component in a method of *r* can depend upon cell attributes, and vice versa.

6.2.6 *k* Non-Interacting Objects

Non-interacting objects do not require any additional dynamic models other than the ones encapsulated within each object. If many O_i reflect robots in motion, there is no need for a dynamic model in an *O* to orchestrate this motion, unless there are constraints placed on the dynamics, in which case a model would be necessary. All O_i , for example, would have to be scheduled by a simulation engine so that they could perform their individual tasks; however, this scheduling is a part of the model execution and does not affect model design.

6.2.7 *k* Interacting Objects

- *k directionally interacting objects*: interacting objects with a directional flow use functional models: data of some type flow through the methods of objects to achieve a change of state. For example, assume that two robot objects r_1 and r_2 are in a quadtree sub-cell called *cell*. r_1 always hands a part to r_2 . This describes the interacting dynamics of an aggregate object *r* containing r_1 and r_2 . A method f_1 is defined in r_1 that captures what is done with the part when r_1 receives it. Similarly, r_2 does f_2 to the incoming part it gets from r_1 . We now make a functional model that is

composed of two coupled functions f_1 and f_2 (f_1 is linked directionally to f_2): $f_1 \rightarrow f_2$. This coupling is performed in accordance with the aggregation rule to define how the coupling is to occur. The aggregate functional model becomes a method of r .

- *k non-directionally interacting objects*: interacting objects with no directionality in the flow are represented using constraint models. If k objects interact in this fashion, a constraint model is encapsulated within the aggregate object containing all k objects. Robot r_1 may interact with r_2 in such a fashion. A simple method of interaction is one using a spring metaphor. We attach a virtual spring between r_1 and r_2 . Then, given an acceleration to r_1 , both robots will move in accordance with Hooke's law in an effort to achieve equilibrium. The equation for Hooke's law is a constraint model located in an aggregate object r that contains both r_1 and r_2 . The equation has terms that are attributes of r_1 and r_2 . The way in which the equation terms are clustered is determined by an aggregation rule for equation construction.

One theme that arises from our methodology of dynamic models is that one locates a dynamic model in the object for which it is appropriate. Often, this object already exists as does r which has a functional model referencing methods stored in r 's sub-objects such as *arm*, *end-effector* and *foundation*. However, in some cases, an aggregate object must be created to locate the model, as with the two robots interacting in spring-like manner: the model of their interaction does not belong to either of them, only to their aggregate object. The latter concept also applies to *populations* of objects where they are present in a scenario. To take an example, let's consider a constraint model in equation form. Where do the equation and the composite equation terms belong? If a term or part of an equation contains attributes only of class C then, it belongs in C . If a dynamic model in r_1 (on the end of a spring) contains only terms of r_1 's position (including input terms dependent on time alone) then the model belongs inside r_1 . If terms including r_2 's position are in a term or equation, then we must move the term or equation up a level to be located in r .

6.2.8 Continuum Models

It may be readily apparent how to take the average scenario object and define it to be object-oriented; however, models that have a *fluid* or non-discrete nature seem to present an incongruity within our defined approach. However, continuum physical phenomena such as fluids, whether gaseous or liquid, need not present a problem for the methodology. If the fluid object is sub-divided into constituent objects in the same way that scalar and vector fields are discretized for numerical reasons (to solve the field equations), then each discrete part is captured as a sub-object of the field. We briefly considered the concept of a river over our robot space s . If the river has rigid boundaries, the sub-objects of s will contain sub-objects of the river object. These object aggregate relations are time-dependent. Objects can be seen to move with the field or stay fixed. If they stay fixed, the dynamics associated with each object follow the methods of finite difference formulations. If they move, they become fluid *particle* objects and are not unlike the robots. If a fluid object is compressible, or the object can add or subtract sub-objects, we add these objects over time and our static models change to accommodate the change in structure. This is a case of having to delete

and add objects dynamically. In the following section, we'll treat an example of dynamic object creation using a biological metaphor.

6.2.9 Morphogenesis

In the previous examples of dynamic systems, a dynamic model was specified within an object for which it was relevant. But, what if the dynamics cause a change in the static model of an object? This is what happens in biological systems and what we call morphogenesis [33]—a temporal change in structure. Lindenmeyer Systems [30] (L-Systems) capture a dynamic way of modeling that falls under the declarative class of dynamic models: rules are specified to model change of object structure. We begin with an object whose state is defined by an attribute serving as an initial condition ω , and continue simulation by growing a static model tree composed of objects. For most engineered devices, such as robots, we do not generally consider dynamic growth to apply; we apply growth to natural objects. However, we stated earlier that our robot was capable of changing shape, so we can carry this metaphor further by stating that the robot link structure in Fig. 2 is the piece of the robot that grows like a tree over time. At the end of the recursive subdivision, an end-effector grows onto the end of each link chain much as a flower grows at the tip of a branch or plant stem. The recursion defined in the productions provides for a tree of objects that grows and is constructed as the methods are applied. After a tree has grown, other state updating methods can be applied for modifying the object states; however, in the majority of cases, growth and decay methods will continue to be applied in parallel with methods that, for instance, change the state of a tree structure (engineered, biological or otherwise) as a result of external forces such as wind or water.

For our models, we will ignore the *joint* object and not directly model the time-dependent change of link width and length. The L-System production model for simulating the link-tree in Fig. 16 is defined below:

$$\begin{aligned}
 p_1 : & \quad \omega : a \\
 p_2 : & \quad a \rightarrow I[A][A]A \\
 p_3 : & \quad A \rightarrow IB \\
 p_4 : & \quad B \rightarrow [C][C] \\
 p_5 : & \quad C \rightarrow ID \\
 p_6 : & \quad D \rightarrow K
 \end{aligned}$$

Production p_1 is the initial condition (or terminating condition). p_2 provides the basic support structure for the plant with an internode, two angle branches and a straight branch. Each branch is constructed with p_3 . p_4 provides a two branch structure at the end of each of the three branches just created, and each of these new branches contains a flower (via productions p_5 and p_6). In the case of L-System model definition, productions p_1 through p_6 represent a single dynamical model defined as a method of the overall aggregate robot object r ; however, it is also possible to separate rules so that a rule defines the dynamics for the object for which it applies. In this case, the rule becomes a method in that object. This

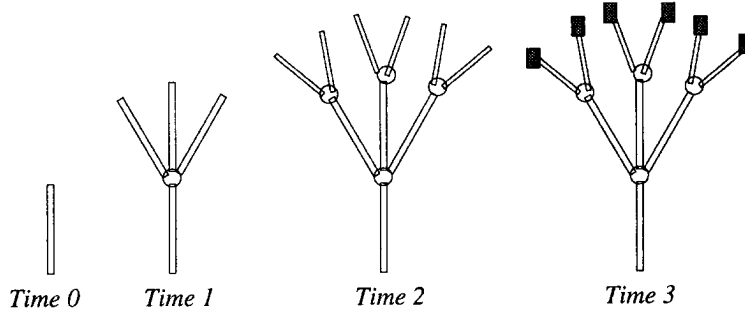


Figure 16: Time snapshots for robot link tree.

“distributed” rule approach is a logical one and since r represents an aggregation of all sub-objects, all six rules are aggregated as a method of r through the composition properties of aggregation. These approaches may suggest different model execution methods for globally situated rules and distributed rules, but we shall not address the model execution issues here since our focus is on model design.

We let time advance be associated with the execution of rules until the appearance of a physical segment (I or K) occurs. The changing state of the robot link tree is shown in Fig. 16. The static model for the growth at time 3 is illustrated in Fig. 17 and the corresponding object and sub-object definitions are illustrated in Fig. 18. Fig. 18 also contains the “internode” I objects that define the individual links.

6.3 Multimodels

We identified dynamic models as being one of three types, and it is possible to create a hierarchy of dynamic models by refining a component of one model as representing another dynamic model. So, for example, one may take a state of r and refine this into a functional model containing two coupled functions. This sort of model decomposition is called heterogeneous model decomposition [11, 32, 13] since more than one model type is used during refinement. Homogeneous refinements are more commonly used, where a model component is refined into similar components but using more detail. In [13], multimodels were visualized outside of an object-oriented framework. In OOPM, a multimodel may be embedded in several physical objects; however, the individual multimodel layers can still be abstracted by refining dynamic model components. Even though we have specified multimodels as applying to dynamic objects, their utility is just as applicable to static models. For example, consider a static model of s : object s contains a quadtree model as an attribute. Each cell of the quadtree contains static models of all robots r inhabiting the cell. Moreover, a link contained with r may be subdefined into yet another model type: a collection of finite volume objects used mostly in finite element analysis.

Multimodels, whether of static or dynamic models, involve changes in scale so that as we refine our models, we change the scale of our scenario and new sub-objects emerge at each abstraction level. For homogeneous refinement, a scale change is accompanied by a regular kind of scale change. Consider the static case first. A landscape s at one level can be sub-divided into cells that are the same shape as s but have metric transformation applied to each of them. This represents a model type we will call *array*. For dynamic models, one

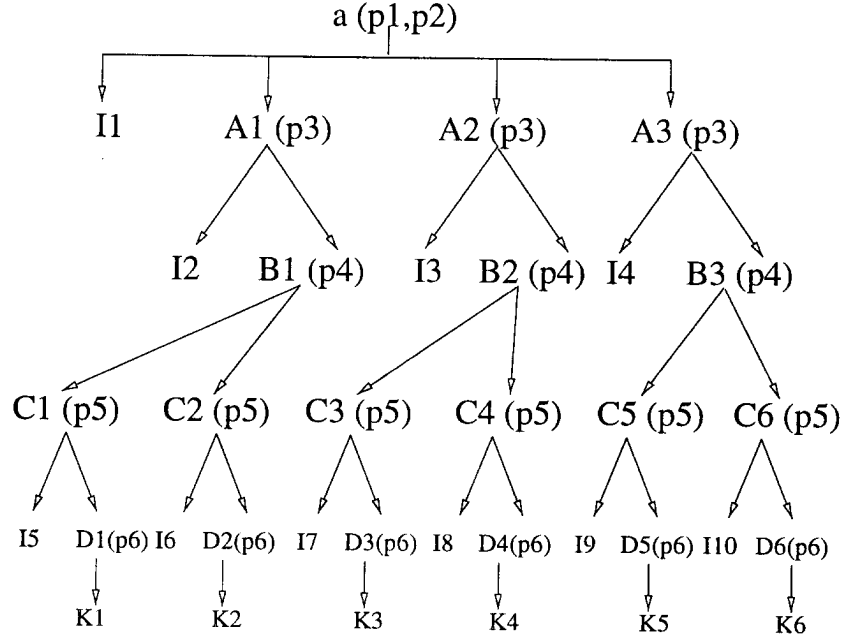


Figure 17: Static model for the robot at time 3.

can look inside r 's computer to find a digital design, composed of interconnected blocks. Each block is subdivided into blocks, yet again, but with blocks of finer granularity. On the other hand, with dynamic model heterogeneous decomposition, we find that we define more coarse grained dynamics for r using an FSA with finer grained dynamics using other model types such as the functional block model. For static models, we may decide to subdivide each cell of s using quadrees. This represents a shift in model type: from an array to a quadtree. Recent work on multimodeling and a new taxonomy for structural and behavioral abstraction is found in [15, 28].

Every dynamic model $obj.M()$ has model components. For multimodeling, the following three model components are important: 1) attribute reference, 2) attribute value, and 3) method. Refinements can be made for each of these model component types.

1. An *attribute reference* is denoted by referring to an attribute $obj.a$. In particular, attributes which hold the set or subset of state space can be used for multimodels by refining the attribute $obj.state$ into a method: $obj.state = m1()$ for a code method refinement or $obj.state = M1()$ for a dynamic model refinement. Examples of this type of refinement are: 1) a Petri net place, 2) a compartment of a compartmental model or 3) a level in a System Dynamics model.
2. An *attribute value* is denoted by referring to the value of an attribute $obj.a$, denoted

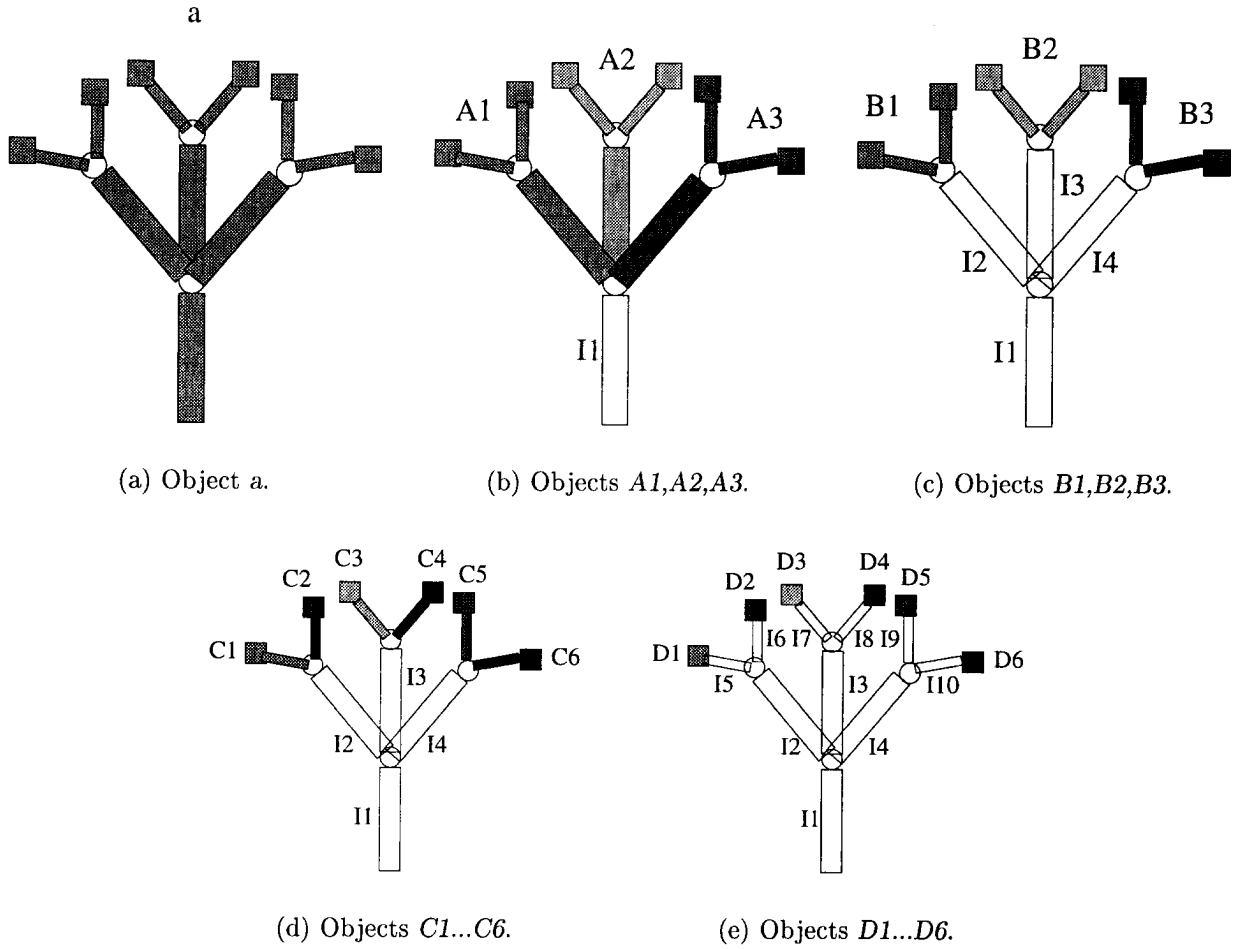


Figure 18: Objects created by time 3.

as $v(obj.a)$. Attribute values appropriate for multimodels reflect a *phase* or aggregate state. A multimodel refinement of an attribute value is performed as either $v(obj.state) = m1()$ or $v(obj.state) = M1()$. Examples of this type of refinement are: 1) an FSA state or a 2) Markov model state.

3. A *method* is denoted by referring to an objects method: either $m1()$ for a code method, or $M1()$ for a dynamic model method. Examples of this are: 1) a function in a block model, 2) a Petri net transition, or 3) a component in a graphical constraint model such as a bond graph. Recall that methods can be constraint relations as well as functions.

6.4 Location of Models

In what object or class does a model belong? This is a key question that arises when building the conceptual model. As we have seen with the different dynamic model types, there are specific approaches for model location depending on the type. In general a good heuristic for

model positioning is to *place a model in an object whose composite objects contain all model components*. For example, when we built a functional model for two robots r_1 and r_2 , we put the coupled two-function model of $f_1 \rightarrow f_2$ inside of an aggregate object r containing r_1 and r_2 . The model could not belong in any single robot even though the model's components are located in individual robots. The same heuristic can be employed for all model types. For an equational model, we might have the following model inside object $o1$:

$$\frac{d}{dt}\rho = k_1\rho + u(t)$$

where ρ is a density attribute within object $o1$. Since this model includes only object $o1$'s attribute and reference to an external input, it belongs in $o1$. Contrast this against:

$$\frac{d}{dt}\rho = k_1\rho + o2.\gamma$$

which contains a constraint relation including a term involving another object's ($o2$) attribute γ . This model must be placed in an aggregate object that contains both $o1$ and $o2$.

6.5 Predator-Prey Model

Consider that some robots act like predators and some act like prey. In this case, an applicable dynamic model to create is along the lines of the Lotka-Volterra model [33]. A conceptual model is shown in Fig. 19. This model suggests that we have a physical scenario composed of an environment (weather), landscape and a population of organisms. There are two types of populations: predator and prey. For the sake of the biological metaphor, we choose *Panther* as the class of predator and *Bird* and *Deer* as sample prey classes. The Lotka-Volterra model is an example of a general population model that can be characterized as a *p-state* ecological model [10]. The designation of *p-state* is positioned in Fig. 19 within the *Population* class where it belongs. For completeness, we have included other types of ecological models [10, 19] and where they fit within the class hierarchy:

- General Population Model (*p-state*): a model that specifies the dynamics of single or inter-species populations.
- Structured Population Model (*i-state distribution*): a population model where other independent variables such as size or age are used to “structure” the population into classes. We placed this within class *BirdPop*. A discrete set of structured classes could also be created under *BirdPop* if desired, such as *Hatchling*, *Juvenile* and *Adult*.
- Individually Model (*i-state configuration*): a set of continuous-time equations, one per individual. If one chooses a discrete event-type approach, using rules for the model type for example, other model types are possible. Wolff [52] refers to a rule-based model as an individual-oriented model (IOM) to differentiate it from the i-state configuration model, termed an individual-based model (IBM).

Let's note the rules for generalization and aggregation:

- Aggregation:

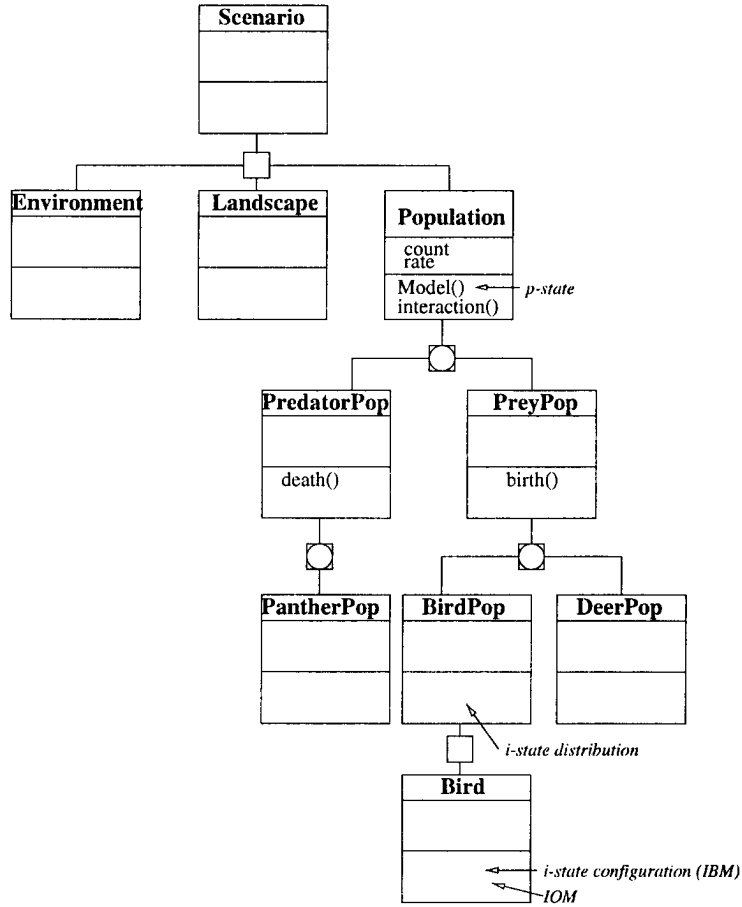


Figure 19: Lotka-Volterra population dynamics.

1. An specific aggregation rule for attribute *count*:

$$Population.count = PredatorPop.count + PreyPop.count$$

A more general aggregation rule for *count*, keeping in mind updates and additions to this conceptual model, is:

$$C.count = \sum_i C_i.count$$

where C matches any class containing *count* and C_i matches the sub-classes of C .

2. An aggregation rule for dynamic model method *Model()* in *Population*:

$$\begin{aligned} PredatorPop.rate &= PreyPop.birth() - PredatorPop.death() \\ PreyPop.rate &= PreyPop.birth() - interaction() \end{aligned}$$

3. An aggregation rule for code method *interaction()* in *Population*:

$$interaction() = PreyPop.count \times PredatorPop.count$$

- Generalization: We let *count* be inherited (passed down) but not any of the methods defined in *Population*, *PredatorPop* or *PreyPop*.

In reviewing our model, we realize several important benefits from the use of OOPM in creating this model. The main benefit is one of knowledge representation that focuses on class creation and the lexical naming of equational terms such as *birth()* and *interaction()*. By making names explicit, we make the model more comprehensible. The benefits of structure passing are inheritance and aggregation. The definition of *Population.Model()* is invariant to additional *PredatorPop* or *PreyPop* sub-classes we may choose to add in the future. For example, we may later add *AlligatorPop* under *PredatorPop*. Since *AlligatorPop* would pass *count* upward via the first aggregation rule, the population model need not be redefined.

7 Programming and Implementation

We are building a system called MOOSE: multimodeling object-oriented simulation environment. MOOSE will capture the essence of OOPM by leading the user through the phases of model development shown in Fig. 3. The user constructs a conceptual model using a Tk/Tcl graphical user interface. The user adds attributes and methods. For those attributes and methods that are defined as *models*, there is a model window that permits visual editing. As the model is executed, the simulation output is shown on a scenario window. Our progress to date has illustrated the use of simulation to the planning process [27, 26]; however, we are still building the graphical user interface utilities for the model window. A sample scenario window for an air force mission application was constructed along with a simulation built on top of SimPack [12, 13, 9].

We draw a dividing line between the actual implementation and the *logical design* which is used as a basis for code implementation. Our focus in this article has been on this logical design. Some implementation choices, for example, have dictated that for a particular object-oriented language, creating formal objects for every individual in a population may not be computationally feasible, even though our design is drawn so that these objects exist. This is not a problem and reflects that the design stands by itself and is used as an intermediate vehicle from concept to code. Different computer languages have their own unique features, and the basic object-oriented physical design should not be bound by what is offered or not offered by these languages. C++ is an example of a language that offers inheritance but not composition so while inheritance is supported in the form of derived class structure, composition is handled on a case-by-case basis where the programmer stores the structure in the object(s) affording computational efficiency.

8 Conclusions

To build simple systems, we may sometimes get away without using a model design. In such a case, we may sketch a few formulae and proceed directly to the coding phase. However, with the increasing speed of personal computers, we are in a period of increased development for model design that might best be captured by the word “integration.” As scientists and engineers, we have our own individual static and dynamic models for our part of the world. But this is not enough when we want to integrate models together. Suddenly, we find

ourselves overwhelmed with the sheer size of model types, and frequently some may not have model types but have only coded their simulations. To get a handle on this situation, we need a blueprint as if we were going to perform this integration as a metaphor to building a house. Without a blueprint, the electrician and carpenter are at odds as to how to interact. They each construct their own complex parts and one only hopes that the resulting glued-together construction will function as a whole. The blueprint helps them to work together. Our methodology for object-oriented physical design is like the blueprint, permitting models of different types to fit together so that more complex and larger systems can be studied. These larger systems require an interdisciplinary approach to model design and so we must agree on a basic language for blueprints.

Our immediate goals are to apply this general methodology to various technical areas including the simulation of multi-phase particle flows in the University of Florida Engineering Research Center for Particle Science and Technology and an integrated modeling environment for studying the effects of changes in hydrology to the Everglades ecosystem managed by the South Florida Water Management District. For decision making in the military, many levels of command and control exist, and the methodology provides a consistent approach in using models for planning and mission analysis both “before action” and during “after action review.” All of these systems have a characteristic in common even though they may appear at first quite different: they involve the modeling of highly complex, multi-level environments, often with individual code and models developed by different people from different disciplines. MOOSE development is underway and C++ code and GUI interfaces are being constructed to make it possible for analysts to use our system. A longer range goal is to allow our models to be distributed over the Internet (or over processors for a parallel machine). The object oriented concepts of re-use and encapsulation will help greatly in this endeavor. Also, we are trying to create a bridge between the use of modeling in simulation and general purpose programming. As various authors have noted [8, 14], if one liberally applies the concept of metaphor to software engineering, the differences between software and systems engineering begin to dwindle to the point where software engineering can be considered a *modeling process*.

Acknowledgments

I would like to acknowledge the graduate students of the MOOSE team for their individual efforts in making MOOSE a reality: Robert Cubert, Tolga Goktekin, Gyooseok Kim, Jin Joo Lee, Kangsun Lee, and Brian Thorndyke. In particular, Brian and Robert critiqued an early version of the manuscript and Tolga designed the geometry in Figs. 10(a) and 10(b). We would like to thank the following funding sources that have contributed towards our study of modeling and implementation of a multimodeling simulation environment for analysis and planning: (1) Rome Laboratory, Griffiss Air Force Base, New York under contract F30602-95-C-0267 and grant F30602-95-1-0031; (2) Department of the Interior under grant 14-45-0009-1544-154 and the (3) National Science Foundation Engineering Research Center (ERC) in Particle Science and Technology at the University of Florida (with Industrial Partners of the ERC) under grant EEC-94-02989.

References

- [1] Osman Balci and Richard E. Nance. Simulation Model Development Environments: A Research Prototype. *Journal of the Operational Research Society*, 38(8):753 – 763, 1987.
- [2] Jerry Banks and John S. Carson. *Discrete Event System Simulation*. Prentice Hall, 1984.
- [3] G. M. Birtwistle. *Discrete Event Modelling on SIMULA*. Macmillan, 1979.
- [4] Grady Booch. On the Concepts of Object-Oriented Design. In Peter A. Ng and Raymond T. Yeh, editors, *Modern Software Engineering*, chapter 6, pages 165 – 204. Van Nostrand Reinhold, 1990.
- [5] Grady Booch. *Object Oriented Design*. Benjamin Cummings, 1991.
- [6] Alan H. Borning. THINGLAB – A Constraint-Oriented Simulation Laboratory. Technical report, Xerox PARC, 1979.
- [7] Peter C. Breedveld. A Systematic Method to Derive Bond Graph Models. In *Second European Simulation Congress*, Antwerp, Belgium, 1986.
- [8] Timothy Budd. *An Introduction to Object Oriented Programming*. Addison-Wesley, 1991.
- [9] Robert M. Cubert and Paul A. Fishwick. OOSIM User’s Manual. Technical report, University of Florida, Department of Computer and Information Science and Engineering, 1996.
- [10] Donald L. DeAngelis and K. A. Rose. Which Individual-Based Approach is Most Appropriate For a Given Problem? In Donald L. DeAngelis and Louis J. Gross, editors, *Individual-Based Models and Approaches in Ecology*, pages 67–87. Chapman and Hall, New York, 1992.
- [11] Paul A. Fishwick. Heterogeneous Decomposition and Coupling for Combined Modeling. In *1991 Winter Simulation Conference*, pages 1199 – 1208, Phoenix, AZ, December 1991.
- [12] Paul A. Fishwick. Simpack: Getting Started with Simulation Programming in C and C++. In *1992 Winter Simulation Conference*, Arlington, VA, December 1992.
- [13] Paul A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds*. Prentice Hall, 1995.
- [14] Paul A. Fishwick. Toward a Convergence of Systems and Software Engineering. *IEEE Transactions on Systems, Man and Cybernetics*, May 1996. Submitted for review.
- [15] Paul A. Fishwick and Kangsun Lee. Two Methods for Exploiting Abstraction in Systems. *AI, Simulation and Planning in High Autonomous Systems*, pages 257–264, 1996.

- [16] Paul A. Fishwick and Bernard P. Zeigler. A Multimodel Methodology for Qualitative Model Engineering. *ACM Transactions on Modeling and Computer Simulation*, 2(1):52–81, 1992.
- [17] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990. Second Edition.
- [18] Ian Graham. *Object Oriented Methods*. Addison Wesley, 1991.
- [19] Thomas G. Hallam, Ray R. Lassiter, Jia Li, and William KcKinney. Modeling Populations with Continuous Structured Models. In Donald L. DeAngelis and Louis J. Gross, editors, *Individual-Based Models and Approaches in Ecology*, pages 312–337. Chapman and Hall, New York, 1992.
- [20] Donald Hearn and M. Pauline Baker. *Computer Graphics*. Prentice Hall, 1994.
- [21] Nevin Heintze, Joxan Jaffar, Spiro Michaylov, Peter Stuckey, and Roland Yap. *The CLP(R) Programmer's Manual: Version 1.1*, November 1991.
- [22] David R. C. Hill. *Object-Oriented Analysis and Simulation*. Addison-Wesley, 1996.
- [23] John A. Jacquez. *Compartmental Analysis in Biology and Medicine*. University of Michigan Press, 1985. Second edition.
- [24] Robert Kowalski. *Logic for Problem Solving*. Elsevier North Holland, 1979.
- [25] Averill M. Law and David W. Kelton. *Simulation Modeling & Analysis*. McGraw-Hill, 1991. Second edition.
- [26] Jin Joo Lee. *A Simulation-Based Approach for Decision Making and Route Planning*. PhD thesis, June 1996.
- [27] Jin Joo Lee and Paul A. Fishwick. Real-Time Simulation-Based Planning for Computer Generated Force Simulation. *Simulation*, 63(5):299–315, November 1994.
- [28] Kangsun Lee and Paul A. Fishwick. A Methodology for Dynamic Model Abstraction. *SCS Transactions on Simulation*, 1996. Submitted August 1996.
- [29] William Leler. *Constraint Programming Languages: Their Specification and Generation*. Addison Wesley, 1988.
- [30] Aristid Lindenmeyer. Mathematical Models for Cellular Interaction in Development. *Journal of Theoretical Biology*, 18:280–315, 1968.
- [31] Greg Michaelson. *An Introduction to Functional Programming through Lambda Calculus*. Addison Wesley, 1989.
- [32] Victor T. Miller and Paul A. Fishwick. Heterogeneous Hierarchical Models. In *Artificial Intelligence X: Knowledge Based Systems*, Orlando, FL, April 1992. SPIE.

- [33] J. D. Murray. *Mathematical Biology*. Springer Verlag, 1990.
- [34] Richard E. Nance. Simulation Programming Languages: An Abridged History. In *1995 Winter Simulation Conference*, pages 1307 – 1313, Washington, DC, December 1995.
- [35] Donald A. Norman. *The Design of Everyday Things*. Currency Doubleday, New York, 1988.
- [36] Louis Padulo and Michael A. Arbib. *Systems Theory: A Unified State Space Approach to Continuous and Discrete Systems*. W. B. Saunders, Philadelphia, PA, 1974.
- [37] L. C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1991.
- [38] James L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.
- [39] Herbert Praehofer. Systems Theoretic Formalisms for Combined Discrete-Continuous System Simulation. *International Journal of General Systems*, 19(3):219–240, 1991.
- [40] G. Revesz. *Lambda Calculus Combinators and Functional Programming*. Cambridge University Press, 1988.
- [41] Chell A. Roberts, Terrence Beaumariage, Charles Herring, and Jeffrey Wallace. *Object Oriented Simulation*. Society for Computer Simulation International, 1995.
- [42] Nancy Roberts, David Andersen, Ralph Deal, Michael Garet, and William Shaffer. *Introduction to Computer Simulation: A Systems Dynamics Approach*. Addison-Wesley, 1983.
- [43] Ronald C. Rosenberg and Dean C. Karnopp. *Introduction to Physical System Dynamics*. McGraw-Hill, 1983.
- [44] Jeff Rothenberg. Object-Oriented Simulation: Where do we go from here? Technical report, RAND Corporation, October 1989.
- [45] James Rumbaugh, Michael Blaha, William Premerlani, Eddy Frederick, and William Lorenson. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
- [46] Hanan Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, 1990.
- [47] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
- [48] Lee W. Schruben. Simulation Modeling with Event Graphs. *Communications of the ACM*, 26(11), 1983.
- [49] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, 2 edition, 1991.
- [50] Jean Thoma. *Bond Graphs: Introduction and Application*. Pergamon Press, 1975.

- [51] Patrick Henry Winston and Berthold Klaus Paul Horn. *LISP*. Addison Wesley, second edition, 1984.
- [52] Wilfried F. Wolff. An Individual-Oriented Model of a Wading Bird Nesting Colony. *Ecological Modelling*, 72:75–114, 1994.
- [53] William A. Woods. What’s in a Link: Foundations for Semantic Networks. In Daniel Bobrow and Allan Collins, editors, *Representation and Understanding*. Academic Press, 1975.
- [54] Bernard P. Zeigler. Towards a Formal Theory of Modelling and Simulation: Structure Preserving Morphisms. *Journal of the Association for Computing Machinery*, 19(4):742 – 764, 1972.
- [55] Bernard P. Zeigler. *Theory of Modelling and Simulation*. John Wiley and Sons, 1976.
- [56] Bernard P. Zeigler. DEVS Representation of Dynamical Systems: Event-Based Intelligent Control. *Proceedings of the IEEE*, 77(1):72 – 80, January 1989.
- [57] Bernard P. Zeigler. *Object Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press, 1990.

Biography

Paul A. Fishwick is an Associate Professor in the Department of Computer and Information Science and Engineering at the University of Florida. He received the PhD in Computer and Information Science from the University of Pennsylvania in 1986. He also has six years of industrial/government production and research experience working at Newport News Shipbuilding and Dry Dock Co. (doing CAD/CAM parts definition research) and at NASA Langley Research Center (studying engineering data base models for structural engineering). His research interests are in computer simulation modeling and analysis methods for complex systems. He is a senior member of the IEEE and the Society for Computer Simulation. He is also a member of the IEEE Society for Systems, Man and Cybernetics, ACM and AAAI. Dr. Fishwick founded the `comp.simulation` Internet news group (Simulation Digest) in 1987. He has chaired workshops and conferences in the area of computer simulation, and will serve as General Chair of the 2000 Winter Simulation Conference. He was chairman of the IEEE Computer Society technical committee on simulation (TCSIM) for two years (1988-1990) and he is on the editorial boards of several journals including the *ACM Transactions on Modeling and Computer Simulation*, *IEEE Transactions on Systems, Man and Cybernetics*, *The Transactions of the Society for Computer Simulation*, *International Journal of Computer Simulation*, and the *Journal of Systems Engineering*. Dr. Fishwick’s WWW home page is <http://www.cise.ufl.edu/~fishwick> and his E-mail address is fishwick@cise.ufl.edu.

Asynchronous Parallel Discrete Event Simulation

Yi-Bing Lin and Paul A. Fishwick, *Senior Member, IEEE*

Abstract—Complex models may have model components distributed over a network and generally require significant execution times. The field of parallel and distributed simulation has grown over the past fifteen years to accommodate the need of simulation the complex models using a distributed versus sequential method. In particular, asynchronous parallel discrete event simulation (PDES) has been widely studied, and yet we envision greater acceptance of this methodology as more readers are exposed to PDES introductions that carefully integrate real-world applications. With this in mind, we present two key methodologies (*conservative* and *optimistic*) which have been adopted as solutions to PDES systems. We discuss PDES terminology and methodology under the umbrella of the personal communications services application.

I. INTRODUCTION

OUR purpose is to introduce the basic technical concepts of distributed simulation of event-based models (so called *discrete event models*), and to tie these generic concepts to a specific application: personal communications services (PCS). Several introductory articles have been presented in the literature such as Fujimoto [1], Nicol *et al.* [2] and Richter *et al.* [3]. These papers have helped to disseminate the *asynchronous parallel discrete event simulation* (PDES) methodology for a wide readership. Our approach is similar but stresses a single real world application for discussing the methodology of PDES. By defining the methodology and all PDES terminology within the context of the PCS application, this paper serves both as a tutorial to PDES and as an introduction to PCS simulation modeling. PCS is a rich enough application to illustrate most basic PDES concepts.

The processing elements in PDES can either be of a *parallel* or *distributed* nature. An MIMD machine with multiple asynchronous elements performing message passing is an example of a parallel machine. Distributed elements normally refers to local or wide area networks composed of inter-connected set of heterogeneous workstations and computers. PDES is used for one of two reasons: 1) one wants to execute a model faster than is possible in a sequential machine, or 2) one must model in a distributed fashion because of a constraint that a process

(i.e., computation) must be distributed rather than localized to a single processor. One author (Lin) has demonstrated various speedups possible on a distributed memory architecture for the PCS application [4], [5]. There is no question that PDES speeds up otherwise serial computations during a simulation. The second reason for PDES (distributed model constraint) is based on a situation where models for system components are stored in physically different locations. The other author (Fishwick) is building a prototype distributed simulation of a process plant where each plant component is ultimately co-located with the manufacturer of that component.

The paper proceeds as follows. First, in Section II, we define our terms within the PDES area and demonstrate the generic approach to distributed simulation. In Section III, we introduce the PCS application and demonstrate the need for synchronization of incoming messages to a given process. There are two key approaches to synchronization. Method 1, defined in Section IV, is termed the *conservative method* since it ensures that the causal relation among time consecutive events will be maintained at all times during the simulation. Method 2 is defined in Section V, and identifies the *optimistic method*. In this approach, the causal relation can be broken with subsequent fixing of state variables. We close in Section VI with directions for the future of PDES.

Throughout this paper, we use three font styles to represent different concepts. The typewriter type style represents attributes or methods (e.g., `SendMessage()`) of objects. The *italic* type style represents variables such as *LP* or *p*. The serif type style represents event types such as *CallArrival*.

II. PARALLEL DISCRETE EVENT SIMULATION

A. Basic Terminology

We begin by defining terms which are commonly found in the simulation and PDES fields. These terms will be revisited in Section 3 when we assign the terms to the PCS application. The study of any physical system to be simulated begins with the creation of a *model*. Such a model can be in one of several types [6]: 1) conceptual, 2) declarative, 3) functional, 4) constraint, 5) spatial or 6) multimodel. One begins with a conceptual model which describes qualitative terms and class hierarchies for the system. In many ways, the conceptual model "organizes" the definition of attributes, methods and general characteristics of each system component without going so far as to ascribe dynamics to components. The next four model types reflect an orientation to system construction; a system may be constructed as a Petri net [7], queuing model [8] or as a cellular automaton [9] for instance. The last model

Manuscript received January 15, 1995; revised July 25, 1995. This work was supported by the Rome Laboratory, Griffiss AFB, NY, under Contract F30602-95-C-0267, and Grant F30602-95-1-0031, the Department of the Interior under Grant 14-45-0009-1544-154, and the National Science Foundation Engineering Research Center (ERC) in Particle Science and Technology, University of Florida (with Industrial Partners of the ERC), under Grant EEC-94-02989.

Y.-B. Lin is with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: liny@csie.nctu.edu.tw).

P. A. Fishwick is with the Department of Computer and Information Sciences, University of Florida, Gainesville, FL 32611 USA (e-mail: fishwick@cis.ufl.edu).

Publisher Item Identifier S 1083-4427(96)03838-6.

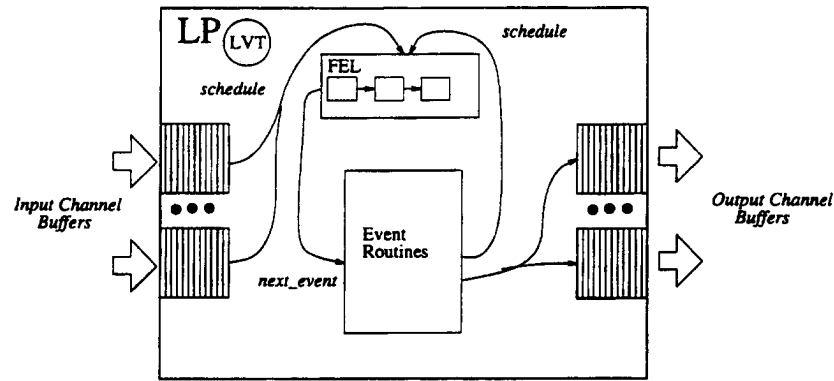


Fig. 1. Anatomy of a logical process (LP).

type (multimodel) permits the integration of basic model types to create a model composed of component models [10], [11] where each component model represents a level of abstraction for the system.

The PCS area, to be discussed in Section 3, uses a *spatial model* in that the system is viewed as a hexagonal discretization of a large two-dimensional space representing an area where cellular communications are to be implemented. Spatial models can be executed in several ways including *time slicing*, *event scheduling* and *parallel and distributed*. Our approach will be to use a parallel and distributed approach to *model execution*, while using the concept of event scheduling within each process. Speaking of *process*, we must define this term appropriately. Model components for a PCS implementation will be a collection of hexagonal cells. Other model types, such as a queuing model, are composed of other components (facilities). A *logical process* (LP) is defined as a set containing basic model components, so a PCS logical process will be a set of hexagons, or just one hexagon. A *physical process* or *processor* is a set of *logical processes* mapped in a way that conforms to the architecture of the parallel/distributed system.

An LP contains several objects:

- *Local Virtual Time* (LVT): time associated with the LP. The LP does not know another LP's time unless communicated via a message.
- *Future Event List* (FEL): event list used when there are internal events posted within the LP itself.
- *Event*: an item within the FEL.
- *Message*: an item sent from one LP to another.

The FEL is composed of events, where an event combines the following objects: 1) time stamp, 2) token, 3) event type. The time stamp reflects when the event is to occur. An event's occurrence correlates with the execution of an *event routine* for that LP. The token is associated with whatever is flowing through the network of LP's. For the PCS application, *portables* (i.e., mobile phones) flow through the system. An event type specifies what will happen to the token (arrival, boundary crossing, departure, incoming call). An LP has input channels and output channels where each channel has a first-in/first-out (FIFO) buffer associated with it. A message is equivalent to an event that must be moved from one LP

to another. Messages which simply enter an FEL and are processed are generally called events. When an event must be issued to another LP, it becomes a message. The relationship among the above terms is shown in Fig. 1.

Messages arrive in one of several input channel buffers and are routed directly to the LP's FEL. Note that simple LP's may involve a calculation such as 1) taking the timestamp from an incoming message, 2) adding a value to this timestamp, and 3) sending the new message to the output buffers. Such an LP would not have any need of an FEL and would be a "pure" distributed simulation. This kind of technique, however, is wasteful of the computing elements since there will be a large price to pay in communications overhead among inter-LP communication. A simple addition is not sufficient to warrant a distributed approach. On the other hand, if the processing element can be *made to do work* then the communications overhead becomes less critical. The kind of work ideally suited in simulation is a sequential simulation within the LP, composed of the usual FEL and event routines. Thus, the distributed simulation is hybrid in form with sequential simulation coinciding—and synchronized with—distributed simulation. The LVT of this more substantial LP is updated by removing the highest priority event (lowest timestamp) from the FEL and executing the associated event routine. Some (or all) of these event routines will contain scheduling commands to place events with new times back into the FEL. Some event routines will involve messages to be issued through the output buffer(s) to a target LP.

B. Object Oriented Implementation

A PDES consists of several PDES objects or LP's. These LP's execute asynchronously with coordination to complete a simulation run. To implement the objects in an LP (as described in Section II-A), the attributes and methods of the LP are classified into four categories (see Table I):

- A *clock* mechanism indicates the progress of the LP. An attribute LVT represents the timestamp of the event that just occurred in the LP. The `LVTUpdate()` method updates LVT to advance the "clock" of the LP.
- A *FEL* mechanism processes the events occurring in the LP. The FEL is basically a *priority queue* with one

TABLE I
ATTRIBUTES AND METHODS OF AN LP

Mechanism	Attributes	Methods
Clock	LVT	LVTUpdate()
FEL	eventList	Enqueue() Dequeue() Cancel()
Synchronization		ReceiveMessage() SendMessage() ExecuteMessage()
Application	to be elaborated	to be elaborated

attribute and three methods. An attribute eventList maintains the events to occur in the future. The Enqueue() method inserts a time-tagged event into eventList so that eventList maintains its ordered sequence. The Dequeue() method deletes the event with the minimum timestamp in eventList. The Cancel() method deletes the event with a specified timestamp in eventList.

- A *synchronization* mechanism interacts with other LP's to coordinate the execution of PDES. The ReceiveMessage() method receives messages from other LP's (these messages will be inserted into the FEL for processing). The method ExecuteMessage() executes events in the FEL. The SendMessage() method sends output message (generated by the execution of events) to their destination LP's. It is probably more appropriate to consider ExecuteMessage() as a method of the FEL. However, this method is affected by the PDES synchronization mechanisms to be described later. Thus the method is classified as part of the synchronization mechanism.
- An *application* mechanism represents a sub-model for a specific simulation application to be simulated by the LP (to be elaborated).

C. PDES Implementation Platforms

PDES systems have been implemented in different parallel architectures such as BBN Butterfly [12]–[14], Sequent [15]–[17], JPL Mark III [18], Simulated Stanford Dash Multiprocessor [19], Transputers [20], [21], CM-1/CM-5 [22], KSR [23], and iPSC/860 [24]. PDES has also been implemented in workstations connected by a local area network [4] which is widely available in both the industrial and the academic environments.

III. PERSONAL COMMUNICATION SERVICES

We use *personal communication service* (PCS) network simulation to illustrate PDES functionality. A PCS network [25], [26] provides low-power and high-quality wireless access for PCS subscribers or *portables*. The service area of a PCS network is populated with a number of radio ports. Every radio port covers a sub-area or *cell*. The port is allocated a number of channels (time slots, frequencies, spreading codes or a combination of these). A portable occupies a channel for an incoming/outgoing call. If all channels are busy in the radio port, the call is blocked. In PCS network planning,

PCS network modeling (usually conducted by simulation experiments) is required to investigate the usage of radio resources. Since PCS network simulation is time-consuming, PDES effectively speeds up the process of PCS network simulation. Specifically,

- The size of the PCS network under study is usually large (e.g., thousands of cells). A typical sequential PCS simulation run takes over 20 hours, while the corresponding PCS PDES takes less than 3 hours using 8 processors [4].
- Another popular parallel approach, the *parallel independent replicated simulation* [27]–[29] (running multiple simulation replications concurrently) does not work for PCS simulation. In most cases, the PCS designer is interested only in the behavior of the PCS network at the engineered workload (e.g., the workload at which the blocking probability is 1%). To calibrate the simulation at the engineered workload, the setup of input parameters for the next simulation run is dependent on the previous run.

Now we describe the PCS model and its mapping to the corresponding PDES. For demonstration purposes, we describe a simplified PCS model without considering the details of the radio signal propagation issues (such as Rayleigh fading, co-channel interference, and so on). We assume that there are S cells in the PCS network, and on the average, there are n portables in a cell. Every port is allocated some number of channels. A portable resides at a cell for a period of time which is a random variable with some distribution (e.g., exponential [30]–[32]). Then the portable moves to a neighbor cell based on some routing function (e.g., equal routing probabilities for all neighbors). The call arrivals to a portable is a random process (e.g., Poisson), and is independent of the portable movement. A call is connected if a channel is available. Otherwise, the call is *blocked*. When a portable moves from one cell to another while a call is in progress, the call requires a new channel (in the new cell) to continue. This procedure of changing channels is called *handoff* or *automatic link transfer* (ALT). Several handoff schemes have been proposed in the literature [33]–[35]. In this paper, we consider the simplest scheme called *nonprioritized scheme*. In this scheme, if no channel is available in the new cell, then the call will be dropped or forced terminated immediately.

The PCS example is probably more realistic to the reader if we add some geometry to these moving vehicles (portables). Unfortunately, whether a vehicle moves from one cell to another cannot be simply determined by the physical movement of the vehicle. We also need to consider the radio propagation. It is possible that the connection to a vehicle changes from one port to another even if the vehicle is stationary—the change of radio signal strength may result in re-connecting the vehicle to a different port. According to the PCS network measurement methods, we determine that the movement (in the sense of port connection) of a vehicle is best characterized by the residence time¹ distribution and the destination cell routing probability. The reader may image that this movement model is equivalent to a simple path approach where a vehicle moves straight with

¹ Residence time refers to the time that a portable *resides* within a cell.

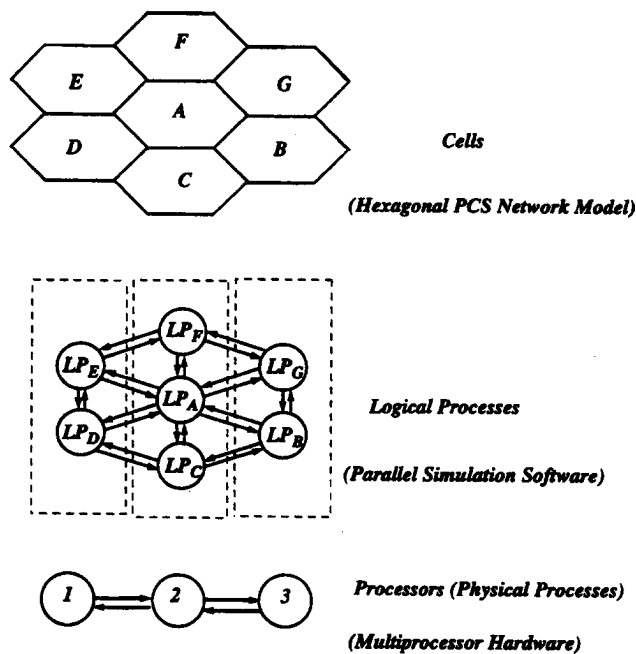


Fig. 2. Cells, logical processes, and processors. A PCS cell is represented by a logical process (LP) in PDES. More than one LP may be mapped to a processor for execution.

an angle. The angle determines the destination cell and the residence time is the product of a constant speed and the diameter of the cell².

To map the PCS model into PDES, the cells in the PCS network are represented by cell objects derived from the PDES objects (i.e., LP's). These LP's are then mapped to processors for execution (see Fig. 2). A cell LP has the following attributes and methods (i.e., the application mechanism of a general LP): A constant attribute `channelNo` represents the total number of channels in a radio port. An attribute `idleChannelNo` represents the number of idle radio channels. A `portableList` collects the information of all portables reside in the cell. There are five methods in the cell object: `CallArrival()`, `CallCompletion()`, `PortableMoveIn()`, `PortableMoveOut()`, and `Handoff()`. These methods will be elaborated later.

The portables in the PCS network are represented by the *portable objects*. A portable object consists of four attributes:

- The `busy` attribute indicates the status of the portable. If `busy=YES` then the portable is in a conversation.
- The `callArrivalTime` attribute represents the next call arrival time.
- The `callCompletionTime` attribute represents the completion time of the current phone call when `busy=YES`. If `busy=NO`, the `callCompletionTime` attribute is meaningless.
- The `portableMoveOutTime` attribute represents the time when the portable moves out of the current cell.

There are two categories of events in a PDES. An *internal* event is scheduled and executed at the same LP (the event

represents the interaction between a cell and a portable within the cell in our PCS example), and an *external* event is scheduled by one LP and is executed by another LP. Thus, after its creation, an internal event is inserted in the FEL by using the `Enqueue()` method, and an external event is considered as a message, and is sent to the destination LP by using the `SendMessage()` method. In the PCS PDES, there are three internal event types and one external event type. The internal event types are described below.

- **CallArrival:** Either the port (the cell) or the portable initiates a call setup. A radio link is required to connect the port and the portable. If no radio link is available or the portable is already busy with another conversation, the call is dropped.
- **CallCompletion:** A phone call completes, and the radio link between the port and the portable is disconnected.
- **PortableMoveOut:** The portable moves out of a cell. If the portable is in a conversation, the radio link between the portable and the port is disconnected.

We treat the `CallArrival` event type as an internally generated event based on a probability distribution. This is just an abstraction of the actual situation where arrivals are sent from outside the LP to one of the LP's input channels. Therefore, a more detailed simulation would involve "electromagnetic messages" reflecting the true nature of incoming calls. The use of a probability function is an abstraction for this underlying process.

The external event type is described below.

- **PortableMoveIn:** A portable moves in a new cell. If the portable is in a conversation, then a new radio link between the cell (port) and the portable is required. If no radio channel is available, the call is forced terminated.

In PDES, the execution of a `PortableMoveOut` event at a logical process `LP_A` always results in the scheduling of a `PortableMoveIn` event for the destination logical process `LP_B`. This event type is *external* (to `LP_B`), and the scheduling of the event requires communication between `LP_A` and `LP_B`.

An event/message m is of the format

$$m = (\text{timeStamp}, \bar{p}, \text{eventType})$$

where *eventType* represents the type of the event, *timeStamp* represents the (simulated) time when the event occurs, and $\bar{p}$ is the pointer which points to the corresponding portable p . The execution of the event message m at a cell object LP is described as follows. The `LP.ExecuteMessage()` method invokes different methods according to the event type of m (the Pascal-like "case" statement is used in the definition shown at the bottom of the next page). The methods invoked in `ExecuteMessage()` are described below. When the event type of m is `CallArrival`, the following action is taken.

```
CallArrival(p) {
  if p.busy=YES then
    /* A call is already in progress when the new */
    /* call arrives at LVT. In other words, a busy line */
    /* occurs and the new call arrival is ignored. */
    update the busy line statistic;
  else /* i.e., p.busy=NO. */
```

²But note that our movement model is practical—it is used to approximate real radio systems, unlike the simple path approach.

```

if idleChannelNo = 0 then
  /* The call arrival is blocked. */
  update the blocking statistic;
else /* I.e., idleChannelNo > 0. */
  idleChannelNo ← idleChannelNo - 1;
  p.busy ← YES;
  generate the call holding time  $t$ , and
    p.callCompletionTime ← LVT +  $t$ ;
end if
end if
generate the next inter-call arrival time  $t'$  and compute
the next call arrival time as
  p.callArrivalTime ← LVT +  $t'$ .
invoke ScheduleNewEvent( $p$ );
/* Schedule a new event (to be described). */
}

```

Note that the busy line and call blocking statistics are output measures of the PCS simulation (not shown in our PDES example)³.

When the event type of m is CallCompletion, the following action is taken.

```

CallCompletion( $p$ ) {
  /* Release occupied channel at call completion. */
  idleChannelNo ← idleChannelNo + 1;
  p.busy = NO;
  invoke ScheduleNewEvent( $p$ );
  /* Schedule a new event (to be described). */
}

```

When the event type of m is PortableMoveIn, the following action is taken.

```

PortableMoveIn( $p$ ) {
  if p.busy = YES then /* A handoff occurs. */
    invoke Handoff( $p$ ); /* To be described. */
  end if
  generate the portable residence time  $t$  and compute the
  next move time p.portableMoveOutTime ← LVT +  $t$ ;
  invoke ScheduleNewEvent( $p$ );
  /* Schedule a new event (to be described). */
}

```

The method Handoff() is described below.

```

Handoff( $p$ ) {
  if idleChannelNo = 0 then
    /* No channel is available to connect the */
    /* handoff call i.e., the handoff fails. */
  end if
}

```

³ Call blocking is a major performance measure of a PCS network. A PCS network is usually engineered at 1% or 2% blocking probabilities.

```

    update the forced termination statistic
    (not shown in our PDES example);
    p.busy = NO;
  else /* The handoff succeeds. */
    idleChannelNo ← idleChannelNo - 1;
  end if
}

```

If the event type of m is PortableMoveOut, the following action is taken.

```

PortableMoveOut( $p$ ) {
  if p.busy = YES then
    idleChannelNo ← idleChannelNo + 1;
    /* When a communicating portable moves */
    /* to a new cell, it releases the occupied */
    /* channel of the old cell. */
  end if
  determine the destination cell ( $LP'$ ) to which the
  portable moves;
  generate an output message
   $m' = (LVT, \bar{p}, \text{PortableMoveIn})$ ;
  invoke SendMessage( $m', LP'$ );
  /* A PortableMoveIn event is scheduled for  $LP'$ . */
}

```

Note that the timestamp of m' is the same as that of m .

In our implementation, the execution of the event message m results in the scheduling of exactly one future event m' . When m is executed, one or more attributes of the corresponding portable are updated. Then the next event for the portable is determined based on the updated values of the attributes. If the event type of m is PortableMoveOut, then a PortableMoveIn event message with the same timestamp is scheduled for the destination LP as described in the definition of PortableMoveOut(). For the other three event types, the message $m' = (ts, \bar{p}, \text{eventType})$ is determined by invoking ScheduleNewEvent():

```

ScheduleNewEvent( $p$ ) {
  if p.busy = NO then
    /* The portable is idle at LVT. The next */
    /* event occurring to  $p$  is either a call arrival */
    /* or a cell crossing movement. */
     $ts \leftarrow \min(p.\text{callArrivalTime},$ 
       $p.\text{portableMoveOutTime})$ ;
    if  $ts = p.\text{callArrivalTime}$  then
       $m'.\text{eventType}' \leftarrow \text{CallArrival}$ ;
    else  $m'.\text{eventType}' \leftarrow \text{PortableMoveOut}$ ;
    end if
  end if
}

```

```

ExecuteMessage( $m$ ) {
  LVTUpdate( $m.\text{timeStamp}$ )
  /* I.e., LVT ←  $m.\text{timeStamp}$ . */
  case ( $m.\text{eventType}$ ) of
    CallArrival: invoke CallArrival( $m.p$ );
    CallCompletion: invoke CallCompletion( $m.p$ );
    PortableMoveIn: invoke PortableMoveIn( $m.p$ );
    PortableMoveOut: invoke PortableMoveOut( $m.p$ );
  end case
}

```

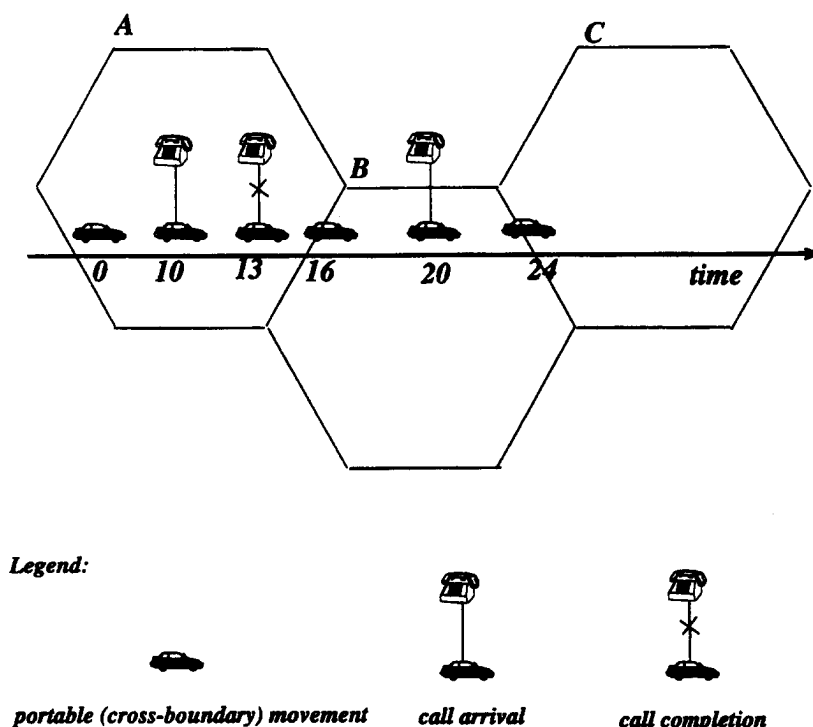


Fig. 3. A simple PCS example.

```

else /* I.e., p.busy=YES. The portable is busy at */
/* LVT. The next event occurring to p is either a */
/* call arrival, a call completion or a cell crossing */ and an event
/* movement. */
ts ← min(p.callArrivalTime,
        p.callCompletionTime,
        p.portableMoveOutTime);
if ts = p.callArrivalTime then
    m'.eventType' ← CallArrival;
else if ts = p.callCompletionTime then
    m'.eventType' ← CallCompletion;
else m'.eventType' ← PortableMoveOut;
end if
end if
}

```

Consider the example illustrated in Fig. 3. In this figure, a portable is represented by a car (although in many PCS systems, portables are carried by pedestrians). A call arrival is represented by a phone connected to the car. A call completion is represented by a cross (disconnection) on the phone line.

At time 0, portable p_1 is at cell A. At time 10, a phone call for p_1 occurs. The call completes at time 13, and the portable moves to cell B at time 16. At time 20, another phone call for p_1 arrives. At time 24, p_1 moves to cell C (and a handoff occurs).

In PDES, cells A, B, and C are simulated by logical process LP_A , LP_B , and LP_C respectively. At the beginning of the simulation, the attributes of p_1 are

```

busy = NO,
callArrivalTime = 10,

```

```

callCompletionTime = ?,
portableMoveOutTime = 16

```

$$m_1 = (10, \bar{p}_1, \text{CallArrival})$$

is scheduled and inserted in the FEL of LP_A . When the LVT of LP_A advances to 10, m_1 is executed by invoking LP_A . CallArrival(p_1). Suppose that an idle channel exists. The call is connected and the call holding time for the conversation is generated (which is 3, or the call completion time is $10 + 3 = 13$). The next call arrival time is also generated (which is 20 in Fig. 3). Thus the attributes of p_1 are modified as

```

busy = YES,
callArrivalTime = 20,
callCompletionTime = 13,
portableMoveOutTime = 16

```

and a new event

$$m_2 = (13, \bar{p}_1, \text{CallCompletion})$$

is scheduled and inserted in LP_A 's FEL. When the LVT of LP_A advances to 13, m_2 is executed. The method LP_A . CallCompletion(p_1) is invoked and the attributes of p_1 are modified as

```

busy = NO,
callArrivalTime = 20,
callCompletionTime = ?,
portableMoveOutTime = 16

```

and a new event

$$m_3 = (16, \bar{p}_1, \text{PortableMoveOut})$$

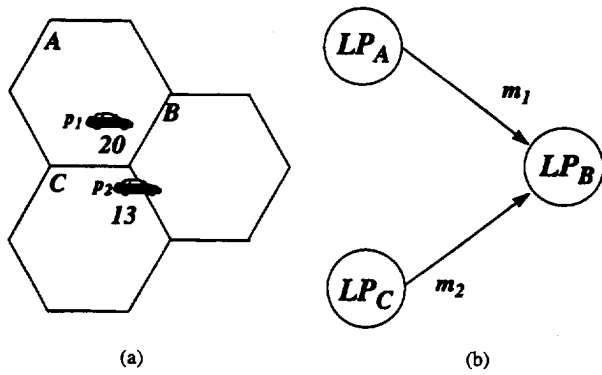


Fig. 4. PDES synchronization problem.

is scheduled. At LVT 16, m_3 is executed. The method LP_A .PortableMoveOut(p_1) is invoked to determine the destination cell (which is B in Fig. 3), and a message

$$m_4 = (16, \bar{p}_1, \text{PortableMoveIn})$$

is sent from LP_A to LP_B by invoking LP_A .SendMessage(m_4, LP_B). Note that the portable p_1 migrates to LP_B when m_4 is sent. (In GIT/Bellcore's PCS implementation [4], a message is part of a portable object, and sending a message automatically migrates the corresponding portable object.) When LP_B 's LVT advances to 16, it executes m_4 . The next portable move time is generated (which is 24). The attributes of p_1 are modified as

busy = NO,
callArrivalTime = 20,
callCompletionTime = ?,
portableMoveOutTime = 24

and a new event $m_5 = (20, \bar{p}_1, \text{CallArrival})$ is scheduled.

A PDES is correct if the following rule is satisfied.

Local Causality Constraint Every LP processes events in nondecreasing timestamp order.

The major problem of PDES is that the logical processes are executed at different speeds. Consider the scenario in Fig. 4 that portable p_1 moves from cell A to cell B at time 20 with an ongoing phone call (i.e., a handoff call), and portable p_2 moves from cell C to cell B at time 13 with an ongoing phone call (see Fig. 4(a)).

Consider the PDES scenario in Fig. 4(b). LP_A sends a PortableMoveIn event (message) m_1 (for p_1) with timestamp 20 to LP_B . Later LP_C sends m_2 (for p_2) with timestamp 13 to LP_B . If LP_B executes m_1 before m_2 arrives, then the modifications to LP_B .idleChannelNo is out of the timestamp order, and the local causality rule is violated. Thus the simulation result is not correct.

To solve this problem, the executions of the logical processes must be synchronized. The remainder of this paper describes two popular asynchronous synchronization mechanisms, the *conservative* and the *optimistic* methods.

IV. CONSERVATIVE METHOD

The conservative simulation [36] is *conservative* in the sense that it does not execute an event before it ensures that the local causality rule is satisfied. The conservative simulation follows

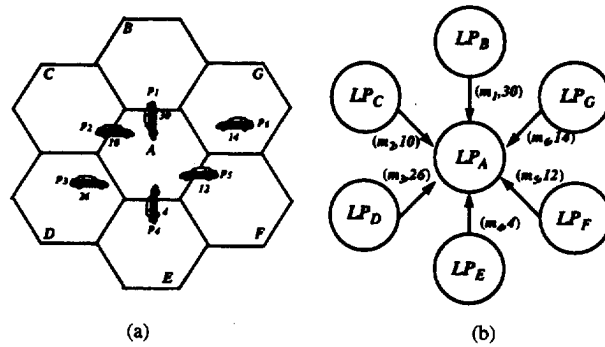


Fig. 5. The input waiting rule. In (a), the number below a car represents the time when the portable crosses the cell boundary.

two rules: the *input waiting rule* and the *output waiting rule*. It also assumes that

- the messages are received in the order they are sent (the *FIFO communication property*), and
- the communication channels among LP's are fixed and never change during the simulation. In Fig. 4(b), LP_A (LP_C) has one *output channel* directed to LP_B , and LP_B has two input channels (one from LP_A and one from LP_C).

A. Basic Synchronization Mechanism

In a conservative simulation, every logical process LP repeats the following two steps.

Step 1. LP waits to select an input message m from its input channels (extra data structures are required to implement input channels in a logical process) by invoking LP .ReceiveMessage(). This method is implemented based on the input waiting rule to be described. The method inserts m into LP 's FEL.

Step 2. Let ts be the timestamp of m . LP .ExecuteMessage() is invoked to process all events in the FEL with timestamps no larger than ts in nondecreasing timestamp order. The execution may invoke LP .SendMessage() to send output messages. This method is implemented based on the output waiting rule to be described. If the termination condition is satisfied (e.g., $LP.LVT > 5000$), then exit the loop. Otherwise go to Step 1.

The waiting rules are described as follows.

The Input Waiting Rule: An LP does not process any input message until it has received at least one message from each of its input channels. The input message with the smallest timestamp is selected for processing. Fig. 5 shows how the input message is selected for the PCS simulation.

Fig. 5(a) illustrates a PCS system where 6 portables p_1, p_2, p_3, p_4, p_5 , and p_6 move from cells B, C, D, E, F , and G to cell A at times 30, 10, 26, 4, 12, and 14, respectively. In the PDES model (see Fig. 5(b)), the PortableMoveIn events of $p_1, \dots, p_6$ are represented by the messages $m_1, \dots, m_6$ sent to LP_A . By the input waiting rule, m_4 is the next message to be executed in LP_A .

Assume that all messages sent from one LP to another are in nondecreasing timestamp order (this property will be

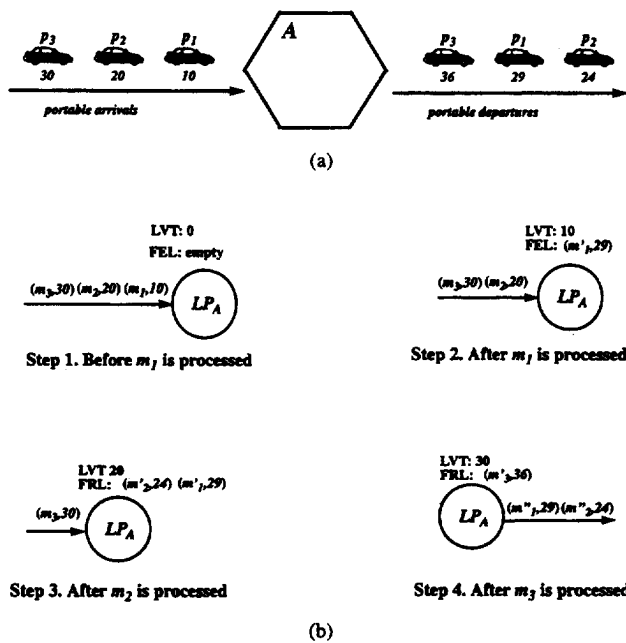


Fig. 6. The output waiting rule.

guaranteed by the output waiting rule to be described next), then the input waiting rule ensures that the timestamp of the selected message is no larger than any input messages to be processed in the future.

The Output Waiting Rule: An LP does not send an output message to another LP until it ensures that no output messages with smaller timestamps will be scheduled (at LP) in the future. Assume that all input messages are handled in nondecreasing timestamp order (the property is guaranteed by the input waiting rule). The output waiting rule is satisfied if an LP only sends output messages with timestamps no larger than its current LVT value.

Consider the following PCS example. Portables p_1, p_2 and p_3 move into cell A at times 10, 20, and 30, and move out of the cell at times 29, 24, and 36, respectively (see Fig. 6(a)). This situation occurs since a portable, once inside cell A, may take a dramatically different from other portables. Some portables may stay in the same physical location for a period while other portables continue moving toward an adjacent cell to A.

In PDES, m_1, m_2 , and m_3 are input messages representing the arrivals of p_1, p_2 and p_3 , respectively (see Step 1 in Fig. 6(b)). When m_1 is processed, a move event m'_1 for p_1 is scheduled with timestamp 29 (see Step 2 in Fig. 6(b)). In the conservative simulation, m'_1 cannot be sent to the destination LP immediately, or the output waiting rule may be violated. In our PCS PDES implementation, the portable move is simulated by two types of events: a PortableMoveOut event and a PortableMoveIn event. In Fig. 6(b), m'_i and m''_i represent the PortableMoveOut event and PortableMoveIn event of portable p_i , respectively. When the event m'_i is scheduled, it is inserted in LP_A 's FEL. When the LVT of LP_A advances to the portable "move time" (i.e., the timestamp of m'_i), m'_i is processed, which results in sending

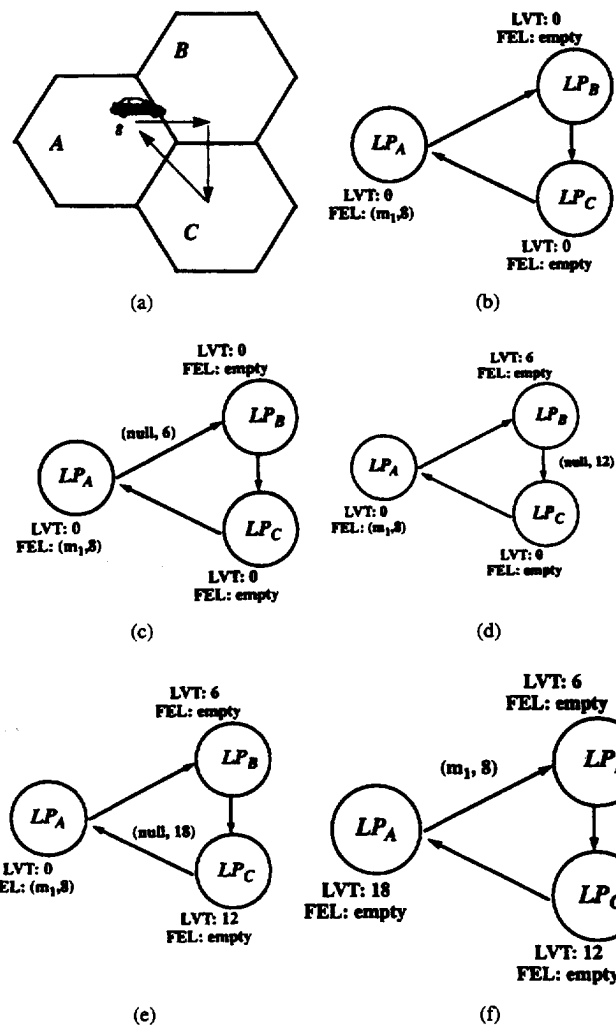


Fig. 7. Deadlock and deadlock resolution.

the PortableMoveIn event m''_i (with the timestamp of m'_i) to the destination. In Fig. 6(b), m'_2 and m'_1 are sent after Step (3) and before Step (4); i.e., when LP_A is sure that next input message to be handled has timestamp larger than m'_1 and m'_2 . Note that m'_2 is sent before m'_1 is.

Since the output waiting rule is guaranteed by using the two "move" event types, the conservative SendMessage() method simply sends the output message to the destination. Note that for other applications, a different conservative SendMessage() method may be required to implement the output waiting rule.

The correctness of the conservative simulation can be proved by induction on the interaction of the two waiting rules.

B. Deadlock and Deadlock Avoidance

The input waiting rule may result in deadlock (LP's are waiting for input messages from each other and cannot progress) even if the simulated system is deadlock free.

Consider a three-cell PCS network (see Fig. 7(a)).

There is one portable in the network, and the portable moves in the path $A \rightarrow B \rightarrow C \rightarrow A$. At time 0,

the portable is in cell A. The portable moves from cell A to cell B at time 8. In the conservative simulation, a `PortableMoveOut` event m_1 is scheduled in LP_A initially (see Fig. 7(b)). By the input waiting rule, LP_A waits for an input message from LP_C before it can process m_1 . Similarly, LP_C does not send out any output message before it receives an input message from LP_B , and LP_B does not send out any output message before it receives an input message from LP_A (i.e., before m_1 is processed). Thus the PDES is in the deadlock situation. Two deadlock resolutions have been proposed: *deadlock avoidance* [36] and *deadlock detection/recovery* [37], [38]. It has been shown [39] that the cost of deadlock detection/recovery is much higher than deadlock avoidance. This article will focus on the deadlock avoidance mechanism.

In a PCS network, a portable is expected to reside in a cell for a period of time before it moves. Assume that every portable resides in a cell for at least six time units before it moves to a new cell. The information that “a portable resides in a cell for at least 6 time units” is used in the deadlock avoidance mechanism to predict when an LP will receive an input message, and “6 time units” is referred as the *lookahead* value. The lookahead information is carried by the control messages called *null messages*. A null message does not represent any event in the simulated system. Instead, it is used to break deadlock as well as to improve the progress of a conservative simulation.

In Fig. 7(b), at the beginning of PDES, the LVT's of the three LP's are 0, and a `PortableMoveOut` event m_1 with timestamp 8 is in LP_A 's FEL. At time 0, LP_A sends a null message with timestamp $0 + 6 = 6$ (the LVT value plus the lookahead value) to LP_B (see Fig. 7(c)). The null message implies that no portable will move in cell B earlier than time 6. Thus, the LVT of LP_B advances to 6 when the null message arrives (Fig. 7(d)). Since no portable arrives at cell B before time 6, it implies that no portable will move out of cell B before time 12 and LP_B sends a null message with timestamp 12 to LP_C . After the sending of several null messages, LP_A will eventually receive a null message with timestamp larger than 8 (see Fig. 7(e)), and by the input and output waiting rules, m_1 is sent from LP_A to LP_B and the deadlock is avoided (see Fig. 7(f)).

C. Exploiting Lookahead

It is important to exploit the lookahead to improve the progress of a conservative simulation. Experimental studies have indicated that the larger the lookahead values, the better the performance of the conservative simulation [39]. Based on the techniques proposed in [40]–[42], we give three PCS examples for lookahead exploration. The first two examples assume single cell entrance and exit. The single entrance/exit PCS model has been used in modeling highway cellular phone systems [43]. The results can be easily generalized for multiple entrances and exits. The techniques introduced can be combined to exploit greater lookahead.

1. *Lookahead Method 1 (FIFO)*: In a large scale PCS network, a cell may only cover a street, and the portables

leave the cell in the order they move in (the FIFO property; see Fig. 8(a)).

Consider the corresponding FIFO LP for cell A in PDES. The lookahead for the LP can be derived by a presampling technique proposed by Nicol [41]. The idea is to presample the residence times of the arrival portables.

If the FEL is not empty, then the next departure time can be easily computed. In the PCS PDES, the move-out timestamp of a portable is computed and stored in `portableMoveOutTime` of the portable object at the time when the `PortableMoveIn` event is processed. The FIFO property guarantees that the next departure time is the minimum of the `portableMoveOutTime` values of portable objects in the FEL. Thus, the precomputed next departure times can be used as the lookahead. If the FEL of the LP is empty at timestamp $LP.LVT$, then the lookahead can be generated by the same presampling technique. Since the portable will arrive at the cell later than $LP.LVT$, it will leave the cell later than $LP.LVT + t$ (where t is the presampled portable residence time). The FIFO property guarantees that after time $LP.LVT$, no portables will depart earlier than $LP.LVT + t$, and the LP may send null messages with this timestamp to the downstream LP's.

2. *Lookahead Method 2 (Minimum Inter-Boundary Crossing Time)*: Consider the example in Fig. 8(b) where the FIFO portable movement property in the previous example does not hold. In practice, the inter arrival times to the cell (for the portables from the same entrance) cannot be arbitrary small. Instead, a minimum cell crossing time τ is assumed. Let p_i ($i = 1, 2, 3, \dots$) be the i th portable arrival after time $LP.LVT$. The portable residence time for p_i is t_i . Then the departure time of p_i is later than $LP.LVT + (i - 1)\tau + t_i$, and the next departure time at the cell after $LP.LVT$ is later than $LP.LVT + \Delta$ where

$$\Delta = \min_{1 \leq i < \infty} [(i - 1)\tau + t_i]. \quad (1)$$

Since $\tau > 0$, there exists j such that

$$j\tau \geq \min_{1 \leq i \leq j} [(i - 1)\tau + t_i] = \Delta.$$

In other words, to compute Δ it suffices to consider the first j presampled residence timestamps in (1). Fig. 9 displays a situation where we employ formula (1). Four portables arrive using times 10, 14, 19 and 22. Let $\tau = 3$ so that we know that no two consecutive portable arrivals will be less than 3. The residence times for the portables are placed in parentheses in Fig. 9. The variable j is increased by 1 until the above inequality is satisfied. Suppose that LP_A needs to send a null message to its downstream before it receives the `PortableMoveIn` event for p_1 . The residence times of the subsequent arriving portables are pre-samples as $t_1 = 9, t_2 = 4, t_3 = 1, t_4 = 5 \dots$. Our algorithm proceeds as follows:

- (a) For $j = 1$, $1 \times 3 \geq \min[0 + 9] = 9$? No.
- (b) For $j = 2$, $2 \times 3 \geq \min[0 + 9, 3 + 4] = 7$? No.

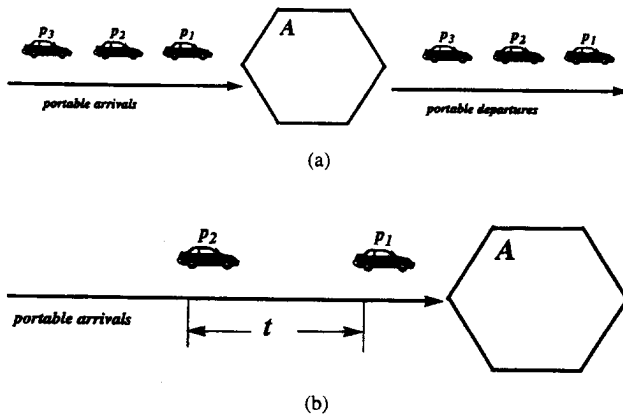


Fig. 8. Examples for lookahead exploiting.

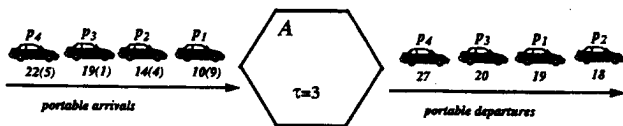


Fig. 9. Portables entering and leaving cell A.

- (c) For $j = 3$, $3 \times 3 \geq \min[0 + 9, 3 + 4, 6 + 1] = 7$? Yes.

From this procedure, we derive $\Delta = 7$ by using the first three pre-sampled residence times.

3. **Lookahead Method 3 (Minimum Residence Time):** If the FIFO portable movement property does not preserve, and τ does not exist (or is too small to be useful), then the technique proposed in the previous example may not work. In a PCS simulation, the total number $N = S \times n$ of portables is an input parameter. To compute the next lookahead value for an LP, it suffices to sample the next N portable residence times, and (1) is re-written as [42]

$$\Delta = \min_{1 \leq i \leq N} t_i$$

The last two examples may require a large number of operations to generate a lookahead value. In [40], $O(1)$ algorithms have been proposed to generate the lookahead values.

When the `ExecuteMessage()` method processes a null message in an LP, it invokes a method `ComputeLookahead()` to compute the timestamp of the output (null) messages. The `ComputeLookahead()` method may implement the lookahead exploiting techniques described above. Then the new null message is sent to some or all output channels by invoking the `SendMessage()` method.

V. OPTIMISTIC METHOD

The optimistic simulation [44] is *optimistic* in the sense that it handles the arrival events aggressively. When a message m arrives at an LP, `LP.ReceiveMessage()` simply inserts m in the *input queue* (the optimistic simulation terminology for the FEL). The logical process assumes that the events already in its input queue are the “true” next events. The `ExecuteMessage()` method proceeds to execute these events in

timestamp order, and `SendMessage()` is invoked whenever an output message is scheduled. When a message arrives at the LP, the timestamp of the message may be less than some of the events already executed. (This arrived message is referred to as a *straggler*.) The optimism was unjustified, and therefore a method `Rollback()` is invoked by `ExecuteMessage()` to cancel the erroneous computation. To support rollback, data structures such as the *state queue* and the *output queue* are required (to be elaborated).

Several strategies for cancelling incorrect computation were surveyed by Fujimoto [45]. Two popular cancellation strategies called *aggressive cancellation* [44] and *lazy cancellation* [46] are described in this section.

A. Cancellation Strategies

Consider the example in Fig. 10. For simplicity, assume that cell C has one radio channel (i.e., $LP_C.channelNo = 1$ in PDES). In this example, portable p_2 moves from cell B to cell C at time 10 (event 1), and make a phone call at time 13. The call is completed at time 21. Portable 1 moves from cell A to cell C at time 16 (event 2), and attempts to make a phone call at time 20. Since the only radio channel is used by portable 2, the call attempt from portable 1 is blocked. Portable 1 moves from cell C to cell D at time 24. Figs. 11, 12, and 13 illustrate the data structures of LP_C (the logical process corresponding to cell C) assuming that message m_1 (the message that represents event 2) arrives at LP_C earlier than message m_5 (the message that represents event 1) does. In LP_C , a state queue and an output queue are maintained to supported rollback. In our example, the state variable (attribute) for LP_C is the number of idle channels $LP_C.idleChannelNo$. The state variable is checkpointed and saved in the state queue from time to time. The snapshots in the state queue are used to recover the state of LP_C when rollbacks occur. The output queue records the *anti-messages* of the output messages that have been sent from LP_C . The anti-messages are used to annihilated *false* messages sent in the incorrect computation.

In Fig. 11(a), LP_C receives m_1 that is inserted in LP_C 's input queue. Initially, the output queue of LP_C is empty, and the value of $LP_C.idleChannelNo$ at timestamp 0 is saved. After m_1 is executed, the system state at timestamp 16 is checkpointed, and a call arrival event (message m_2) is scheduled for LP_C itself (see Fig. 11(b)). Note that after its execution, m_1 is kept in the input queue (this message may be re-executed if a rollback occurs). A pointer in the input queue indicates the next event to be executed. The anti-message m_2^- of m_2 is saved in LP_C 's output queue. The message m_2^- is identical to m_2 except that it includes a destination field (in the original optimistic or Time Warp algorithm [44], the sender and the destination are recorded in both the output message and the corresponding anti-message for flow control). To summarize, the `ExecuteMessage()` method for the optimistic simulation saves the system state after an event execution (note that the state may be saved after several event executions), and the executed event is not deleted from the input queue. The `SendMessage()` method saves the anti-messages in the output queue when it sends an output message.

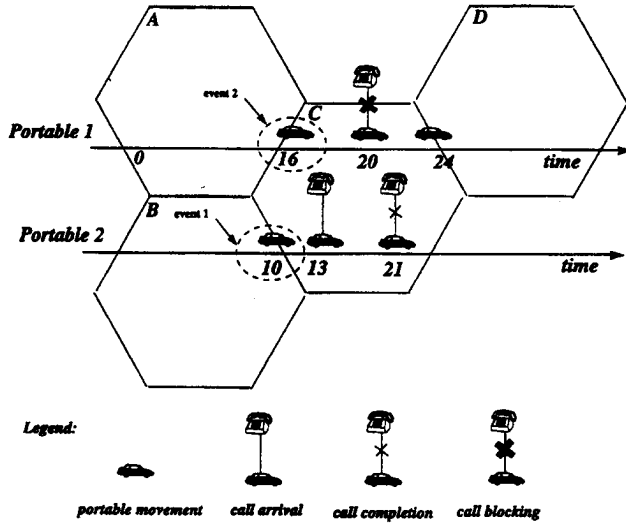


Fig. 10. A PCS example for optimistic PDES. Events 1 and 2 will be represented by messages m_5 and m_1 respectively in the optimistic PDES (see the next figures).

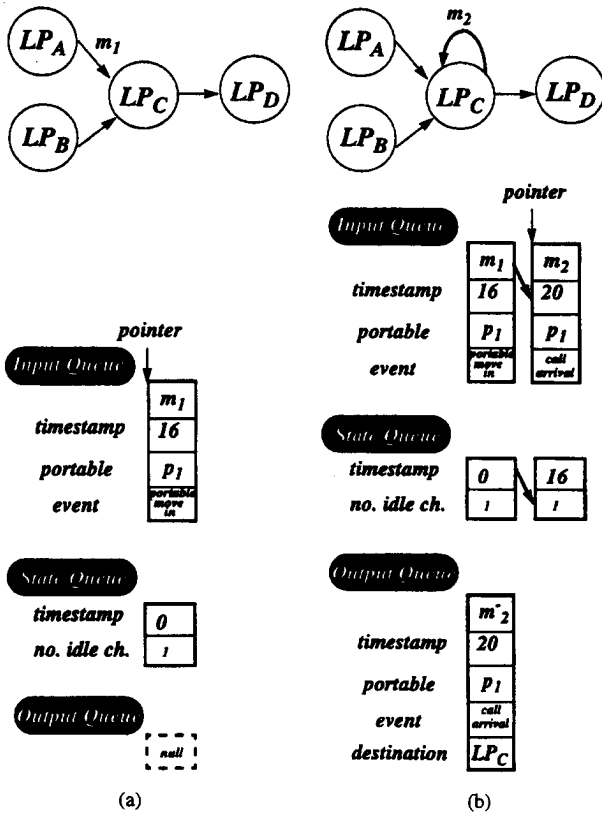


Fig. 11. The data structures of LP_C before/after rollback.

After m_2 is executed, the number of idle channel is decremented by 1, and

$$LP_C.idleChannelNo = 0$$

is saved in the state queue. A PortableMoveOut event m_3 is scheduled at timestamp 24, and its anti-message m_3^- is stored in the output queue (see Fig. 12(a)).

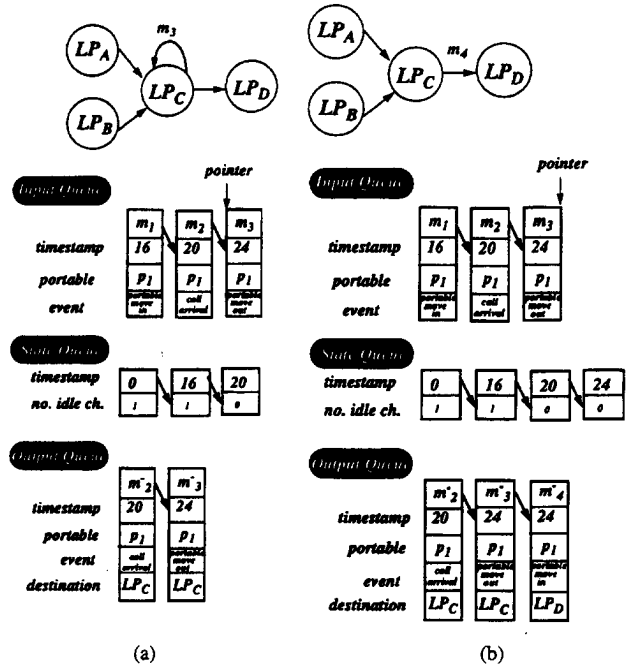


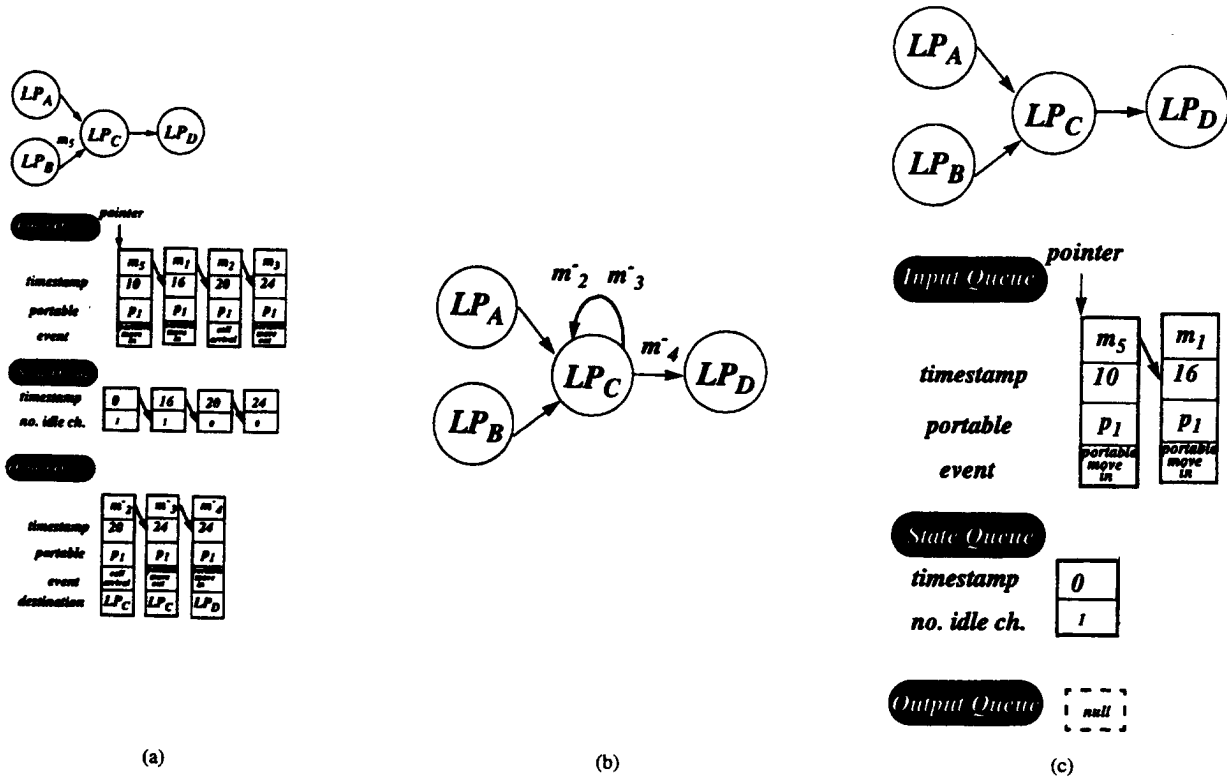
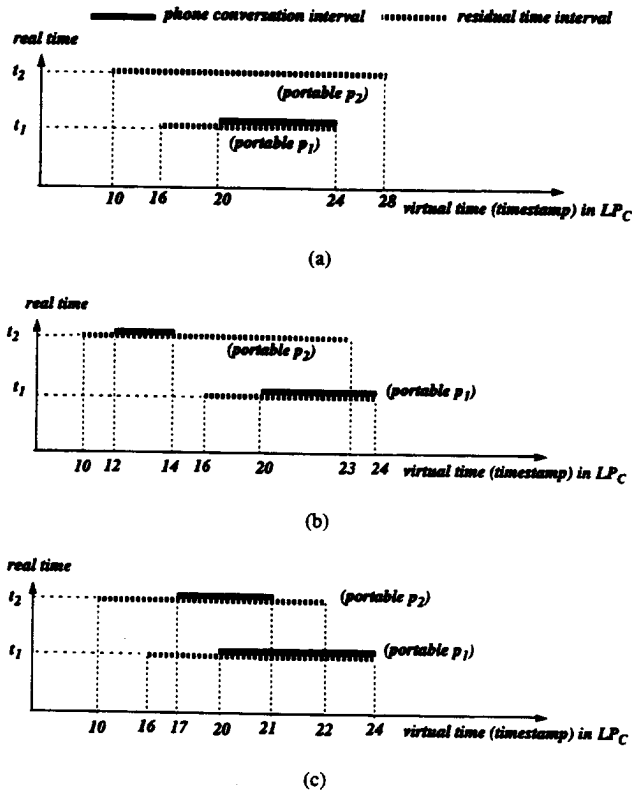
Fig. 12. The data structures of LP_C before/after rollback (cont.).

When m_3 is executed, a PortableMoveIn message m_4 is sent to LP_D (see Fig. 12(b)). After m_4 is sent, the straggler m_5 (the event that p_2 moves in LP_C at timestamp 10) arrives. Since $LP_C.LVT = 24$, the out-of-order execution is detected (see Fig. 13(a)) by $LP_C.ReceiveMessage()$, and $LP_C.Rollback()$ is invoked. Two strategies for cancelling incorrect computation are described below.

Aggressive Cancellation: When a straggler arrives, aggressive cancellation assumes that the out-of-order computation, as well as all other computations that may have been affected by this computation are not correct. Thus, the out-of-order computation is recomputed, and $LP_C.Rollback()$ cancels the affected computations immediately by sending anti-messages. In our example, a rollback of LP_C at timestamp 10 occurs. In Fig. 13(b), the anti-messages m_2^- , m_3^- , and m_4^- are deleted from the output queue, and are sent to their destinations to annihilate false messages m_2 , m_3 , and m_4 , respectively. After the rollback (see Fig. 13(c)), messages m_2 and m_3 (and m_4 in LP_D) are removed from the input queue. The state of LP_C at timestamp 0 is re-stored. Then $LP_C.ExecuteMessage()$ resumes the simulation by executing m_5 .

Lazy Cancellation: It is possible that the erroneous computation still generated correct output messages. In that case, it is not necessary to cancel the original message that was sent. In lazy cancellation, logical processes do not immediately send the anti-messages for any rolled back computation. Instead, they wait to see if the reexecution of the computation causes any of the same messages to be regenerated. If the same message is recreated, there is no need to cancel the original. Otherwise, an anti-message is sent. In our example, lazy cancellation applies to three situations.

- 1) If portable p_2 arrives at cell C (LP_C) at time 10 and leaves cell C at time 28 without making any phone call

Fig. 13. The data structures of LP_C before/after rollback (cont.).Fig. 14. Situations when lazy cancellation applies (in these situations, $t_2 > t_1$).

(see Fig. 14(a)) then the arrival of m_5 in Fig. 13(a) will not affect the executions of m_1, m_2 , and m_3 . (Note that

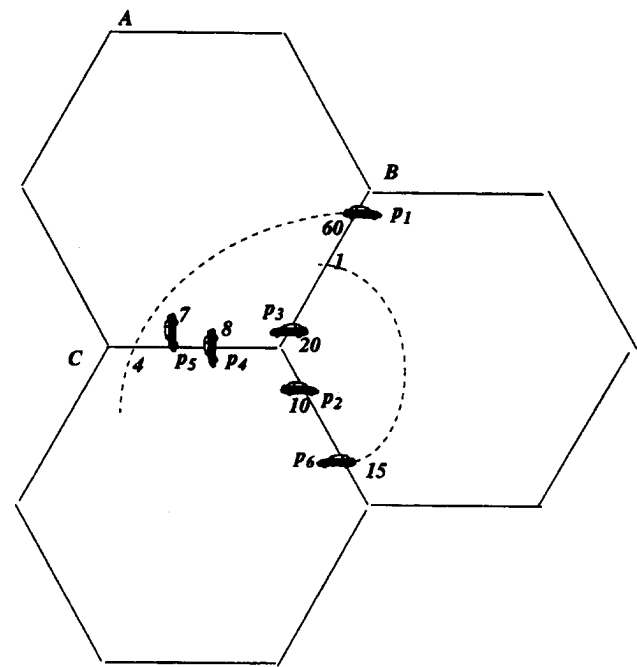


Fig. 15. An PCS example for fossil collection in optimistic PDES.

in PDES, whether a call for p_2 occurs in the interval $[10], [28]$ can be detected in the portable object.) Thus messages m_1, m_2 , and m_3 do not need to be reexecuted after m_5 is executed. This is called *jump forward* or *lazy reevaluation* [1].

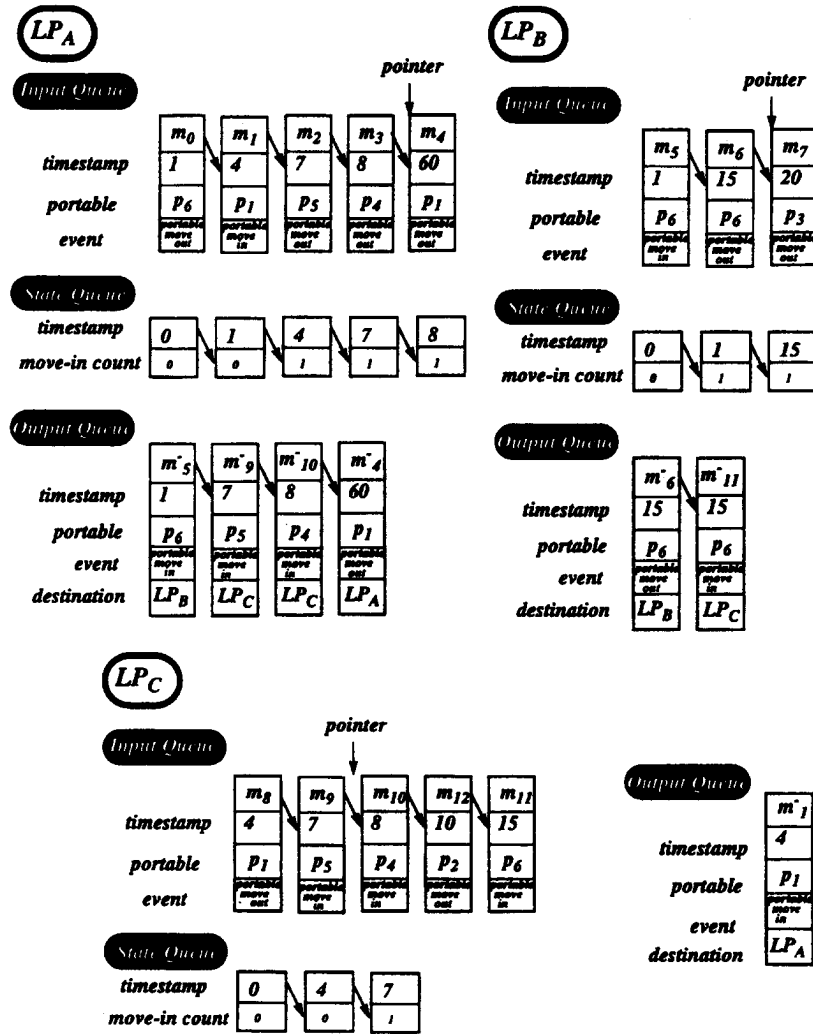


Fig. 16. The optimistic PDES before fossil collection.

In this case, $LP_C.ReceiveMessage()$ simply inserts m_5 in the input queue, and the pointer of the input queue points to m_5 . $LP_C.ExecuteMessage()$ executes m_5 and the pointer jumps directly after m_3 without re-executing m_1, m_2 , and m_3 .

- The call for p_2 does not block the call for p_1 if p_2 's call completes before p_1 's call arrives (see Fig. 14(b)) or p_2 's call overlaps p_1 's call but LP_C has two or more radio channels (i.e., $LP_channelNo \geq 2$; see Fig. 14(c)). In these cases, the channel utilization (not shown as a state variable in our example) changes, but the subsequent messages (i.e., m_2, m_3 , and m_4) scheduled due to the execution of m_1 are not affected. Thus, messages m_1, m_2 , and m_3 are re-executed to reflected the correct channel utilization. No anti-messages need to be sent (i.e., m_2^-, m_3^- , and m_4^- are not sent out). Like the previous case, $LP_C.ReceiveMessage()$ simply inserts m_5 in the input queue. After m_5 has been executed, $LP_C.ExecuteMessage()$ will re-execute m_1, m_2 , and m_3 without re-generating any output messages.

If lazy cancellation does succeed most of the time, then the performance of the optimistic simulation is improved by eliminating the cost of cancelling the computation which would have to be reexecuted. If lazy cancellation fails, then the performance degrades, because erroneous computations are not cancelled as early as possible. In our PCS simulation, we may exploit situations that lazy cancellation does not fail (as described above), and a logical process can be switched between aggressive cancellation and lazy cancellation to reduce the rollback cost.

B. Memory Management

To support rollback, it is necessary to save the "history" (the already executed elements in the input, the output, and the state queues) of a logical process. However, it may not be practical to save the whole history of a logical process because memory is likely to be exhausted before the simulation completes. Thus, it is important that we only save "recent history" of logical processes to reduce the memory usage.

Memory management for the optimistic simulation is based on the concept of *global virtual time* (GVT). The GVT at

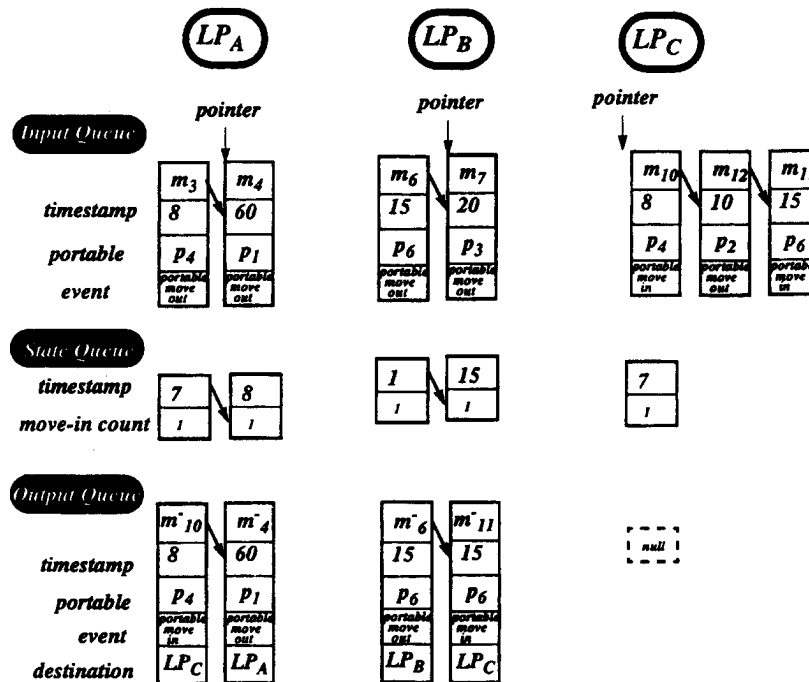


Fig. 17. The optimistic PDES after fossil collection.

(execution) time t is the minimum of the timestamps of the not-yet executed messages (these messages are either in the input queue or are in transit) in the optimistic simulation at time t . (Several other operational definition of GVT are given in [47], [48].) It has been pointed out [44] that at any given time t , a logical process cannot be rolled back to a timestamp earlier than the GVT at t . Therefore the storage for all messages with timestamps smaller than the GVT value can be reclaimed for other usage. The process of reclaiming the storage for the obsolete elements is called *fossil collection*.

The GVT computation is not trivial in a distributed system because it may be difficult to capture the messages in transit. Several GVT algorithms have been developed in the systems with the FIFO communication property [49] or without the FIFO communication property [50], [51].

In GIT/Bellcore PCS PDES (where eight workstations are connected by a local area network), all logical processes are frozen during GVT computation. By utilizing the low level communication mechanism, all transient messages are guaranteed to arrive at their destinations before the GVT computation starts. The fossil collection procedure works as follows. A coordinator initiates the procedure by freezing the execution of every logical process. After all transient messages arrive at their destinations, every logical process reports its *local minimum value* (the minimum of the timestamps of all unprocessed messages in the input queue) to the coordinator. The coordinator then compute the GVT value as the minimum of the received local minimums. The GVT value is broadcast to all logical processes for fossil collection.

To illustrate the storage reclaimed in fossil collection, consider the example in Fig. 15. In this example, we ignore the phone call events and assume that all Portable-

MoveIn/PortableMoveOut events must be executed in their timestamp order in the optimistic simulation. We further assume that the state variable of a logical process is the number of portables move in the corresponding cell after time 0. Portable 1 moves from cell C to cell A at time 4 and moves from cell A to cell B at time 60. Portable 2 moves from cell C to cell B at time 10. Portable 3 moves from cell B to cell A at time 20. Portable 4 moves from cell A to cell C at time 8. Portable 5 moves from cell A to cell C at time 7. Portable 6 moves from cell A to cell B at time 1 and moves from cell B to cell C at time 15.

Fig. 16 illustrates the elements in the input/output/state queues of LP_A , LP_B , and LP_C after all transient messages arrive at their destinations, and the GVT value (which is $8 = \min(60, 20, 8)$) is found.

Fig. 17 illustrates the elements in the input/output/state queues of LP_A , LP_B , and LP_C after the fossil collection procedure is completed.

All messages with timestamps smaller than 8 were fossil collected. Note that fossil collection for the state queue is not exact the same as that for the input/output queues. In the state queue, the element with the largest timestamp smaller than the GVT value (i.e., 8) must not be removed (see Fig. 17). The other elements with timestamps smaller than 8 are removed.

C. Performance Evaluation

The performance of an optimistic PCS PDES implementation has been investigated in [4]. In this study, a version of Time Warp has been developed that executes on 8 DEC 5000 workstations connected by an Ethernet.

In the experiments, *speedp* was used as the output measure where the sequential simulator used the same priority queue

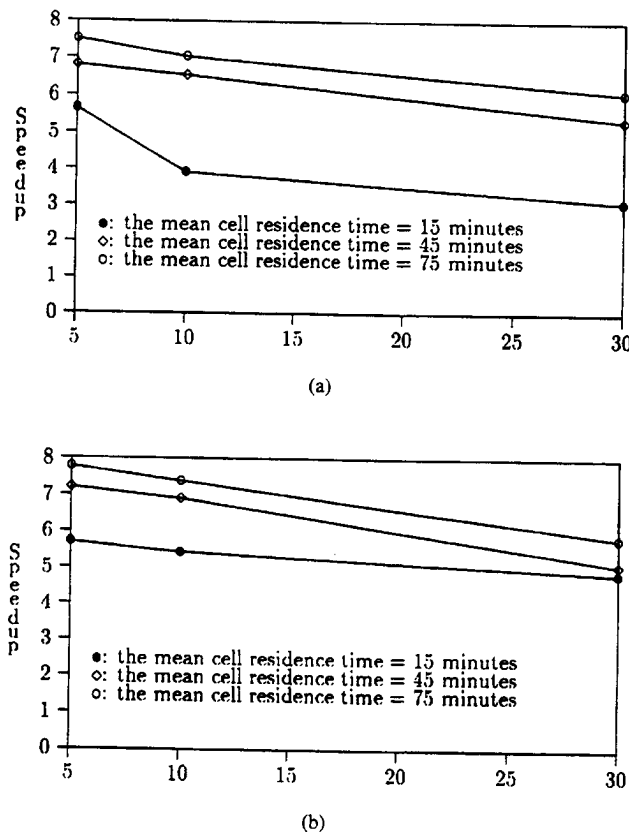


Fig. 18. Speedup of the optimistic PDES (The call holding time is exponentially distributed with mean 3 min. Eight processors are used in the parallel simulation.) The expected number of portables per cell is 50 in (a), and 75 in (b).

mechanism as that of PDES for managing the pending set of events, but did not have the state saving, rollback and fossil collection overheads associated with the PDES implementation. The 1024 cells are simulated for 2.5×10^5 simulated seconds. Fig. 18 shows the performance of the optimistic PDES. The figure indicates good performance of PDES for the PCS application. PDES is particularly efficient when the number of portables is large, the cell residence time is long, and the call interarrival time is short.

VI. FUTURE DIRECTIONS FOR PDES

This paper describes the asynchronous parallel discrete event simulation (PDES) mechanisms and optimization techniques by examples of personal communications services (PCS) network simulation. We described the conservative and the optimistic PDES mechanisms and several optimizations tailored for the PCS simulation. The performance of the optimistic method was briefly discussed. Since the conservative optimizations (tailored for PCS) introduced in this paper are new and were not previously reported, no performance studies have been conducted. Investigating the performance of these optimizations will be one of our future research directions.

The optimization techniques described in the paper are general and apply to other simulation applications such as battlefield simulation, VLSI simulation, queueing network simulation and computer architecture simulation. However,

these optimization techniques may need to be tailored for specific applications. Many studies have devoted to this issue (see [1], [2], [52]–[54] and references therein). The PCS example can be seen as being a member of a larger class of simulation model where one first discretizes the spatial domain into a grid, and then simulates moving entities from one grid cell to another. In this sense, the PCS problem is isomorphic to the problems of particle/n-body simulation.

An important research direction that has not been fully exploited is the building of user-friendly PDES environments. Such an environment should provide convenient tools to develop simulation application. Methods should also be provided to tailor general optimization techniques to fit a specific simulation application. We anticipate that these user-friendly environments can be constructed by the object-oriented models described in [6].

ACKNOWLEDGMENT

C. Carothers and Y. C. Wong provided useful comments to improve the quality of this paper.

REFERENCES

- [1] R. M. Fujimoto, "Parallel discrete event simulation," *Comm. ACM*, vol. 33, no. 10, pp. 31–53, Oct. 1990.
- [2] D. M. Nicol and R. M. Fujimoto, "Parallel simulation today," *Ann. Oper. Res.*, vol. 53, pp. 249–286, Dec. 1994.
- [3] R. Richter and J. C. Walrand, "Distributed simulation of discrete event systems," in *Proc. IEEE*, Jan. 1989, vol. 77, no. 1, pp. 99–113.
- [4] C. Carothers, R. M. Fujimoto, Y.-B. Lin, and P. England, "Distributed simulation of PCS networks using time warp," in *Proc. Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1994, pp. 2–7.
- [5] C. Carothers, Y.-B. Lin, and R. M. Fujimoto, "A re-dial model for personal communications services network," to appear in *45th Vehicular Technology Conf.*, 1995.
- [6] P. A. Fishwick, *Simulation Model Design and Execution: Building Digital Worlds*. Englewood Cliffs, NJ: Prentice Hall, 1995.
- [7] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
- [8] A. M. Law and D. W. Kelton, *Simulation Modeling & Analysis*, 2nd ed. New York: McGraw Hill, 1991.
- [9] T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*, 2d ed. Cambridge, MA: MIT Press, 1987.
- [10] P. A. Fishwick and B. P. Zeigler, "A multimodel methodology for qualitative model engineering," *ACM Trans. Modeling Comp. Simulation*, vol. 2, no. 1, pp. 52–81, 1992.
- [11] P. A. Fishwick, "A simulation environment for multimodeling," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 3, pp. 151–171, 1993.
- [12] M. Ebling, M. Di Loreto, M. Presley, F. Wieland, and D. Jefferson, "An ant foraging model implemented on the time warp operating system," in *Proc. 1991 SCS Multiconf. on Distributed Simulation*, Mar. 1991, pp. 21–26.
- [13] P. Hontalas, B. Beckman, M. Di Loreto, L. Blume, F. Reiher, K. Sturdevant, L. Warren, J. Wedel, F. Wieland, and D. Jefferson, "Performance of the colliding pucks simulation on the time warp operating systems (Part 1: Asynchronous behavior & sectoring)," in *Proc. 1989 SCS Multiconference on Distributed Simulation*, Mar. 1989, pp. 3–7.
- [14] R. M. Fujimoto, "Time warp on a shared memory multiprocessor," in *Proc. 1989 Int. Conf. on Parallel Processing*, Aug. 1989, vol. III, pp. 242–249.
- [15] R. Ayani and H. Rajaei, "Parallel simulation of a generalized cube multistage interconnection network," in *Proc. 1990 SCS Multiconference on Distributed Simulation*, Jan. 1990, pp. 60–63.
- [16] G. S. Thomas and J. Zahorjan, "Parallel simulation of performance Petri Net: Extending the domain of parallel simulation," in *Proc. 1991 Winter Simulation Conf.*, 1991, pp. 564–573.
- [17] D. A. Reed and A. Malony, "Parallel discrete event simulation: The Chandy-Misra approach," in *Proc. 1988 SCS Multiconf. on Distributed Simulation*, Feb. 1988, pp. 8–13.

- [18] E. Wieland, L. Hawley, A. Feinberg, M. Di Loreto, L. Blume, P. Reiher, B. Beckman, P. Hontalas, S. Bellenot, and D. Jefferson, "Distributed combat simulation and time warp: The model and its performance," in *Proc. 1989 SCS Multiconf. on Distributed Simulation*, Mar. 1989, pp. 14–20.
- [19] L. Soule and A. Gupta, "An evaluation of the Chandy-Misra-Bryant algorithm for digital logic simulation," *ACM Trans. Modeling Comp. Simulat.*, vol. 1, no. 4, pp. 308–347, 1991.
- [20] D. Beazner, G. Lomow, and B. Unger, "A parallel simulation environment based on time warp," to appear in *Int. J. Comp. Sim.*, 1995.
- [21] S. Turner and M. Xu, "Performance evaluation of the bounded time warp algorithm," *The 6th Workshop on Parallel and Distributed Simulation*, 1992.
- [22] B. Lubachevsky, "Efficient distributed event-driven simulations of multiple-loop networks," *Comm. ACM*, vol. 21, no. 2, Mar. 1989.
- [23] K. Ghosh, K. Panesar, R. M. Fujimoto, and K. Schwan, "PORTS: A parallel, optimistic, real-time simulator," in *Proc. 8th Workshop on Parallel and Distributed Simulation*, 1994.
- [24] G. Gaujal, A. G. Greenberg, and D. M. Nicol, "A sweep algorithm for massively parallel simulation of circuit-switched networks," *J. Parallel and Distributed Computing*, vol. 18, no. 4, pp. 484–500, 1993.
- [25] D. C. Cox, "Personal communications—A viewpoint," *IEEE Commun. Mag.*, vol. 128, no. 11, pp. 8–20, 1990.
- [26] ———, "A radio system proposal for widespread low-power tetherless communications," *IEEE Trans. Commun.*, vol. 39, no. 2, pp. 324–335, Feb. 1991.
- [27] P. W. Glynn and P. Heidelberger, "Analysis of initial transient deletion for parallel steady-state simulation," *SIAM J. Scien. Stat. Comp.*, vol. 13, no. 4, pp. 904–922, 1992.
- [28] P. Heidelberger, "Discrete event simulations and parallel processing: Statistical properties," *SIAM J. Scien. Stat. Comp.*, vol. 9, no. 6, pp. 1114–1132, Nov. 1988.
- [29] Y.-B. Lin, "Parallel independent replicated simulation on a network of workstations," *Simulation*, vol. 64, no. 2, pp. 102–110, 1995.
- [30] W. C. Wong, "Packet reservation multiple access in a metropolitan microcellular radio environment," *IEEE J. Select. Areas Commun.*, vol. 11, no. 6, pp. 918–925, 1993.
- [31] ———, "Dynamic allocation of packet reservation multiple access carriers," *IEEE Trans. Veh. Technol.*, vol. 42, no. 4, 1993.
- [32] Y.-B. Lin, "Determining the user locations for personal communications networks," *IEEE Trans. Veh. Technol.*, vol. 43, no. 3, pp. 466–473, 1994.
- [33] Y.-B. Lin, S. Mohan, and A. Noerpel, "Sub-rating channel assignment strategy for PCS hand-offs," *IEEE Trans. Veh. Technol.*, vol. 45, no. 1, pp. 122–130, 1996.
- [34] ———, "Queueing priority channel assignment strategies for handoff and initial access for a PCS network," *IEEE Trans. Veh. Technol.*, vol. 43, no. 3, pp. 704–712, 1994.
- [35] Y.-B. Lin, A. Noerpel, and D. Harasty, "Sub-rating channel assignment strategy for hand-offs," to appear in *IEEE Trans. Veh. Technol.*, 1995.
- [36] K. M. Chandy and J. Misra, "Distributed simulation: A case study in design and verification of distributed programs," *IEEE Trans. Software Eng.*, vol. SE-5, no. 5, pp. 440–452, Sept. 1979.
- [37] K. M. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Comm. ACM*, vol. 24, no. 11, pp. 198–206, Apr. 1981.
- [38] J. Misra, "Distributed discrete-event simulation," *Comp. Surveys*, vol. 18, no. 1, pp. 39–65, Mar. 1986.
- [39] R. M. Fujimoto, "Performance measurements of distributed simulation strategies," in *Proc. 1988 SCS Multiconf. on Distributed Simulation*, Feb. 1988, pp. 14–20.
- [40] Y.-B. Lin and E. D. Lazowska, "Exploiting lookahead in parallel simulation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 4, pp. 457–469, Oct. 1990.
- [41] D. M. Nicol, "Parallel discrete-event simulation of FCFS stochastic queueing networks," in *Proc. ACM SIGPLAN Symp. on Parallel Programming: Experience with Applications, Languages and Systems*, 1988, pp. 124–137.
- [42] D. B. Wagner and E. D. Lazowska, "Parallel simulation of queueing networks: Limitations and potentials," in *Proc. 1989 ACM SIGMETRICS and Performance '89 Conf.*, 1989, pp. 146–155.
- [43] S. S. Kuek and W. C. Wong, "Ordered dynamic channel assignment scheme with reassignment in highway microcells," *IEEE Trans. Veh. Technol.*, vol. 41, no. 3, pp. 271–277, 1992.
- [44] D. Jefferson, "Virtual time," *ACM Trans. Progr. Lang. Syst.*, vol. 7, no. 3, pp. 404–425, July 1985.
- [45] R. M. Fujimoto, "Optimistic approaches to parallel discrete event simulation," *Trans. Soc. Comp. Sim.*, vol. 7, no. 2, pp. 153–191, June 1990.
- [46] A. Gafni, "Rollback mechanisms for optimistic distributed simulation," in *Proc. 1988 SCS Multiconf. on Distributed Simulation*, Feb. 1988, pp. 61–67.
- [47] D. Jefferson, "Virtual time II: The cancelback protocol for storage management in time warp," in *Proc. 9th Ann. ACM Symp. on Principles of Distributed Computing*, Aug. 1990, pp. 75–90.
- [48] Y.-B. Lin, "Memory management algorithms for parallel simulation," *Information Sciences*, vol. 77, no. 1, pp. 119–140, 1994.
- [49] ———, "Determining the global progress of parallel simulation," *Inform. Proc. Lett.*, vol. 50, 1994.
- [50] B. Samadi, "Distributed simulation, algorithms and performance analysis," Ph.D. Dissertation, Dept. Computer Science, Univ. of California, Los Angeles, 1985.
- [51] F. Mattern, "Efficient distributed snapshots and global virtual time algorithms for non-FIFO systems," *J. Parallel Distrib. Comp.*, vol. 18, no. 4, pp. 423–434, 1993.
- [52] R. M. Fujimoto, "Parallel discrete event simulation: Will the field survive?," *ORSA J. Computing*, vol. 5, no. 3, 1993.
- [53] D. Arvind, R. Bagrodia, and Y.-B. Lin (Eds.), in *Proc. 8th Workshop on Parallel and Distributed Simulation*, ACM, 1994.
- [54] M. Bailey and Y.-B. Lin, Eds., in *Proc. 9th Workshop on Parallel and Distributed Simulation*, ACM, 1995.



Yi-Bing Lin received the B.S.E.E. degree from National Cheng Kung University in 1983, and the Ph.D. degree in computer science from the University of Washington, Seattle, in 1990.

Between 1990 and 1995, he was with the Applied Research Area, Bell Communications Research (Bellcore), Morristown, NJ. In 1995, he was appointed full professor, Department and Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan. His current research interests include design and

analysis of personal communications services network, distributed simulation, and performance modeling.

Dr. Lin is a subject area editor of the *Journal of Parallel and Distributed Computing*, an associate editor of the *International Journal in Computer Simulation*, an associate editor of *SIMULATION*, a member of the editorial board of *International Journal of Communications*, a member of the editorial board of *Computer Simulation Modeling and Analysis*, Program Co-Chair for the 8th Workshop on Distributed and Parallel Simulation, and General Chair for the 9th Workshop on Distributed and Parallel Simulation.



Paul A. Fishwick (M'87–SM'92) received the B.S. degree in mathematics from the Pennsylvania State University, University Park, the M.S. degree in applied science from the College of William and Mary, Williamsburg, VA, and the Ph.D. degree in computer and information science from the University of Pennsylvania, Philadelphia, in 1986.

He is an associate professor, Department of Computer and Information Sciences, University of Florida, Gainesville. He also has six years of industrial/government production and research

experience working at Newport News Shipbuilding and Dry Dock Co. (doing CAD/CAM parts definition research), and at NASA Langley Research Center (studying engineering data base models for structural engineering). His research interests are in computer simulation modeling and analysis methods for complex systems.

Dr. Fishwick is a senior member of the Society for Computer Simulation. He is also a member of the IEEE Systems, Man and Cybernetics Society, the ACM and AAAI. He founded the comp.simulation Internet news group (Simulation Digest) in 1987, which now serves over 15,000 subscribers. He was chairman of the IEEE Computer Society technical committee on simulation (TCSIM) in 1988 and 1990, and he is on the editorial boards of several journals including the *ACM Transactions on Modeling and Computer Simulation*, the *IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS*, *The Transactions of the Society for Computer Simulation*, the *International Journal of Computer Simulation*, and the *Journal of Systems Engineering*.

Improved Decision Making through Simulation Based Planning

Paul A. Fishwick, Gyooseok Kim, Jin Joo Lee
Computer and Information Science and Engineering Department
University of Florida

Abstract

Real-time military planning and decision making involves several different modeling techniques, including rule-based, operator-based and dynamic-model based approaches. While rule-based approaches are generally fast and are more appropriate for simple scenarios, simulation methods and dynamic models, indigenous to the simulation literature, are necessary to plan within environments involving large-scale uncertainty, multiple interacting elements and complex dynamics. Planning techniques must inter-operate to yield the best decisions, and we have found that simulation based planning serves as an architecture for detailed model levels for both real-time and off-line decision making. We introduce *Simulation Based Planning* as a methodology for addressing the complexity involved in Air Force missions while employing an example of *air interdiction*.

1 Introduction

Decision making and planning are critical operations for all military missions. Moreover, planning occurs over several different time scales depending on the amount of time that one has to plan prior to committing to a particular plan. Planning is a hierarchical enterprise since many techniques can be used to determine near-optimal plans. For example, if one has information on costs between events during a mission, and the goal is to minimize cost, then a mathematical programming approach, based on lowest cost path determination, may yield satisfactory results. An even higher level type of planning is possible by using heuristics in the form of operators and rules [Fikes 72]. Rules can use certainty factors or fuzzy sets.

Our long-term goal is to explore this hierarchy of planning approaches, and our first step toward this goal is to provide high level planners with a technique we call *Simulation Based Planning* (SBP). Military missions involve many interacting elements including concurrently active adversarial tasks and uncertain information regarding ground-based anti-aircraft capability. As the complexity of a mission and knowledge-base increases, it is valuable to use computer simulation and more detailed dynamic models to obtain an answer to the broad question “Which is the best approach to take given our mission and all currently available knowledge¹?” A good method for answering this question is to use simulation since the simulation technique is generally useful for obtaining answers to “what if” scenarios. The quality of the answer depends on how much time is available prior to committing to the plan. If more time is available, more simulation experiments can be executed in real time. If time is of the essence, higher level models will need to be simulated. In any event, our goal is to plan using the most detailed dynamic models available rather than to limit all planning to the use of a singular planning technique such as a decision tree.

¹Where the knowledge about the terrain, enemy unit motions and postulated enemy plans incorporate a great deal of uncertainty and can change over time.

Our contribution in the area of planning is to develop a method that allows simulation to be used in real-time, where the simulation is embedded within the decision making system. Consider a simulation system that manages force engagements up to the battalion level. CASTFOREM [CASTFOREM 96] provides one such modeling capability [Wargames 96]. CASTFOREM is driven by decision trees and rules to guide what actions occur at any given time. For interactive graphical output on the state of units, JANUS [Janus 96] can be used. Our goal is to allow a program such as CASTFOREM the ability to make decisions based, not on rules or decision tables, but on multiple simulations that are run *within the overall simulation*. Typically, simulation has been used widely in the military for offline decision analysis and “what if” weapons effectiveness assessment. Our proposal suggests that we enable simulation to support “course of action” (COA) analysis, and embed it directly within the force simulation. The approach yields a two-level simulation procedure: simulations for COA analysis guiding the decisions that drive the simulation of units, platoons, companies and battalions. This “simulation within a simulation” approach is novel, but it can be time consuming. For that reason, our work stresses the use of multi-level models so that different aggregation levels can be executed so that the planning can be performed with real-time constraints.

Planning, regardless of the specific domain, involves three components: 1) model type, 2) plan set, 3) plan evaluation. The first step in planning is to determine the modeling language (or type) to use. For rule-based approaches, this language can be “rules” or “predicate logic.” For other approaches, there are many alternatives: equation sets, finite state automata, Petri nets, functional block models, queuing models. Often the model type is visual in structure [Fishwick 95]. The next step is to create a set of candidate plans. For rules, this set is often created through backward chaining. For more detailed model types, the set is created by creating an experimental design and performing simulation (i.e., a type of forward chaining). Plan evaluation is the key step where elements in the plan set are simulated to determine the best plan(s). For our purposes, we view all modeling approaches as being definable hierarchically, so that model types can include both rules and detailed queuing models, for instance, defined at different abstraction levels. These hierarchical model types are termed *multimodels* [Fishwick 95].

We will first discuss the application of simulation-based planning in Section 2: air interdiction. Then, we define the method of simulation-based planning, and finally we illustrate our prototype simulation application which serves as an aid to plan interdiction missions. In Section 3, we focus on route planning using a low-level strike mission on a munitions factory. Information on the implementation is included in Section 4, followed by conclusions in Section 5.

2 Air Interdiction

A typical use of the application of force is air interdiction, where the purpose is to destroy, delay, or disrupt existing enemy surface forces while they are far from friendly surface forces [Drew 92]. The interdiction mission includes attacks against supplies and lines of communication. The objective of the interdiction mission is to reduce the enemy threat by diminishing enemy combat effectiveness or by preventing a buildup of combat capabilities.

To achieve this objective, careful and comprehensive planning is required to isolate an area and to stop all support from reaching the theatre of conflict. One must systematically attack the significant elements of the enemy's logistical structures (transportation lines and centers, supply depots and storage facilities, repair and modification centers, staging areas, and industrial installations) and maintain a high degree of destruction until the desired effect is achieved.

There are two levels of the interdiction plan : the interdiction plan at the theatre level and at the tactical air force level [AFM 1-7 54]. A theatre interdiction plan establishes the general scheme of employment, and enumerates available forces by type and number. The plan also outlines logistical support, delineates force responsibility, establishes the general system of targets, prescribes the priority of target systems, and describes the anticipated results. A theatre-level plan is developed through tactical air force planning. This planning involves the day-to-day conduct of operations for the implementation of the assigned portion of the broad theatre interdiction task. It covers specific and detailed actions of the forces to be employed. A large part of the mission is dependent on which particular route or air corridor is used.

The task of the attack aircraft is to strike the target swiftly and accurately with whatever munitions are carried, and then to return safely to base. To carry out this task, we must penetrate the enemy defense. Most difficulties arise here because methods of penetrating enemy defenses can vary according to the strength and sophistication of the hostile detection, reporting, command and control network, and how much intelligence is available on the nature of the defense capability [Spick 87]. Mission planning will also involve maintaining a balance between fuel and munitions resources to determine the load to be carried. Considering these constraints, selecting the best routes can be a complex undertaking.

2.1 Simulation Based Planning (SBP)

SBP refers to the use of computer simulation [Law 91, Fishwick 95] to aid in the decision making process. In much the same way that adversarial trees are employed for determining the best course of action in board games, SBP uses the same basic iterative approach where a model of an action is executed to determine the efficiency of an input or control decision within the given situation. However, board game trees implement a *static* position evaluation function whereas, in SBP, a model (serving as a *dynamic* evaluation structure) is executed to determine the effects of making a "move." With the ability to simulate models at different abstraction levels, SBP executes detailed models of a physical phenomenon when there is sufficient planning time, or when fast computation and parallel methods are instrumented.

The military has been using simulation-based planning for many decades in the form of *constructive model* simulation. A constructive model is one based on equations of attrition and, possibly, square or hexagon-tiled maps using discrete jumps in both space and time. To decide whether to accept a course of action, one can use a constructive model (a "wargame") to evaluate the alternatives. Related work by Czigler et al. [Czigler 94] demonstrates the usefulness of simulation as a decision tool. Our extension in SBP is one where we permit many levels of abstraction for a model, not just the aggregate abstraction level characterized by Lanchester equations and combat result tables. The idea is to allow the planner the flexibility to balance the need between the required level of detail and the amount of time

given to make a decision.

Although the planning system shown in Figure 1 is divided into 5 functional blocks, we will describe the overall framework in terms of three components: the experimental design component, the output analysis component and the simulation component. Experimental design is a method of choosing which configurations (parameter values) to simulate so that the desired information can be acquired with the minimal amount of simulation [Law 91]. Since we treat uncertainty in the planning domain as random variates based on probability distributions, repeated simulations (i.e., replications) using sampled data are necessary in order to perform the proper analysis, which includes confidence intervals about the mean for each result. We apply both heuristic and standard experimental methods to reduce the overall simulation time in two aspects: 1) in reducing the number of replications, and 2) in reducing the overall computation time spent on a single simulation of a plan on a particular route.

Following the experimental design, we simulate models of individual objects. This component is called Trial (block 2) as shown in Figure 1. Figure 2 displays the lower level model of the Trial block where each entity of the planning domain is modeled individually using the appropriate model type. We assume that there are seven types of physical objects: *BlueAC*, *RedAC*, *Radar*, *SAM*, *Wx*, *Tgt*, *Zones* and one abstract object called *Eval*. The class *BlueAC* stands for Blue Aircraft and *RedAC* for Red Aircraft. *Radar* represents a ground radar site. *SAM* represents a ground Surface-to-Air Missile site. *Wx* represents the weather. *Tgt* represents the target that needs to be destroyed. In the current prototype, the target is the red force munitions factory. *Zones* represent the area of defense zone. For the zones, we assume that there are a set of radars strategically located inside the zone such that when an enemy plane *BlueAC* flies inside the zone, it is detected.

During a typical simulation loop, every object updates its local state and perform actions, which in turn may affect other objects in the following time slice. The last object to be called within a simulation loop is *Eval*. *Eval* is responsible for three functions:

1. Maintaining a consistent “current state” of the world that is an aggregated state of all the current states of each object.
2. Deciding the outcome of inter-object events such as an engagement event between a *BlueAC* and a *RedAC*. By allowing either of the objects involved decide the outcome of an event—which would normally be beyond their control—we would violate symmetry and self containment among objects.
3. Evaluating each situation (from the planner’s point of view) for every time slice and maintaining a score which represents the goodness of the plan.

We now perform output analysis using the set of output data produced from the replications using the following blocks: Replicator (block 1), Evaluator (block 3) and Analyzer (block 4). Output analysis is concerned with obtaining the appropriate interpretations of the output data. The Replicator controls the random number streams for each replication. Different random number streams are used for each run, so that the results are independent across runs. We also allow for *common random numbers* (CRN) to provide a controlled environment for comparison among alternatives. This is to eliminate any “environmental

differences” that can exist between different simulations. CRN is a standard variance reduction technique in simulation and we use it across different alternative route plans within the same replication so that we may expedite convergence to the true placement of the mean.

In its simplest form, the Evaluator serves as the accumulator of any relevant simulation data that is produced from the Trial Block. If the objective function within the Trial block produces a set of scores for each alternative, a straightforward evaluation approach is to total the scores produced from the replications for each alternatives. Using the accumulated data produced by the Evaluator block, the Analyzer block calculates the mean, variance and the confidence interval for each alternative. The mean of the replication results serves as the basic “data” point for the response surface representing the goodness of a plan. Variance can be a measure of predictability or stability when the variance is small. Confidence intervals are useful because given a sample output distribution and a confidence level x , the interval states that, within $x\%$ confidence, the true mean lies within the stated interval [Lee 96].

2.2 An Air Interdiction Scenario

As one of the applications of SBP, we have chosen a typical air interdiction scenario, and developed its Simulation Based Planner (C++) and graphical user interface (Tk/Tcl) within our Multimodeling Object-Oriented Simulation Environment (MOOSE) initiative. To illustrate the usefulness of the SBP approach, we consider the air interdiction scenario depicted in Figure 3. Figure 3 defines a scenario with dynamically moving objects. The mission of the blue force aircraft is to destroy a red force munitions factory. There are three Radars ($R1$, $R2$, $R3$) and two Surface-to-Air Missile (SAM) sites ($S1$, $S2$), each with different effective detection ranges. Two red force fighters ($A1$, $A2$) are located in air defense Zone2 and Zone3 respectively, while one red force fighter ($A3$) is located outside of the air defense zones. At first glance, the problem of guiding the blue force around the radar, SAM and air defense zone coverage, and toward the factory seems like a simple problem in computational geometry. The geometry approach is used frequently for route planning problems. A typical rule might be formed as follows “To locate a path, avoid radar and SAM fields, and avoid fighting against enemy fighters.” The problem with this simple approach to route planning is that the reasoning becomes difficult when uncertainty and dynamics are present. This complexity manifests itself as an increasingly large rule base which often proves difficult to create, maintain and verify for consistency.

To illustrate the kind of uncertainty and dynamics which are involved, consider the following available information at some point during the mission.

- Uncertain location and range : Radar $R1$ and $R2$ have been identified as permanent fixtures, but a land based scout report suggests that $R3$ may have mobility. Moreover, the ranges (track, missile, arm range) of SAM site $S1$ is well known, but $S2$ has been reported to have a better guidance system including swift mobility, improving its surveillance capability.
- Uncertain enemy mission : red force fighter $A1$ and $A2$ are known to be on a Combat Air Patrol (CAP) mission, since they are always detected around Zone2 and Zone3; however, $A3$'s mission type is unknown.

In these examples, the behavior of each object is simplified as much as possible, since our purpose is to demonstrate how to handle uncertainty in SBP, and not to focus solely on the complex behaviors of objects. However, we have a plan to include the sophisticated behaviors of each object incrementally. This can be considered as an advantage of the SBP approach: the ability to increase the level of detail of the simulation object model as desired.

3 Route Planning Examples and Results

Figure 4 shows two possible routes (*Route1*, *Route2*) under the environment defined in Figure 3. The goal of blue force aircraft is to destroy the red force munitions factory while satisfying 3 constraints: time or fuel level, safety, and destruction of the target. Given the possible routes, the role of SBP is to choose the best route minimizing time and fuel consumption, and maximizing safety and target destruction. In Figure 4, *Route1* is more attractive than *Route2* if we value mission time above all others, but seems less safe since it is vulnerable to an attack by red fighter *A1*. *Route2* might be considered more safe and achieve higher target destruction than *Route1* by avoiding the attack from fighter *A1* and SAM site *S1*. However, it will be detected by radar *R2*, increasing the probability of losing blue force aircraft or damage to blue force aircraft. Moreover, there is a big chance of being detected by radar *R3* even though its location is uncertain. The table at the lower left of Figure 4 shows the result of the SBP. We display the mean score and the confidence interval half width of each mean at a 90% confidence level. As can be expected, *Route2* is more successful since it avoids direct attacks from the highly destructive enemy fighter and the SAM site (mean score of *Route2*: 69, mean score of *Route1*: -54).

If we delete *Route1* and consider another route based on the result of the previous situation, we may have two routes we want to analyze. Figure 5 illustrates these two candidates. *Route3* was chosen to avoid direct attack from *A1*, but for a short time period it will be detected by *R1*. *Route3* also takes the blue force into the track range of *S1*, but not into its arm or missile range. Being detected in the track range of *S1* does not seem very dangerous since only tracking functions may be performed by *S1*. We can expect its success to depend largely on the result of the samplings for uncertainty factors: specifically, the location and guidance capability of SAM *S2* and the mission type of *A3*. If the powerful guided system of SAM is sampled close to this route, or *A3* has a intercept capability, then the chance of success will be very small. Otherwise, the chance of mission success will be very good. These nondeterministic and stochastic characteristics can be resolved by multiple simulation with varying values for the uncertainty factors. The confidence interval of the mean score of *Route3* is wide in comparison to that of *Route2* due to the reason previously discussed; however, the overall mean score is better than that of *Route2* because of the small chance of being detected by *S2* or intercepted by *A3*.

We can now delete *Route2* and insert a route, *Route4*, which is carefully chosen to minimize the amount of time that a blue force aircraft will be within the detection ranges of *R2* and *R3* as in Figure 6. The result of the SBP shows almost the same mean score for *Route3* and *Route4* (*Route3* : 110.36, *Route4* : 103.08) with *Route3* being slightly better². But

²The goal is to maximize the mean score for determining the better plan.

we can select *Route4* as the best overall route based on its more narrow confidence interval (*Route4* : 1.3, *Route3* : 6.0).

4 Implementation

In this section, we briefly introduce an example of the multimodel which we developed for the Air Force Route Simulation and two analysis methods in the multiple simulations for dealing with uncertainties, which arise from Simulation Based Planning. Additional implementation issues and their potential solutions can be found in [Lee 96].

The purpose of the multimodel is to create a heterogeneous collection of connected sub-models so that one can simulate different parts of the system at different abstraction levels. The choice of a dynamic selection of abstraction level provides flexibility to the simulation based planning activity; real-time constraints can be met by tuning the multimodel. To implement the multimodel, the generic Route Simulation Model in Figure 1 was instantiated to the Air Force Mission Route Simulator, and each object resides inside the Trial block as in Figure 2. Among the seven types of physical objects, we chose one object, *BlueAC*, to explain how we could capture the object's multimodel behavior. The model presented here is not complete since it has not been validated by a Subject Matter Expert (SME). However, the model represents the kind of model that one could obtain from the SME through knowledge acquisition methods. Since building sophisticated and realistic models is not the issue in our current research, simple yet sensible models were built to prove our SBP approach.

The toplevel model of the *BlueAC* object is shown in Figure 7. It is modeled as an FSA with three phases: *Approach Target*, *Return to Base* and *End Mission*. Figure 8 shows the refinement FSA for *Approach Target* phase in Figure 7. Going another level down from the *Traverse Route* phase, Figure 9 illustrates the functional block model for updating the location while traversing the route. Figure 10 illustrates the refinement of *RedAC Alert Mode* in Figure 8.

Assuming that a set of alternate routes and environment data are given through the GUI, dynamic models are simulated and evaluated for each route. The simulation process is replicated and its output results are accumulated and then analyzed by the Analyzer (ref. Figure 1).

For the object, we categorized the uncertainty into several types.

- uncertainty of existence: the object may or may not exist.
- uncertainty of location: an area of uncertainty of the object's location is available but the exact location of the object is uncertain.
- uncertainty of range: the exact detection range or firing range is not known.
- uncertainty of mission: the exact mission type of an object is unknown.
- uncertainty of fire power: the destruction capability of the object is uncertain.

These nondeterministic and stochastic characteristics were resolved by multiple simulations using different samplings of the uncertainty factors. The planning problem becomes

one in optimization for an objective function representing the cost of traversing a route. This cost is currently a function of elapsed time, remaining strength of the unit and the level of success regarding achieving the goal. To reduce the total number of replications in the simulation, we used two different output analysis methods: *iterative* and *non-iterative*. The *iterative* method attempts to quantify significant pairwise differences among the alternatives' means within a given confidence interval. The method is referred to as "*iterative*" because the algorithm *iterates* performing for every iteration, a set number of replications and analyzing data to see if there are any significant differences among each route. Whenever a route is found that is significantly worse than all other routes, this route is then eliminated. The iteration continues until only two routes remain and a difference exists between the two of them.

The *Non-iterative* method is a method that avoids making an unnecessary number of replications to resolve what may be an unimportant difference. When two alternatives are close, we may not care if we erroneously choose one system (the one that may be slightly worse) over the other (the one that is slightly better). Thus, given a correct selection probability P and the indifference amount D , the method calculates how many more replications are necessary to make a selection with the probability of at least P , the expected score of the selected alternative will be no smaller than by a margin of D . In our experiment, we have chosen $P = 0.95$ and $D = 13$. A smaller D will produce more accurate results, but with many more replications.

Recently, we have begun construction of a system, called MOOSE, to enable users to interactively specify multimodels through a modeling window. Output is viewed via a scenario window, similar to those shown in Figures 3-6. MOOSE (Multimodeling Object-Oriented Simulation Environment) represents an implementation for a simulation system that is under construction, and based on an extension to object oriented design (<http://www.cis.ufl.edu/~fishwick/tr/tr96-026.html>). MOOSE is the next generation of SimPack (<http://www.cis.ufl.edu/~fishwick/simpack/simpack.html>), which was initiated in 1990 for providing a general purpose toolkit of C and C++ libraries for discrete-event and continuous simulation.

5 Conclusions

We have discussed the method of simulation-based planning within the confines of an air interdiction example. Our view is not that SBP replaces other forms of planning, but that this new approach can be used in conjunction with existing, higher level planning approaches. This way, given a set of alternatives to consider, SBP is able to extend the planning horizon in three aspects: probabilistic uncertainty is handled through detailed and replicated simulation of models rather than solving them analytically using probability theory; it extends the level of reasoning to a finer level of granularity, producing plans that are closer to the level of execution and discovering subtleties that may be missed by a higher level planner; and finally, it breaks down the complexity of multiagent adversarial planning by employing object-oriented multimodel simulation.

Once the simulation results have been produced, the data can be analyzed and interpreted in several ways to choose the "best" plan. For instance, we can choose the plan which has

not only a good mean score but also the minimum confidence interval width to ensure that it is the safest plan possible. We may also decide to choose a plan that has the most number of highest scores even though the confidence interval width may be large in order to select a plan that has the best potential in spite of risks involved. We can even decide to choose a plan at random (given that the scores are above some threshold) which will produce nondeterministic planning. This is particularly useful for mission planning—opposing forces should not be able to predict one's plan. In addition, similar to how simulation is used for visualization, simulation can be easily used to perform visual playback of how a plan was simulated to explain the planner's decision. This can be very useful for the military since much of the military training is done through *after action review*.

Prior to the advent of fast low-cost personal computers, few researchers would consider simulation of a fairly extensive experimental design to be a possible candidate for real-time mission planning. However, as the speed of low-cost computers increases, the simulation-based planning technique presents itself in a more attractive light. Our longer range goal is to explicitly link several plan model levels together so that, for instance, a rule-based level can be identified from a lower-level simulation. One of the authors (Kim) is studying effective consistency measures which will rectify differences in rules produced empirically (through knowledge acquisition) and rules generated automatically from multiple low-level simulations.

6 Acknowledgments

We would like to acknowledge Karen Alguire, Project Manager, at Rome Laboratory, and sponsorship under United States Air Force grant F30602-95-1-0031.

7 Biographies

Paul A. Fishwick is an Associate Professor in the Department of Computer and Information Science and Engineering at the University of Florida. He received the BS in Mathematics from the Pennsylvania State University, MS in Applied Science from the College of William and Mary, and PhD in Computer and Information Science from the University of Pennsylvania in 1986. He also has six years of industrial/government production and research experience working at Newport News Shipbuilding and Dry Dock Co. (doing CAD/CAM parts definition research) and at NASA Langley Research Center (studying engineering data base models for structural engineering). His research interests are in computer simulation modeling and analysis methods for complex systems. He is a senior member of the IEEE and the Society for Computer Simulation. He is also a member of the IEEE Society for Systems, Man and Cybernetics, ACM and AAAI. Dr. Fishwick founded the `comp.simulation` Internet news group (Simulation Digest) in 1987, which now serves over 15,000 subscribers. He was chairman of the IEEE Computer Society technical committee on simulation (TCSIM) for two years (1988-1990) and he is on the editorial boards of several journals including the *ACM Transactions on Modeling and Computer Simulation*, *IEEE Transactions on Systems, Man and Cybernetics*, *The Transactions of the Society for Computer Simulation*, *Interna-*

tional Journal of Computer Simulation, and the *Journal of Systems Engineering*. Dr. Fishwick's WWW home page is <http://www.cise.ufl.edu/~fishwick> and his E-mail address is fishwick@cise.ufl.edu.

Gyooseok Kim received the B.S. degree in Electronics from Korean Air Force Academy in 1984 and the M.S. degree in Computer Science from Korean National Defense College in 1989. He served in the Korean Air Force as a chief programmer of the Korean Air Defense System. He is currently pursuing a doctoral degree in the Computer and Information Science and Engineering department at the University of Florida. His research interests are focused on knowledge acquisition and validation within qualitative and quantitative simulation. Mr. Kim's E-mail address is kgs@cise.ufl.edu.

Jin Joo Lee received the B.S. degree in Computer Science from Ewha Womans University, Korea in 1988 and the M.S. degree in Computer Science from Brown University in 1991. After receiving the M.S. degree, she was a research engineer at Human Computers Inc., Korea until 1992. She received a PhD degree in the Computer and Information Science and Engineering department at the University of Florida in 1996. Her research interests are in AI planning, simulation and control. Ms. Lee's WWW home page is <http://www.cise.ufl.edu/~jl1> and her E-mail address is jl1@cis.ufl.edu.

References

- [AFM 1-7 54] Department of the Air Force. *Air force Manual No. 1-7 : Theatre Air Forces in Counter Air, Interdiction And Close Air Support Operations*. Department of the Air Force, 1954.
- [CASTFOREM 96] CASTFOREM, Web Reference: <http://hp01.arc.iquest.com/mosaic/060.html>
- [Czigler 94] M. Czigler, S. Downes-Martin and D. Panagos. Fast Futures Contingency Simulation: A "What If" Tool for Exploring Alternative Plans, In Proceedings of the 1994 SCS Simulation MultiConference, San Diego, CA, 1994.
- [Drew 92] D. M. Drew. *Air Force Manual 1-1: Basic Aerospace Doctrine of the United States Air Force*, Volume II, Department of the Air Force, 1992.
- [Fikes 72] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and Executing Generalized Robot Plans, *Artificial Intelligence*,3,1972.
- [Fishwick 95] Paul A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds*, Prentice Hall, 1995.
- [Janus 96] JANUS, Web Reference: <http://hp01.arc.iquest.com/war/janus.html>
- [Law 91] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis* McGraw-Hill, 1991

- [Lee 93] J.J. Lee, W.D. Norris and P.A. Fishwick. An Object-Oriented Multimodeling Design for Integrating Simulation and Planning Tasks, In *Journal of Systems Engineering*, 3, 220-235, 1993.
- [Lee 94] J.J. Lee and P.A. Fishwick. Real-Time Simulation-Based Planning for Computer Generated Force Simulation, *Simulation*, 299-315, 1994.
- [Lee 95] J.J. Lee. and P. A. Fishwick *Simulation-Based Real-Time Decision Making for Route Planning*. In Proceedings of the 1995 Winter Simulation Conference, 1087-1095, 1995
- [Lee 96] J.J. Lee. *A Simulation-Based Approach for Decision Making and Route Planning*. PhD Thesis, University of Florida, 1996.
- [Russell 95] S.J. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*, Prentice-Hall, 1995.
- [Schoppers 87] M. Schoppers. Universal Plans for Reactive Robots in Unpredictable Domains, In *Int. Joint Conference on Artificial Intelligence*, 1987.
- [Spick 87] M. Spick. *An Illustrated Guide to Modern Attack Aircraft*. An Arco Military Book, Prentice Hall Press, New York, 1987
- [Salisbury 93] M. Salisbury and H. Tallis. Automated Planning and Replanning for Battle-field Simulation, In Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, 1993, 243-254, Orlando, FL.
- [Wargames 96] Wargame Catalog, Web Reference: <http://hp01.arc.iquest.com/war/war.html>

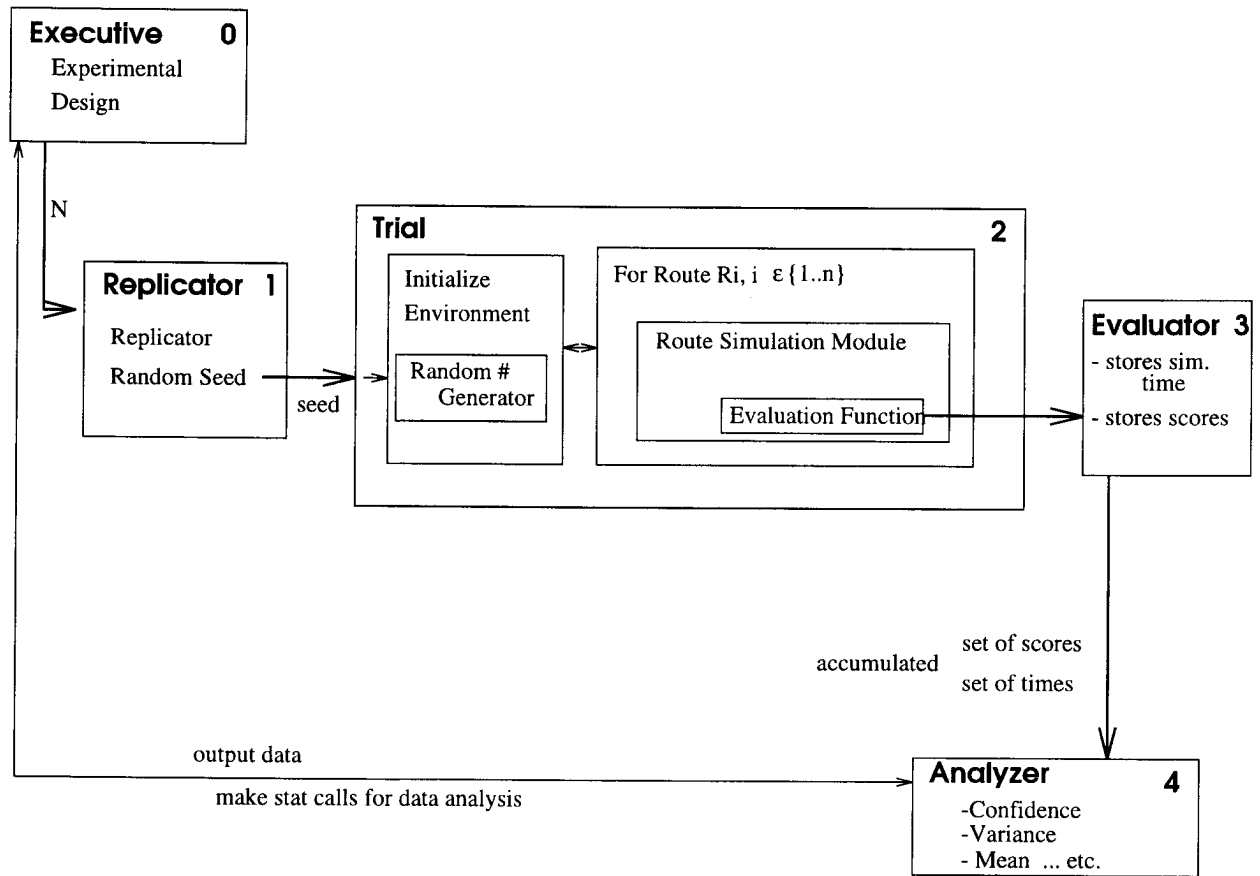


Figure 1: Generic Top Level Architecture of a Route Planner

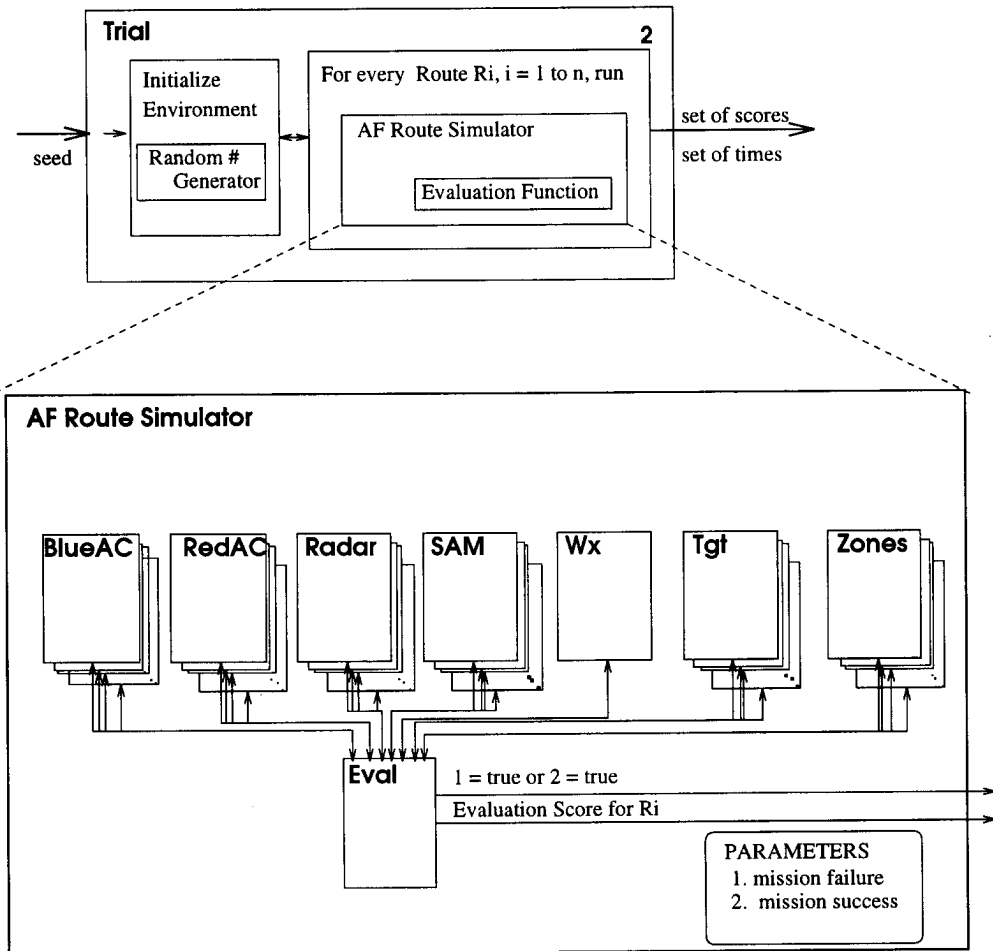


Figure 2: General Simulator Module

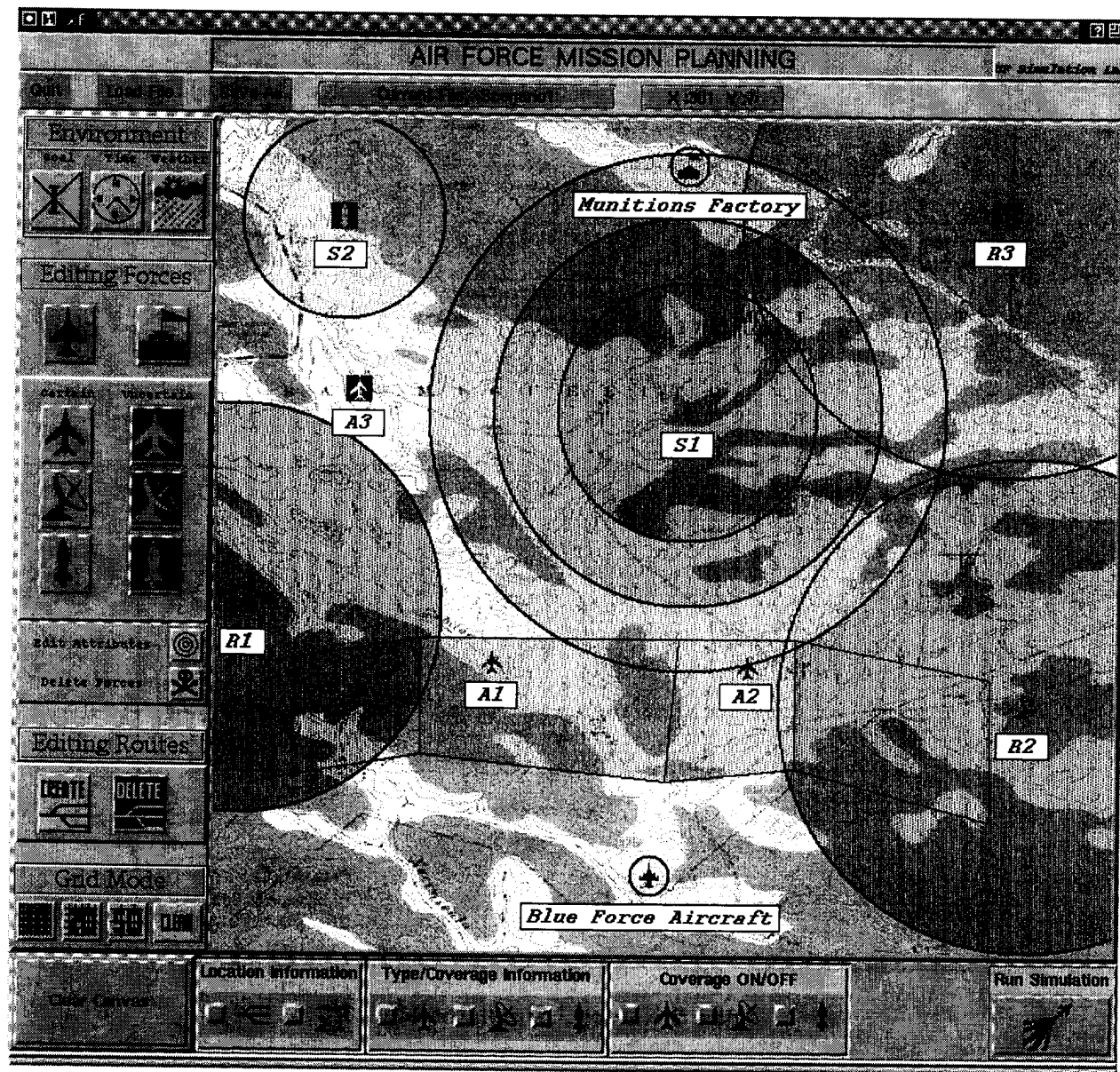


Figure 3: A Typical Air Interdiction Scenario

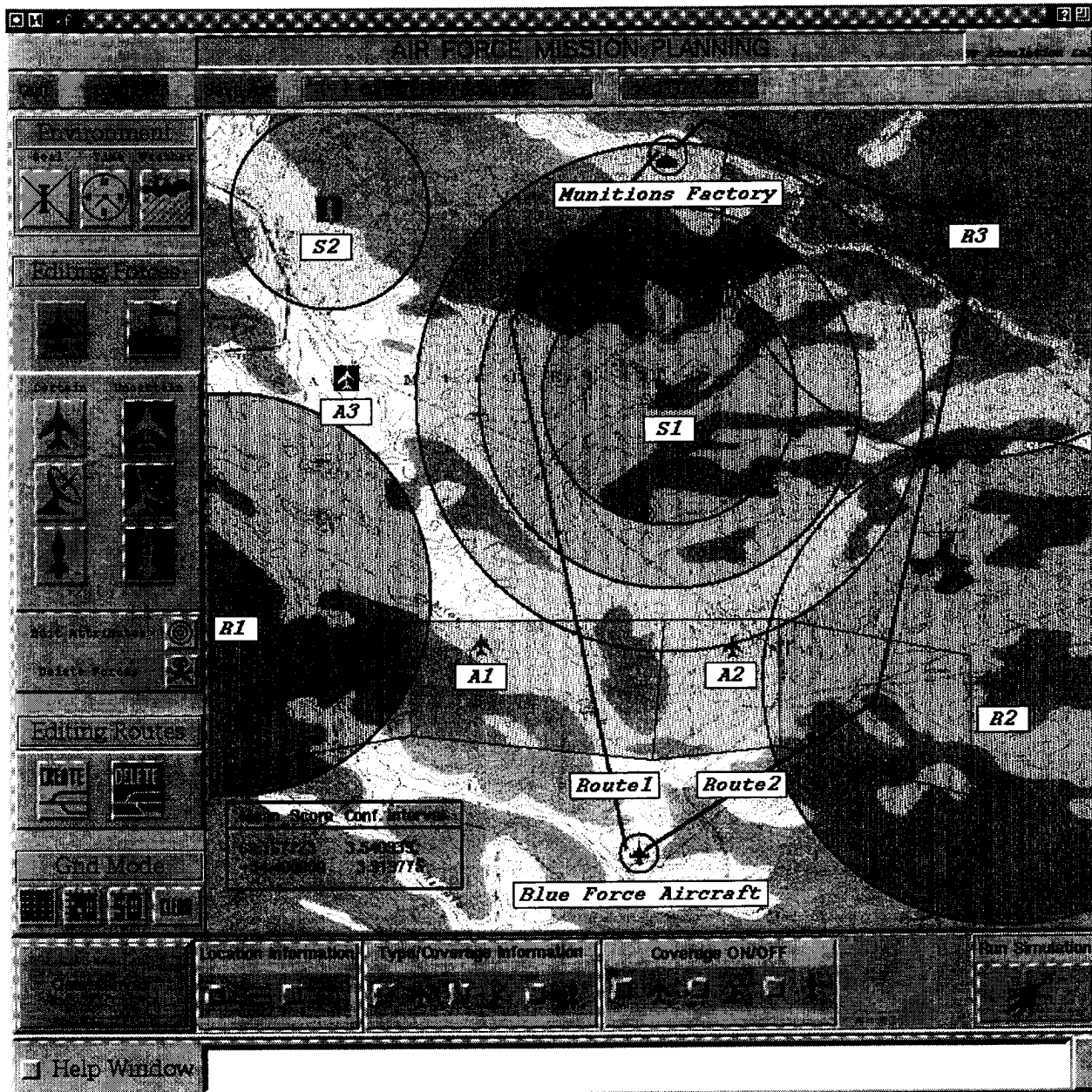


Figure 4: Two Possible Routes in the Figure 3

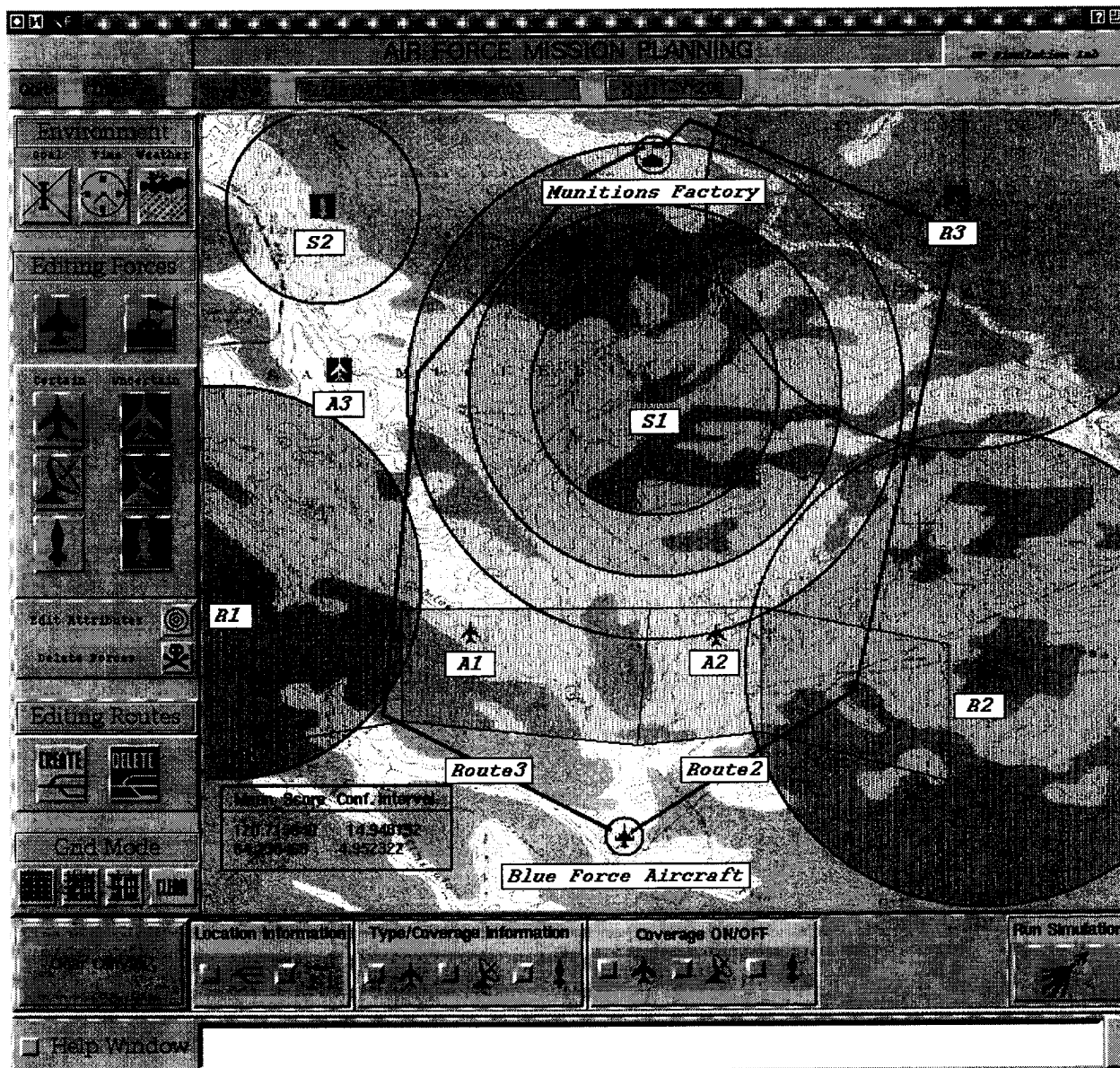


Figure 5: Deleting Route1 in the Figure 4, and Inserting Route3

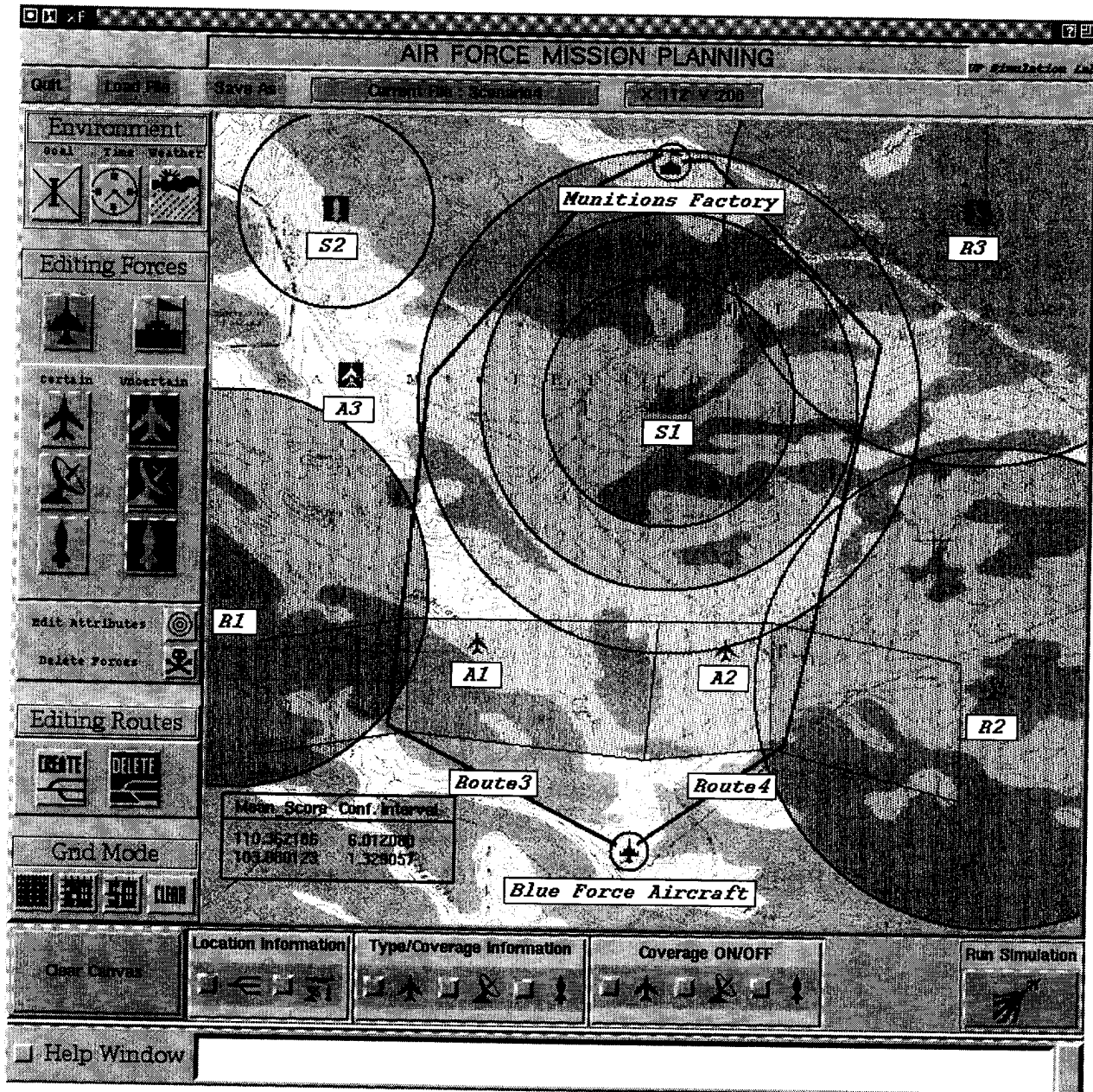


Figure 6: Deleting Route2 in the Figure 5, and Inserting Route4

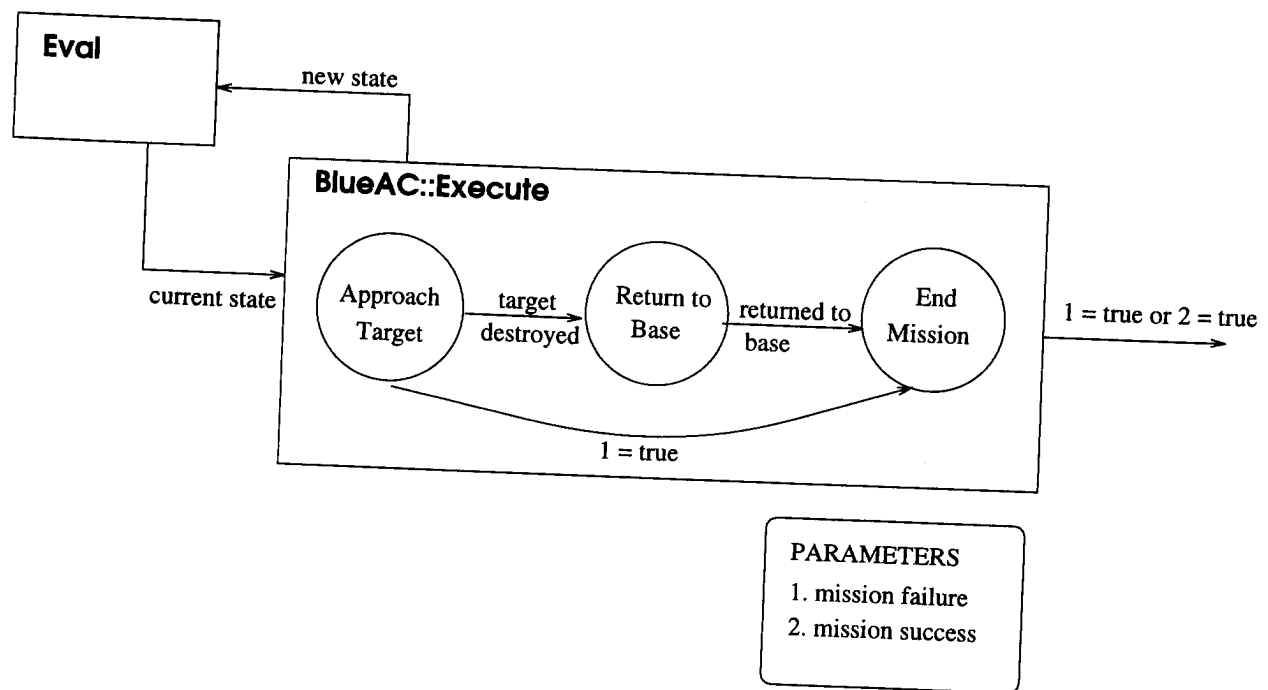


Figure 7: Blue Aircraft (*BlueAC*) object model

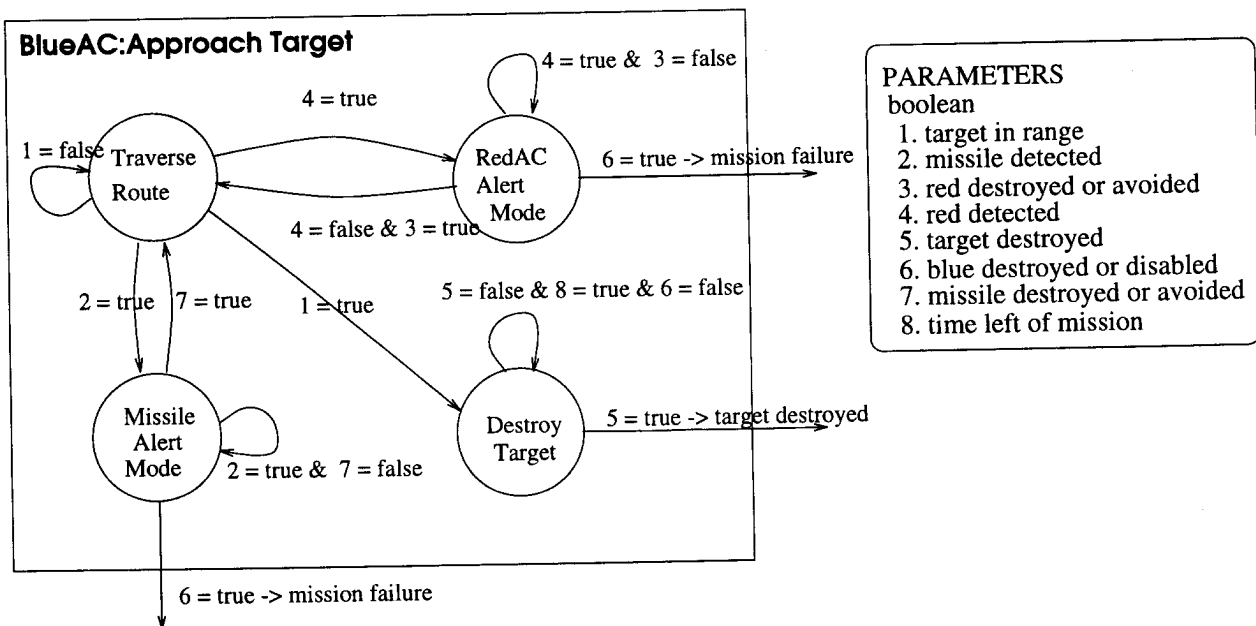


Figure 8: Approach Target for *BlueAC*

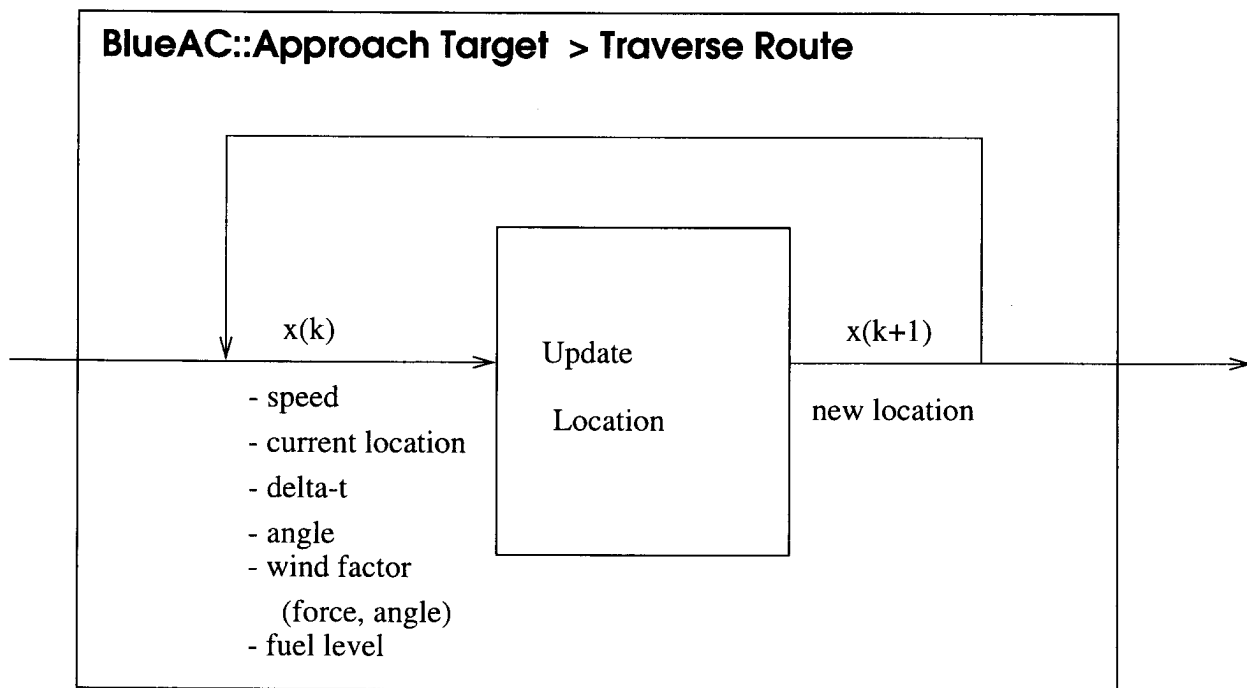


Figure 9: Traverse Route Function for *BlueAC::Approach Target*

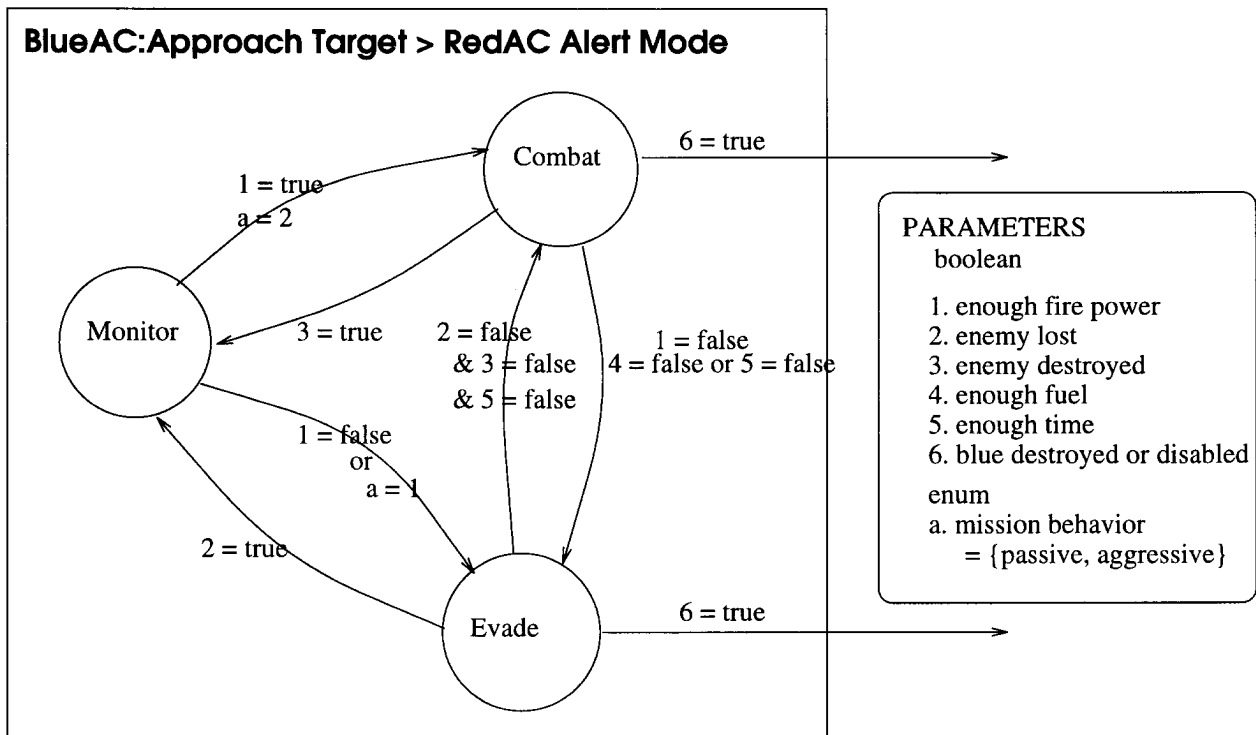


Figure 10: Red Aircraft(*RedAC*) Alert Mode for *BlueAC* object

A SIMULATION-BASED APPROACH FOR
DECISION MAKING AND ROUTE PLANNING

By

JIN JOO LEE

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1996

Di

To God

And to my parents, Bum Suk Lee and Chung Sook Lee

ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for leading me every step of the way. I am very grateful to my advisor, Dr. Paul A. Fishwick, for his guidance and constant encouragement throughout my years at UF. I would also like to thank other members of my supervisory committee, Dr. Sherman X. Bai, Dr. Douglas D. Dankel, Dr. Timothy A. Davis, and Dr. Li-Min Fu, for their kind understanding and willingness to serve on my committee.

I must thank my dear husband, Dongwook, for his love and for always believing in me even at times when I did not. I thank my darling daughter Jungah(Josi), for being a healthy and a happy child. Without them, these past few years would have been quite barren. My deepest respect and gratitude go to my parents for their wisdom and their enormous love. Everything that I am today, it is because of them. I thank my wonderful sisters and brother for their constant support and loving concerns.

My special thanks to Sooha Lee, Hyesun Lee, Kangsun Lee and Inhee Chung for their friendship. I would like to thank the following colleagues of the Simulation Group: Gyooseok Kim for his time in developing the GUIs for my applications and Robert M. Cubert for his work with the Blocks language for multimodeling.

I would like to mention Ms. Jung-Hee Choi and Ms. Ji-Soo Choi for their loving help with Jungah. It is because of people like them that a mother like me can pursue such a dream.

Finally, I want to acknowledge the Institute for Simulation and Training, Science Applications International Corporation (SAIC) and Rome Laboratory, Griffiss Air Force Base for funding most of the research done in this dissertation.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	D iii
ABSTRACT	D vi
CHAPTERS	D 1
1 INTRODUCTION	D 1
1.1 Problem Statement	D 2
1.2 Background and Related Work	D 4
1.2.1 Simulation and AI Planning	D 4
1.2.2 Intelligent Control	D 7
1.2.3 Decision Science and Game Theory	D 7
1.2.4 Scheduling	D 7
1.2.5 Object-Oriented Multimodeling Methodology	D 9
1.3 Contribution to Knowledge	D 14
1.4 Outline	D 15
2 METHODOLOGY	D 17
2.1 Simulation-Based Planning	D 17
2.2 The SBP framework	D 18
2.2.1 Simulation block : Trial	D 19
2.2.2 Experimental Design block : Executive	D 23
2.2.3 Output Analysis blocks : Replicator, Evaluator, Analyzer	D 33
2.3 Rule-based Systems and SBP	D 38
3 A MULTIMODEL DESIGN FOR DECISION MAKING	D 45
3.1 A Truck Depot Example	D 45
3.2 Intelligent Objects	D 49
3.2.1 Exception Control	D 50
3.2.2 Mixture Control	D 52
3.2.3 Optimal Height Control	D 53
3.3 Non-Intelligent Objects	D 54
3.3.1 Model Design	D 54
3.3.2 Model Execution	D 57
4 ADVERSARIAL ROUTE PLANNING	D 58
4.1 Mission Planning for Computer Generated Forces	D 58
4.2 Planner Architecture	D 59

4.2.1	World Database (DB)	D 59
4.2.2	Reactive Behavior	D 60
4.2.3	Planning Behavior	D 61
4.2.4	Expert System	D 66
4.3	Interface between Planner and other Command Entities	D 66
4.4	Interface between Terrain Analyzer and Planner	D 68
4.4.1	Terrain Analyzer	D 69
4.5	Demonstration Mission Scenario	D 70
4.6	Planning Results	D 74
5	NON-ADVERSARIAL ROUTE PLANNING	D 79
5.1	Rover Route Planning	D 79
5.2	Simulation-Based Rover Route Planning System	D 82
5.2.1	Planner	D 82
5.2.2	Experimental Design	D 88
5.3	Prototype Environment	D 91
5.4	Planning Results	D 92
6	NONDETERMINISTIC ADVERSARIAL ROUTE PLANNING	D 97
6.1	Air Force Mission Planning	D 97
6.2	Planner Architecture	D 99
6.2.1	Multimodel of AF Mission Route Simulator	D 99
6.3	Demonstration Mission Type	D 111
6.4	An Air Interdiction Scenario	D 113
6.5	Planning Results	D 115
7	CONCLUSIONS	D 126
8	FUTURE WORK	D 129
	REFERENCES	D 131
	BIOGRAPHICAL SKETCH	D 134

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

A SIMULATION-BASED APPROACH FOR
DECISION MAKING AND ROUTE PLANNING

By

JIN JOO LEE

August, 1996

Chairman: Dr. Paul A. Fishwick
Major Department: Computer and Information Science and Engineering

Decision making is an active area of research in simulation, systems engineering and artificial intelligence. One subset area of decision making, automated route planning, is covered in this work with our approach based on the technique of simulation rather than on purely heuristic or geometric techniques. This new technique is called simulation-based planning (SBP). Simulation-based planning is useful for route planning under various conditions including uncertain locations and events with potential adversarial activity. We propose that it is only by using simulation that one can make the most effective plan in uncertain and complex environments.

SBP extends the planning area mainly in three aspects. First, probabilistic uncertainty is handled through detailed and replicated simulation of models rather than solving them analytically, for example, using probability theory. Second, simulation models naturally extend the level of reasoning to greater detail, often involving continuous state space. Thus, SBP is able to produce plans that are closer to the level

of execution. Additionally, one can often discover subtleties that may be missed by higher level planners which are often rule-based. Third, the complexity of multi-agent adversarial planning breaks down when object-oriented multimodel simulation is used. Here, each agent or adversary is individually modeled and simulated in response to each plan. In addition, to ensure that SBP can be used within reasonable time constraints, we develop general experimental design algorithms and techniques which reduce the overall simulation time.

CHAPTER 1 INTRODUCTION

Planning is indispensable to our everyday lives. We plan for everything, whether we work, relax or sleep. Having a plan—a good plan—greatly affects the successful outcome of events and this is why humans spend time to plan, hoping that planning ahead will increase the probability of success. Planning is a fundamental aspect of human intelligence and therefore it has become a fundamental problem in modeling intelligent behavior within Artificial Intelligence. Through time, planning domains have evolved from the typical “toy-world” domains such as the blocks world to “real-world” domains such as mission planning in the military. The earlier classical planning methods are no longer appropriate for these real-world planning problems which can contain uncertainties and real-time constraints in decision making. A major problem with these uncertain domains is that it is hard to predict the outcome of events, which is essential to planning.

Our approach to solving the problem is Simulation-Based Planning (SBP)—a method that incorporates simulation models into the planning process. By building simulation models for individual objects and simulating them to gather the combined effects at the required level of detail, we are able to plan for more complex, adversarial environments more readily and effectively. In addition, SBP’s ability to reason at a level of finer granularity can bridge the gap between the two planning areas: 1) the classical AI planning domain which represents the coarser level of planning, dealing mainly with symbolic and abstract actions; and 2) the intelligent control planning domain representing the lowest level of planning, dealing with actual optimal execution of the given plan.

1.1 Problem Statement

The general problem of planning in AI is commonly identified with problems that are highly conceptual where individual actions are of the form “Go To Supermarket” and “Buy milk” [34]. A plan is an ordered set of such high-level actions. Here, the concern is not how one will physically (at a detailed level) get to a supermarket, but rather it is on the ordered set of actions whose logical effects will satisfy the goal. STRIPS [13, 14] is a classical example of such an approach to planning. This type of approach is reasonable if the execution of the produced plan is not the responsibility of the planner. Difficulties arise when execution becomes part of the planning system, and planning problems take place in an environment over which the planner has no control, such as another agent or an enemy, and when there is uncertainty of available information or uncertainty of another agents’ reaction. In such cases, accurate prediction of the resulting states of plan execution is extremely difficult.

Past reasons for using rules in planners centered around the idea that rules are easy to build, less expensive to execute, and that they adequately reflect the heuristics of the human decision maker (e.g. the company commander). However, because thousands of rules are involved just to model one agent or entity, the complexity of maintaining and reasoning about plans is not easy, especially since rule-based systems are usually centralized. Also, real-time constraints are often hard to meet with the rule-based approach because there is no well-defined way to optimize the reasoning process. As will be discussed in section 2.3, rules alone are inadequate—due mainly to their symbolic nature—in handling planning areas which require reasoning at finer level of detail often involving continuous systems. In addition to trying to model human decision making, it is just as important (if not more) to create planners which yield the best plans or decisions that are likely to succeed even in the

presence of uncertainty and complexity. In actuality, some combination of 1) capturing human decision making heuristics and 2) creating automated planners capable of near-optimal decisions (based on objective functions) is the ideal solution, and it is toward this solution that our work is directed.

In this work, we define a simulation-based planning (SBP) methodology for producing near-optimal decisions and route plans. Since the route planning area of research lies in between the higher level of symbolic AI planning and the lower level of intelligent control, it is an ideal candidate for SBP, where the use of simulation enables reasoning under uncertainty at a finer level of granularity. Route planning can largely be categorized into two types: 1) a set of alternative routes are given and the goal is to select the best route with the best plan of execution, and 2) no alternative routes are given and the goal is to produce a near-optimal traversal strategy in the current environment. In the first type, it is assumed that the set of alternative routes are produced by a higher-level symbolic planner or a human expert. In the second type, selecting an alternative route is no longer an issue but rather selecting the appropriate traversal strategy (e.g. various parameter settings, behaviors) for route traversal is the main concern.

In terms of specific application areas, we have chosen mission planning within the military domain as our main area since it always involves some form of route planning of the first type. In the military, routes greatly affect the success of the whole mission, whether the mission takes place on ground or in the air. We have also applied the SBP method to solve the second type of route planning for the Mars Rover. Finally, as an application of SBP for on-line decision making, we present a simulation model based decision system for controlling a truck depot.

1.2 Background and Related Work

At a fundamental level, general AI planning, decision making, intelligent control and route planning in robotics, all strive to solve a common problem—based on some model of a given process, determining what actions will affect the process in a desired way. At a glance, the problem appears to be different because each area uses different types of models dealing with different levels of abstraction and applications. To view the problem of decision making and route planning in a larger context, we have studied different areas that are relevant to planning. First, we present some background insights about the relationship between simulation and AI planning and then discuss related work.

1.2.1 Simulation and AI Planning

In the simulation literature, simulation is defined as “the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output”; Fishwick [17, page 1]. In the planning literature, Dean and Wellman [10] state that the idea of using a model to formulate sequences of actions is central to planning and, given a sequence of actions, a robot can use the model to simulate the future as it would occur if the actions were carried out. Simulation can provide the robot with information that can be used to suggest modifications or to compare the proposed sequence with an alternative sequence. And we believe humans also have models built and stored in their brain for most objects or systems that exist in the world and these models are used to formulate sequences of actions that would occur in the future if a plan was executed. Especially for problems where the complexity of finding the solution is too complex for humans, such as find a path through a maze, humans tend to use trial and error method to eventually find the correct path. Once simulation models have been built

for individual entities of the world, simulation can be used as a tool to provide the planning system with information useful for evaluating its hypothesis. Therefore, it is logical that we employ simulation within the planning process to simulate the uncertain results of a proposed plan before the plan is selected for execution.

To overcome this increase in complexity of reasoning, many new approaches have been introduced. Schoppers's universal plans generates a reaction to every possible situation that can occur during plan execution [37]. Salisbury and Tallis's automated adversarial planner [35] is a hierarchical, backtracking planner capable of performing automated planning, execution monitoring, and replanning. This planner accepts a mission with its related information such as task organization from the division plan and searches the problem space to find a sequence of actions for the battalion. The plans produced by these methods are robust but can be very large and expensive to execute. Also, replanning is often too slow to be useful in time-critical domains.

From a system theoretic point of view, latest work by Dean et al. [9] focuses on a method based on the theory of Markov decision processes for efficient planning in stochastic domains. This approach is closely related to our work in that the world is modeled as a set of states and actions having a transition model that maps state and action pairs into discrete probability distributions over the set of states. However, it differs considerably in several aspects. First, probability is handled analytically which means the outcome is probabilistic but deterministic. In SBP, it is nondeterministic since data is either sampled from a distribution or produced by a more detailed simulation model. Second, the value or reward of each state has to be predetermined with a probability. With SBP, the value of the entire path is calculated at the end of each simulation based on an objective function. Another way to describe this difference is in terms of the objective function: our method uses a dynamic evaluation function based on simulation data while the other uses a static evaluation function

based on probability and static values. Another difference is the type of domain it can handle. As Dean et al. [9] state, the method is reasonable only in a benign environment while SBP can handle malicious environments involving adversarial agents. In Wellman [41, 42], the problem of route planning is treated as an extension of the well known shortest-path problem. He points out that for most AI route planning domains, the cost of a route is the sum of the cost of its segments and the cost of each edge in a route are assumed to be known with certainty. Standard search algorithms such as best-first and A* are based on these assumptions and optimality is no longer guaranteed when the route costs are probabilistic and dependent. Wellman proposes a generalized dynamic-programming approach based on stochastic dominance which solves these problems. However, again this approach does not consider problems where the state transition itself is nondeterministic.

Uncertainty

In planning, there are several different types of uncertainty [32]. Here, we discuss two categories that are most relevant to our work.

Data Uncertainty - There may be uncertainty in the knowledge of the environment such as the location or existence of the enemy. Uncertainty due to noise of the sensors or other systems also belong in this category.

Randomness - Some domains are inherently random. Even when the available knowledge is complete and certain, stochastic properties exist. For example, it is never possible to perfectly predict the reaction of another agent in response to a given stimuli. Although we may be able to make a good guess, we can never be completely certain and thus, a form of randomness exists. Another good example is uncertainty due to the randomness of the outcome of events. Predicting the outcome of such events as engagement between two entities in the battlefield is difficult due to its random nature.

1.2.2 Intelligent Control

Intelligent control deals with problems that are more physical and less conceptual. The problem of steering a cargo ship to a desired heading [1] or planning a path for a robot are typical problems in intelligent control. Even though the whole task can be simply stated as “steer the ship to heading x ,” at the control level, if we are to build a more practical system, we should be also concerned with tuning the control input to physically steer the ship to a precise heading. Thus, planning and control should be integrated in order to provide a more complete and reliable system. Dean [10] provides a good overview of the various problems and techniques available in these two areas.

1.2.3 Decision Science and Game Theory

Decision science involves the creation of decisions based on a game-theoretic foundation. Given the current “state of the world,” one can embark upon several courses of action (decisions) each of which will yield a payoff or utility [27]. Games can be naturally extended to continuous systems [3] (often found in simulation) by equating the input (or forcing) function to a continuously changing *decision* which alters the payoff given the corresponding state changes.

1.2.4 Scheduling

Scheduling is a decision-making process that is concerned with the allocation of limited resources to tasks over time. Optimization of one or more objectives is its goal. Scheduling problems are very difficult even for situations where the requirements and resource characteristics are completely deterministic [19]. Rogers [31] divides the approaches to scheduling into three groups: 1) pure non-simulation-based approaches, 2) pure simulation-based approaches, and 3) hybrid approaches.

Pure non-simulation-based approaches are again divided into two alternative approaches to schedule generation: the first uses Operations Research (O.R.) tools and the second is based on AI concepts. Space limits further discussion of the O.R. approaches and the reader can consult Dempster et al. [11] for more information. AI approaches have already been discussed in previous sections so they too will not be discussed here.

Pure simulation-based approaches employ the following steps whenever a decision needs to be made—typically in a manufacturing system environment [31].

1. The user initializes the simulation model with the current state of the environment such as released orders and production resources. The planning and order requirements are also given which comes from higher level information systems.
2. The user identifies the set of actions which is applicable at this decision point as candidates for evaluation.
3. The user makes one simulation run (because it is deterministic) for each of the candidates. The results are stored for later analysis.
4. After all options have been evaluated, the one with the best performance is selected and implemented.

Finally, hybrid approaches emerged due to deficiencies of a pure simulation-based approach to dynamic rescheduling. One of the problems is that human activities such as candidate action selection, performance evaluation, and schedule choice are difficult, causing significant time delays in the overall system. Wu and Wysk [43] employ discrete simulation to evaluate a set of sound dispatching rules (in the domain of flexible manufacturing systems). The rule with the best simulated performance in the time period is then applied to the physical system. This is very similar to our

work since it is using simulation to evaluate and choose the best alternative for the current situation. However, a major difference, other than the fact that the domain is totally different, is that these simulations are deterministic. As pointed out [31], there are questions as to the validity of such simulations for systems where there is known stochasticity.

1.2.5 Object-Oriented Multimodeling Methodology

The object oriented approach to simulation is discussed in different literature camps. Within computer simulation, the system entity structure (SES) [46] (an extension of DEVS [30]) defines a way of organizing models within an inheritance hierarchy. In SES, models are refined into individual blocks that contain external and internal transition functions. Within the object oriented design literature [33, 4], the effort is very similar in that object oriented simulation is accomplished by building 1) a class model and 2) dynamic models for each object containing state information. Harel [20, 21] defines useful visual modeling methods in the form of “state charts” so that the dynamics may be seen in the form of finite state machines.

Models that are composed of other models, in a network or graph, are called multimodels [15, 16, 17, 18]. Multimodels allow the modeling of large scale systems at varying levels of *abstraction*. They combine the expressive power of several well known modeling types such as FSAs, Petri nets, block models, differential equations, and queuing models. By using well known models and the principle of orthogonality, we avoid creating a *new* modeling system with a unique syntax. In the original multimodeling concept, when the model is being executed at the highest level of abstraction, the lowest level (representing the most refined parts of the model) is also being executed. Each high level state duration is calculated by executing the refined levels that have a finer granularity.

To optimize execution time of these models, we are making an extension to the multimodeling concept. By using concepts and methods from aggregation, we plan to build levels of aggregated models so that when time is a limiting factor, we can choose to execute at higher levels of abstraction.

The following provides an excellent starting point when deciding how to organize information and build models of dynamical systems from a simulationist point of view [17]:

1. Start with a concept model or the class hierarchy of the system. This phase should involve creating all relationships among classes. Classes can be related in multiple ways but we consider only two key relations, both of which take advantage of hierarchy and the sifting of information either up a tree (aggregation) or down a tree (inheritance). Generalization or inheritance is where attributes and methods are copied to lower level classes. Aggregation is the reverse situation where structural information is passed from the leaf nodes to the root node.
2. Create a class model using a visual approach such as OMT [33]. A class is a set of similar objects. The structure of a class involves two items: attributes and methods. Attributes are static and generalize to capture static models, whereas methods are dynamic and generalize to capture dynamic models. Examples of static models are data models (as in the database systems literature) and semantic networks (in the AI literature). Examples of dynamic models are found in the simulation literature (e.g Petri nets, automata, block models, systems dynamics graphs, bond graphs and equational models). The base type of dynamic models are declarative, functional and constraint. Both static and dynamic models tend to have graphical configurations, but there are exceptions: static model–first-order logic relations, dynamic model– differential equations.

- (a) Static model structures can change over time. They are called static since the model structure does not encode dynamics or a change of state. Instead, a static model encodes structural information much like the attribute `Max_Speed` has a value which can change over time but `Max_Speed` has no coding information, and so cannot “change itself.” Therefore, “static” refers to a lack of inherent dynamics or code, and not to an inability to change.
 - (b) The term “model” reflects physical objects. Non-physical objects and classes, such as the data structure `Linked_List`, are normally not associated with “model” unless a specific real-world metaphor is applied where, for instance, a `Stack` is represented as a physical stack of objects. Therefore, the term “model” is relevant to real-world phenomena.
3. Finally, Construct a *multimodel* to build a network of models each of which defines a part of the overall system. In the usual object oriented approach, phase three translates to creating methods for an object that alter the state of that object. The problem is that phase three can be quite complex depending on the scale of the system being modeled. There needs to be a way of developing multi-level models that specify the phase three dynamics. Our approach is to use multimodels for this purpose.

In addition, within the confines of an example, we provide some guidelines for representing knowledge as models. Let’s consider a physical 2D space called *S*. *S* can be partitioned into cells using several different techniques, such as quadtrees and meshes. Space *S* has attributes in each cell. For example, an attribute might be “water level,” “concentration” or some other physical characteristic of *S*. There are two objects which can roam around on space *S*: *A* and *B*.

1. We create an object S which is an aggregate of, say, 4 cells. Each of these cells has 4 sub-cells. A quadtree model, showing the layout of S, is stored as an attribute of S.
2. Since S is a type of space, then we can form a generalization hierarchy where S inherits attributes from a higher level class called SPACE. We can also form an aggregation hierarchy for S so that all space attributes of the sub-cells are aggregated (passed upward) until we reach S as the topmost aggregation node. The mechanics of where items are physically stored in memory are left to the implementation. Even though S logically contains the entire aggregate space in an array, the implementation may choose to keep the information stored either all at the top node (S) or in the leaf nodes.
3. Let's suppose that there exists one A and one B in a particular cell, and further suppose that the dynamic relation of A to B remains fixed. For example, A is (always) pushing B. Then a dynamic model such as a functional block model is defined within object S or another sub-object which contains both A and B. With a functional model, there will be a function 'a' defined in object A and a function 'b' in object B. The functional model ' $a \rightarrow b$ ' is a model within S. Therefore, dynamic models contain coupling information for functions which lie underneath it within the aggregation hierarchy.
4. Consider that we have eight A objects ($A_1, A_2, \dots, A_8$) in a cell and that they are configured in a circle shape. The dynamics for all As are defined so that the two nearest neighbors of any object operate as if springs connected them together (i.e. a circle of objects with springs connecting each object to the nearest 2 neighbors). There will be a constraint model in S which contains

terms referencing the attributes located in each A_i . The constraint model is useful when there is not one direction.

5. Suppose that A and B move around independently of one another, but within the confines of one sub-cell. Then, there will be a function or dynamic model in each of A and B. There is no higher level “coupling” to form a dynamical model since the objects are independent. If we let A and B move around the entire space S then, the aggregation hierarchy (i.e., the static quadtree model in S) will change over time. Again, this is the logical knowledge representation, whereas an actual program may decide to avoid this implementation overhead by storing all information about the sub-cells in S (at the top level) to avoid changing static models. Most of our route planning domains fall within this category. Many of the objects such as aircrafts, tanks, radars, robots and rocks move around independently of one another but within the confines of a certain planning area.
6. So far we have avoided any dynamic model over the “field” S. We can create a dynamic model in S which defines how sub-cell attributes of concentration or level changes over time. A diffusion equation may serve as this model. There is the question of how the diffusion equation (describing dynamics over the field S) interacts with the other models for the movements of A and B. The interaction of the field and object dynamic models takes place as a part of the model’s structure. For example, when calculating the field at any given time, we might need to know an attribute of A or B. This attribute is referenced directly in the diffusion equation model in S. The same is true for the effect that the diffusion equation has on the models for A and B.

7. As a general rule, a dynamic model involving various functions and state variables should be located in an object which is an aggregate of all objects containing those functions and state variables. This is why we put the diffusion model in S. We could, however, have stored the previously specified functional model $a \rightarrow b$ within a sub-cell which always contains A and B, and is contained within S. It need not appear in S unless A and B are allowed to roam throughout S during the simulation, and not only within a sub-partition of S.

1.3 Contribution to Knowledge

The main contribution of this research is the introduction of a new methodology: Simulation-Based Planning. SBP extends and improves the planning horizon in three aspects. First, it handles probabilistic uncertainty through detailed and replicated simulation of models rather than solving them analytically using probability theory or using simple Monte Carlo sampling. Second, simulation models can naturally extend the level of reasoning to a finer level of granularity, often involving continuous state space. Thus, SBP is able to produce plans that are closer to the detailed level of execution and thereby often discovering subtleties (which would have been missed by a higher-level planner) that may lead to failure of a plan. Finally, multiagent adversarial planning is easily achieved through object-oriented multimodel simulation where each agent or adversary is individually modeled and simulated in response to each plan.

A common impression that people have about simulation is that it is time consuming. This is true when either there are many alternatives to consider or when many replications are necessary in order to gather the effects of uncertain factors. However, we show that simulation time can be reduced, allowing time constraints to be met, through the use of experimental design methods and multimodeling methods

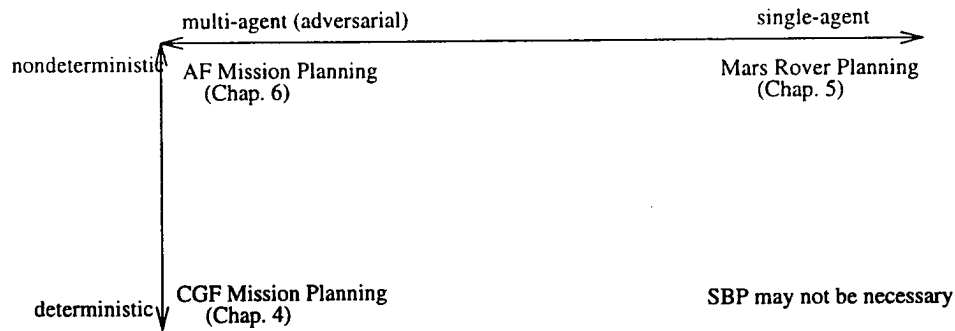


Figure 1.1. Classification of problems

in simulation. Some domain specific heuristics can also be useful in pruning out alternatives. Experimental design is a well established method in simulation which allows the optimization of the experimental process so that as much information as possible is obtained at the least cost. Thus, simulation experiments can be designed so that the plans are evaluated in the time available with a certain confidence level. Another way to control the simulation time (i.e. the planning time) is to decide at which level of abstraction the model will be simulated. The newly extended Multimodeling will allow the simulation models to be defined at different levels of abstraction so that any one of the levels can be chosen for execution. A high-level simulation can be done by sampling from a distribution, employing a closed-form analytic technique. A complex low-level simulation can be done by simulating the state change in greater detail at each time step with the use of differential equations or continuous-state control block models. In section 2.2.2, we provide a general framework for developing an experimental design block called the Executive model that will accomplish such a task.

1.4 Outline

Simulation-Based Planning as a general methodology is discussed in chapter 2. In chapter 3, an example multimodel design for decision making is presented using a truck depot problem.

Figure 1.1 classifies the three sample application domains we have selected in terms of two properties—whether the planning domain is multi-agent or not and whether it involves any nondeterminism (uncertainty of data). The mission planning domain described in chapter 4 represents a planning domain that is multiagent and adversarial but with no data uncertainty. There is uncertainty of enemy’s action but none in terms of available information or randomness of events. In chapter 5, we introduce the Mars Rover route planning application. This problem does not involve any adversarial agent but it does have significant amount of data uncertainty and randomness. The most complex problem of three, the mission planning in the air force in chapter 6 is multiagent, adversarial and uncertain. Route planning domains can also be classified based on whether or not different routes are given as part of the alternatives. The military mission applications in chapters 4 and 6 involve choosing a route from a set of alternative routes, whereas the rover planning application in chapter 5 does not. Finally, conclusions are presented in chapter 7 and future work is discussed in chapter 8.

CHAPTER 2 METHODOLOGY

2.1 Simulation-Based Planning

Simulation-Based Planning refers to the use of computer simulation to aid in the decision making process. In much the same way that adversarial trees are employed for determining the best course of action in board games, SBP uses the same basic iterative approach with the following items: 1) a model of an action is executed to determine the efficiency of an input or control decision, and 2) different models are employed at different abstraction levels depending on the amount of time remaining for the planner to produce decisions. In the first item, board game trees implement a static position evaluation function, whereas, in SBP, a model is executed to determine the effects of making a *move*. In the second item, SBP can run more detailed models of a physical phenomenon when there is sufficient planning time or fast computation or parallel methods are instrumented.

The military has been using simulation-based planning for many decades in the form of *constructive model simulation*. A constructive model is one based on equations of attrition and, possibly, square or hexagon-tiled maps using discrete jumps in both space and time. To decide whether to accept a course of action, one can use a constructive model (a “wargame”) to evaluate the alternatives. Related work by Czigler et al. [6] demonstrates the usefulness of simulation as a decision tool. Our extension in SBP is one where we permit many levels of abstraction for a model, not just the aggregate abstraction level characterized by Lanchester equations and combat result tables. The idea is to allow the planner the flexibility to balance the need between the required level of detail and the amount of time given to make a

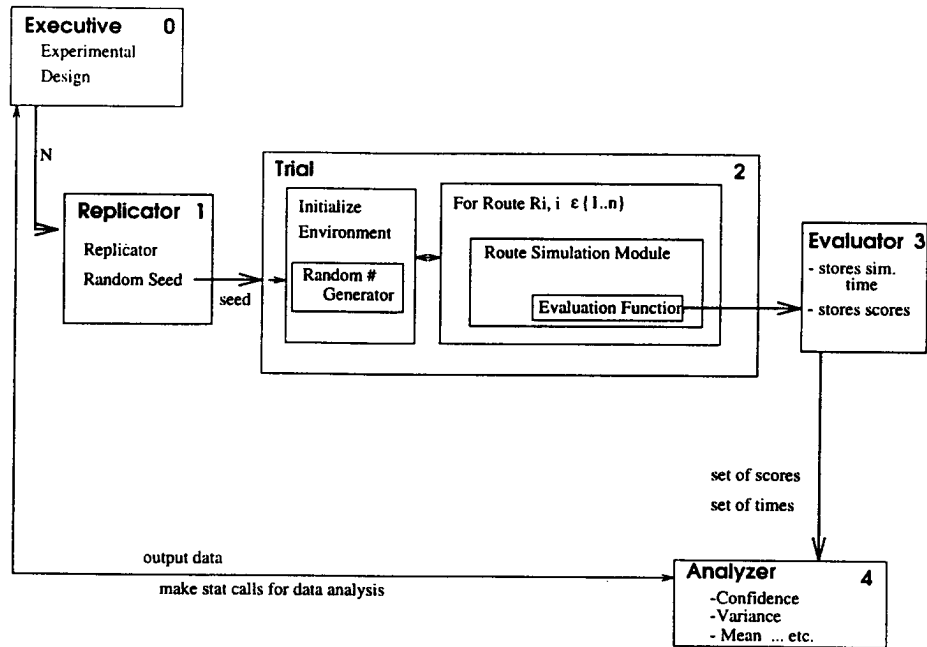


Figure 2.1. Generic Top Level Architecture of a Route Planner

decision. The notion that simulation can be used for decision making is covered in several disciplines, such as for discrete event based models [40].

We are not claiming that SBP is the method to use in all types of planning problems. For planning problems whose problem domain is well known, the outcome of each action certain and their interaction simple, SBP will not be able to exhibit its advantages. And for most cases, SBP should be used in conjunction with some type of higher level reasoning system so that the initial set of candidate plans are produced by this higher level system. Then for evaluating which is the near-optimal plan and for refining the plan to the execution level to ensure near-optimal results, SBP should be employed.

2.2 The SBP framework

We present the SBP framework in terms of three components: the simulation component, the experimental design component and the output analysis component.

The model in 2.1 is the top level general framework for simulation-based route planning systems. We discuss each of the three components and the individual blocks that belong to each component.

2.2.1 Simulation block : **Trial**

To use simulation in planning, we first need to identify the set of *controllable* and *uncontrollable* variables. Speeds, routes, actions of objects are controllable, whereas any kind of uncertainty such as uncertainty of weather conditions and outcome of combat are uncontrollable. The controllable variables or factors are sometimes called as *parameters* in the simulation literature [36]. The main objective of plan simulation is to gather the effects of the uncontrollable through repeated sampling (replication) while varying the parameters to find a near-optimal combination of controllable values in spite of the uncertainty. We say near-optimal because we can never guarantee the optimality of a plan given the uncertainties of actual plan execution. In addition to parameters, there are *noise factors* and *artificial factors*. Noise factors include sources of variation within the real-world system as well as exogenous factors such as customer and supplier characteristics. Artificial factors are simulation specific variables such as the initial state of the system, termination conditions and random number streams. We consider noise factors in chapter 5. Due to the nature of our problem, artificial factors such as initial state of the environment and termination conditions of plans are assumed to be given by the user. Other issues of artificial factors are discussed mostly in section 2.2.2. Here we present some definitions that will help us formally describe the simulation algorithm we have created.

Definitions

We model the environment $\mathcal{E}$ as a finite set of world states $\mathcal{Q}$ and a finite set of actions $\mathcal{A}$ where an action may be taken in every state or in a certain subset of states $\mathcal{Q}$. Let the environment $\mathcal{E}$ consist of a set of K objects $\mathcal{W} = \{W_1, W_2, \dots, W_K\}$ and

let their respective states be $q_i(t)$ for each W_i at time t . Note that we define an object to be an entity that is physically represented as an object in the real world, object such as a radar site, missile site and plane. These objects may be an aggregate of other objects. A Weather object, for example, may be represented as a single object outputting weather conditions at every delta time or as an aggregate of other objects such as clouds, wind, and sun.

Then we can define the world state at time t to be

$$\mathcal{Q}(t) = q_1(t) \times q_2(t) \times \cdots \times q_K(t)$$

Also, we define

$$\mathcal{A}(\sqcup) = (a_1(t), a_2(t), \dots, a_K(t))$$

where $a_i(t)$ represents a set of possible actions W_i can take at time t . Normally, a single action from the set $a_i(t)$ will be chosen to be simulated. But for objects whose models are moving continuously such as a fighter aircraft, a basic action such as UpdateLoc is taken for every time step and any additional action such as fire weapon may also be taken.

Unlike most plan evaluation schemes where the predicted state transition occurs by analytic (therefore, deterministic) and probabilistic functions, SBP nondeterministically chooses the “most likely to happen” actions given the situation. In other words, SBP does not simply choose the state with the highest probability like other existing methods which use stochastic dominance [42], but evaluates the transition by sampling the probability. In the deterministic methods, the state that has the highest probability of being the next state will *always* be the predicted next state, whereas, in SBP, that state will *most likely* be the next state but not necessarily. This is because, in SBP, the next state is determined either by probabilistic sampling or by executing a detailed model (which may eventually involve some type of sampling)

of the transition itself. Thus, both an advantage and possibly a drawback, SBP may come to evaluate actions which do not have the highest probability of occurring.

Since the choice to take action a in a given situation may be uncertain, we model the probability of this choice for W_i in the following example.

Suppose a_{i1}, a_{i2}, a_{i3} are three possible actions given the current state $Q(t)$ ¹. Then, based on the probability distribution or in our case, based on the object models' internal heuristic or logic, the choice is made.

Let us assume that the model we simulated chose a_{i2} . Let there be two possible state changes that can result from this action. We denote this change as $q_{i1}(t+1)$ and $q_{i2}(t+1)$. Then the transition function can be written as

$$\delta(Q(t), a_{i2}) = \{Q(t) \oplus q_{i1}(t+1), Q(t) \oplus q_{i2}(t+1)\} \quad (2.1)$$

where $\oplus$ denotes a change where a new state $q_i(t+1)$ is added and the old state $q_i(t)$ is deleted. Their respective transitional probability can be expressed as

$$Pr[\delta(Q(t), a_{i2}) = Q(t) \oplus q_{i1}(t+1)] = a$$

$$Pr[\delta(Q(t), a_{i2}) = Q(t) \oplus q_{i2}(t+1)] = b$$

If a detailed transition model exists, then it can be simulated to provide the exact transition given the current state. But if no such models exists or if time is limited, we can simply sample the probability given above to decide which will be the next state of q_i . Consequently, the next world state $Q(t+1)$ is obtained when all the objects have transitioned to their next states such that

$$Q(t+1) = q_1(t+1) \times q_2(t+1) \times \cdots \times q_K(t+1)$$

We now define the following:

¹In most cases, an object will make a decision based on a small subset of the world state and thus entire world state Q is usually not needed.

- Let R be a set of routes that need to be simulated and chosen from. The total number of routes is N and R_j denotes the j th route where $1 \leq j \leq N$.
- **Stationary Object** refers to an object that remains physically in the same location throughout the simulation and objects that do not have the ability to physically change location. Ground radars, missile sites and buildings are some examples. A stationary object may no longer exist when it is destroyed during the simulation. Let O be the set of all stationary objects where $|O| = M$.
- **Moving Object** refers to an object that has the ability to physically move and change its location during simulation (e.g. planes, AWACS and missiles). Let D be the set of all moving objects where $|D| = L$.
- **Planner Object** refers to the object which is the planning entity itself. We denote it by O^p .
- Let $\tilde{O}$ be the set of *uncertain* stationary objects. Let $\tilde{D}$ be the set of *uncertain* moving objects. Then, these objects can have one or more of the following uncertainties:
 - initial location (if associate total probability is less than 1 then the object's existence is also uncertain)
 - type (e.g. type of plane, type of missile)
 - configuration (e.g. power, speed, etc)
 - decision logic of the object

Then, following is our simulation algorithm:

1. Initialize environment:

For every replication P , ($1 \leq p \leq n$) setup all objects in $\tilde{O}$ and $\tilde{D}$ by sampling from their respective distributions using the random seed produced by the replicator.

2. For each route R_j ,

```

WHILE ( (not success) or (not failure) )
  Obtain current state  $Q(t)$ 
  FOR each object  $W_i$  in  $W$ 
    Simulate  $W_i$  to take action  $A_i$ 
    Update state  $q_i(t)$  to  $q_i(t+1)$ 
      by 1) detailed simulation of  $A_i$ 
      or
      2) sampling distribution
    Update world state  $Q(t+1)$ 
   $t = t + 1$ 

```

Simulation strategy is usually a mix of time slicing and event scheduling. Time slicing is used to routinely check each object for its responses to any change in the world state. Event scheduling is needed to allow objects to schedule any delayed response or action that is to occur in some future time which may not necessarily coincide with a particular time slice. Simulate until the termination criteria such as goal success or failure is met. The necessary output data of the simulation is now sent to the Evaluator block.

2.2.2 Experimental Design block : Executive

In Simulation, experimental design is a method of choosing which configurations (parameter values) to simulate so that the desired information can be acquired with the least amount of simulating [25]. In experimental design terminology, the input parameters and structural assumptions composing a model are called *factors* and the

output performance measures are called *responses*. The number of runs depends on three factors: the total number of factors, the number of factor levels and the number of replications (repeats). One way to reduce these numbers is by using heuristics. Drawing nomographs of all of the three factors to determine the dominant elements and to evaluate design tradeoffs is another method [38]. Fractional factorial design is also applicable [38, 25]. The variables are screened, based only on a fractional number of runs of the total combinations, for their effect on the response variable. An issue exists, however, whether we can use them effectively inside the planner as part of the system since these methods are mainly designed to be used by humans.

For domains where the response variable is continuous, response surface methods can be used. Chapter 5 illustrates such an example. However, for route planning problems that involves some type of discontinuous surface where factors such as alternative routes and alternative strategies exist, the standard simulation methods can not be applied.

In general, the total number of computer runs, S , required for a replicated, symmetrical experiment [38] is

$$S = p(q_1)(q_2) \dots (q_k) \quad (2.2)$$

where

p = number of replications

q_i = number of levels of i th factor, $i = 1, 2, \dots, k$

k = number of factors in the experiment.

For a typical route traversal simulation we can express the computational complexity of a single run for route R_j , in the worst case, as follows (given that we are using time slicing),

$$\phi(R_j) = O \left((Num_{so} + L)^2 \frac{dist_{R_j}}{\Delta dist} \right) \quad (2.3)$$

where

$dist_{R_j}$ = is the total length of route R_j

$\Delta dist$ = is the size of time slice $\Delta t * Speed(t)$

Num_{so} = is the number of stationary objects previously calculated

L = is the total number of moving objects in the simulation.

Thus, the total time complexity of the experimental part of the SBP algorithm will be, in the worst case,

$$O \left(\sum_{j=1}^N S \cdot \phi(R_j) \right). \quad (2.4)$$

In summary, we can save time in two ways: one by reducing S or by reducing $\phi(R_j)$ while still obtaining meaningful results. We first present various heuristics in reducing S :

- Reducing the number of factors:

For our problem domain, routes, behaviors and the planner's speed are some candidate factors. By setting any one of these factors to be a constant, we are effectively reducing the number of factors. Because, setting a factor to be a constant implies that $q_i = 1$.

- Reducing the levels of i th factor:

Reducing the number of levels of the i th factor from A to B reduces S by a factor of $A - B$. For instance, using some heuristics, we can prune away unpromising routes before they are simulated and thus resulting in the reduction of the number of levels of the route factor.

- Reducing the number of replications:

The number of replications depends on many factors. Most of all, it depends on the type of output analysis method used (i.e. what is the analysis criteria for obtaining the correct selection of the best alternative?). Some different output analysis methods we have employed are discussed in section 2.2.3. Another way to reduce the number of replications is *heuristic* sampling or *controlled* sampling of the uncertainties. This way, we can converge on the answer faster than sampling purely by random. The number of replications will partly depend on the randomness of the data. The wider the range of varied answers, the lower the confidence level will be and therefore, will need additional number of replications. Although in some cases where the input variables create an *unstable* environment, the additional number of replications will not make much difference in reducing the interval width. In effect, the number of uncertain objects in an alternative greatly affects the number of replications needed to reach an acceptable level of accuracy.

Next, we discuss some ways to reduce $\phi(R_j)$, the total simulation time spent during the evaluation process. In time critical situations, the quality of the desired information may be sacrificed to meet a given time constraint.

- Increasing the size of the time slice Δt (assuming the simulation is based on time slicing). Speed up will result at the expense of accuracy since the longer the time slice, the faster the simulation will be.
- Reducing number of stationary (Num_{so}) to be simulated for route R_j . Although the actual number of objects involved in the planning problem can not be controlled, we attempt to reduce this set to a minimal size while still obtaining meaningful data. Acquiring this minimum set of objects to be simulated can be done using the following structures:

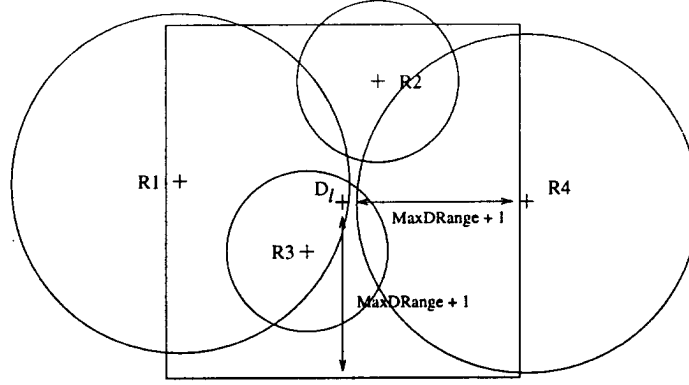


Figure 2.2. Dynamic PlanSim Window

- Dynamic PlanSim Window
- Static PlanSim Window
- $SD_{(t,t)}$

Dynamic PlanSim Window is a rectangular area that includes all stationary objects that may be affected by or effect the moving object. Every moving object has a Dynamic Plansim Window and for every moving object D_l , we will denote its Dynamic PlanSim window as DW_l . It is expected that area covered by DW_l will change over time. Stated more formally, let $D_l(x, y, t)$ denote the fact that D_l 's location is x, y at time t . Let $MaxDRange$ be the maximum detection range (or range of interaction) of all stationary and moving objects. Then the DW_l is defined by the rectangular area whose left top point is $(x - MaxDRange - \alpha, y + MaxDRange + \alpha)$ and right bottom point is $(x + MaxDRange + \alpha, y - MaxDRange - \alpha)$. Figure 2.2 shows an example window with four radar sites R1, R2, R3 and R4. Then, DW_l will contain R1, R2 and R3 but not R4 since the center location of R4 is not inside the window. As shown, DW_l can include stationary objects such as R2 which will not actually be affected by D_l since it is out of R2's range. This occurs when the distance between D_l and the center of a stationary object O is greater than the

radius of O . By checking the distance between D_l and all stationary objects, we can build a more accurate set of DW_l . However, this should only be done if the number of stationary objects are very large and the efficiency gained exceeds the overhead of performing this additional test.

Static PlanSim Window is a rectangular area that remains unchanged throughout one simulation run. If the planning problem does not involve alternative routes, then the static plansim window will remain unchanged for all replications since it will essentially include all the objects of the environment. If alternate routes are involved, we create a window for each route R_j which we will denote as SPW_{R_j} . SPW_{R_j} includes the route and all stationary objects that may affect the planner object. Let route R_j consist of m points $(x_k, y_k), 1 \leq k \leq m$ and line segments which connect them. We then compute the window using the following procedure.

1. Compute $\text{MinX} = \min\{x | x_k, 1 \leq k \leq m\}$ and $\text{MaxX} = \max\{x | x_k, 1 \leq k \leq m\}$. Compute MinY and MaxY in the same way. Now the bounding box of route R_j can be defined by these points where the left top corner is $(\text{MinX}, \text{MaxY})$ and the right bottom corner is $(\text{MaxX}, \text{MinY})$.
2. Extend the boundaries of this box by enlarging it by MaxDRange calculated above. Thus, the Static PlanSim Window's left top corner now becomes $(\text{MinX} - \text{MaxDRange}, \text{MaxY} + \text{MaxDRange})$ and right bottom becomes $(\text{MaxX} + \text{MaxDRange}, \text{MinY} - \text{MaxDRange})$.
3. Create the set SPW_{R_j} for each route R_j by finding all stationary objects who belong inside this box. Figure 2.3 shows a typical Static PlanSim Window.

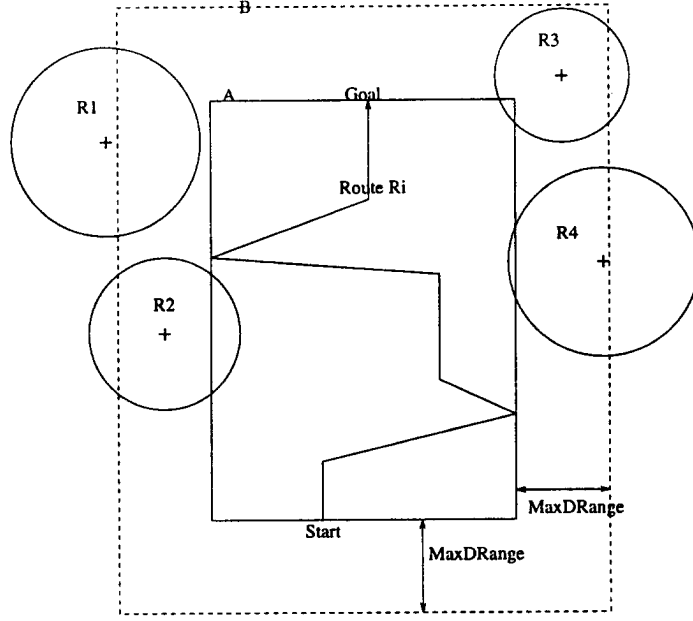


Figure 2.3. Static PlanSim Window for Path i

$\mathbf{SD}_{(l,t)}$ is the set of stationary objects such that for a single simulation run, $SD_{(l,t)} \subset \{O \cup \tilde{O}\}$ and is within the PlanSim window of a moving object D_l at time t . The degree that the performance improves by updating the PlanSim Window and the set SD_l for every δt will vary depending on the total number of stationary objects.

- Number of moving objects that need to be simulated is the entire set D whose size is L .
- If there is no previous simulation history to base it on, we must obtain the total number of stationary objects from the Static PlanSim Window for R_j as described in section 2.2.1. If some simulation history is available, then we can calculate the total number of stationary objects to be:

$$Num_{so} = \bigcup_{l=1}^L \bigcup_{t=0}^{TT} SD_{(l,t)} \quad (2.5)$$

where t denotes the simulation time and 0 denotes the begin time and TT denotes the end time of the simulation.

- Although it is not explicitly shown in 2.3, the level of detail of the simulation object models considerably affects the overall simulation time. Simulation of objects at the lowest level of detail will obviously require the most amount of time. We can save time by using aggregated or abstract models where possible. For example, for the combat simulation of two air planes, we can either simulate them at the lowest level of detail such as simulating each event of gun fire, missile fire and so on or we can simulate the combat result by simply sampling from a distribution. This probability distribution can be constructed based on expert knowledge or can be uniform (random) if no such knowledge is available.

The distance of the route ($dist_{R_j}$) is something that the planner cannot control or reduce. However, by distinguishing routes based on their distance, we can build experimental strategies based on this information. Depending on the particular problem domain, the effect of the route distance on the simulation time varies (e.g. the relation may not be strictly linear as shown in figure 2.4). Thus, the Executive must make a decision as to where the time must be saved, either in S or $\phi(j)$ s, depending on which is dominant.

We now present an approach which can produce simulation results within a given time constraint. The approach is best explained in two phases. In the first phase, we perform a set of n replications. It is hard to tell what is a good value for n but 20 is a commonly used number in simulation [25]. While the n replications are being performed, the CPU times of each replication for each route R_j are recorded².

²An issue exists whether CPU time is a good measure since it can vary depending on the load of the computer or the network.

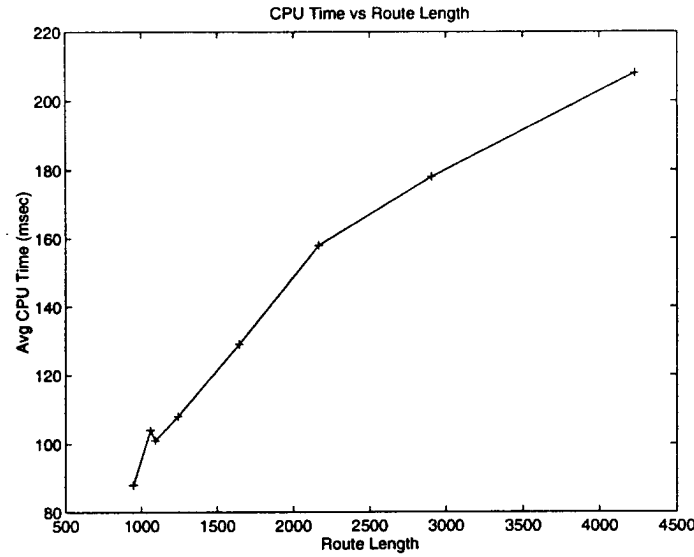


Figure 2.4. Plot of average cpu time vs route length

In the second phase, we make a decision as to which output analysis approach (strategy of performing replications and choosing the appropriate stopping conditions) to use based on the expected simulation time and the current remaining time. Before we go on to describe the algorithm, we define some variables:

- $mCPU(j)$ - denotes the mean CPU time over n replications for route R_j . Instead of the mean, other measures such as maximum CPU time of R_j can be used if you wish to take a conservative approach. We can also gather some sample data and draw a relation between the route length and the mean cpu time needed to simulate the route. Figure 2.4 displays such a graph which was obtained from the simulation trials in chapter 6. Here, we observe that the mean cpu time grows more slowly than the route length. Note that this graph only represents one application domain and is likely to be different based on factors such as the domain, the type of models and the number of objects.
- T - is the given time constraint.
- $Time_Left$ - is the remaining time to perform the simulations.

- Time_Used - is the time used in the first phase.
- Tot_Needed_Time - refers to the total simulation time (real time) necessary in order to produce results under the “Selecting best of k systems” (SelectBestk) algorithm.
- Time_Needed_Per_Rep - is the time needed to perform one round of replication of all the alternatives.
- Min_RN - is the minimum number of replications among all alternative, as calculated by the How_Many_More() function.
- How_Many_More(j) - using the SelectBestk algorithm, the exact number of replications necessary to reach a decision (select the best alternative among k alternatives) for alternative j is returned.

The basic idea of the algorithm is to perform the needed number of replications for each route as determined by the How_Many_More() function given that the remaining time is sufficient for the completion of this method. If the Time_Left is not sufficient, then we perform as many replications as possible given the time allowed. As the replications are performed iteratively, we also try to eliminate any alternatives that appear to be significantly worse than other alternatives. With the latter approach, the idea is to incrementally converge on the answer while trying to meet the time constraints.

```

Tot_Needed_Time = 0
Time_Needed_Per_Rep = 0
Time_Left = T - Time_Used
Min_RN = RN(1);
Num_Routes = N;
FOR each Route j (1 <= j <= N)
    RN(j) = How_many_more(j)

```

```

Time_Needed(j) = RN(j) * mCPU(j)
Tot_Needed_Time = Tot_Needed_Time + Time_Needed(j)
Time_Needed_Per_Rep = Time_Needed_Per_Rep + mCPU(j)
IF ( RN(j) < Min_RN )
    Min_RN = RN(j)
IF ( Tot_Needed_Time > T )
    WHILE ( Time_Needed_Per_Rep < Time_Left )
        FOR ( Remaining Alternatives )
            Perform replication
            Decrement Time_Left
            Eliminate any significantly worse alternatives
            Update the Remaining Alternatives set
            Update Time_Needed_Per_Rep
                with the new set of remaining alternatives
    ELSE
        SelectBestk()

```

The general method of SelectBestk is explained in more detail in the next section.

2.2.3 Output Analysis blocks : **Replicator, Evaluator, Analyzer**

Obtaining the right types of statistical analyses is just as important as performing the right types of simulation runs. With simulation, several different interpretations can be obtained from the same output data. This is the main concern of output analysis and is a distinct feature resulting from using simulation. Different analysis methods apply depending on whether a simulation is *terminating* or *steady state*. Because plans have a definite start and an end time, ours are terminating simulations. Suppose we are simulating k alternatives or systems. We describe in the following our current strategy for obtaining the appropriate outputs and their analyses.

Replicator

Replication provides the easiest form of output analysis. Because our domain is stochastic, we must perform n runs (replications) for each alternative i , each j th

replication generating a sample value for an output variable X_{ij} . Different random number streams are to be used for each run so that the results are independent across runs. For a sufficiently large n , due to the *central limit theorem* [25], the random variable A will be approximately normally distributed. This assumption allows the use of confidence intervals which is later calculated in the Evaluator block.

Another issue is using *common random numbers* (CRN) to provide a controlled environment for comparison among alternatives. This is to eliminate any “environmental differences” that can exist between different simulations. CRN is a standard variance reduction technique in simulation and we use it here across different alternative route plans within the same replication so that we may converge on the final answer faster.

The Replicator block controls the replication environment by controlling the random number streams for each replication. Depending on how the Trial block proceeds with the simulation, the designer may choose to vary the random number streams either in between each execution of the Trial block or within a single execution of the Trial block. In the air force route planning system in chapter 6, for example, a single execution (or replication) of the Trial block consists of simulating all alternatives using a common random number seed. The next time the Trial block is invoked, the Replicator will provide a different random number seed so that a different environment will be created in the next replication.

Evaluator

In its simplest form, the Evaluator serves as the accumulator of any relevant simulation data that is produced from the Trial Block. If the objective function within the Trial block produces a set of scores for each alternative, a straightforward way is to total the scores produced from the replications for each alternatives. Other relevant data such as elapsed CPU time for simulation of each alternative may also

be accumulated so that the Executive block may later analyze and predict future time usage.

Analyzer

Based on statistical criteria (e.g. highest mean, smallest variance, etc.), we can consider several alternate plans and choose the “best” plan for execution. Criteria other than statistical in nature can also be imposed, that are based on heuristics or expert knowledge.

This block is primarily responsible for analyzing data that was accumulated in the Evaluator. Different analysis methods may be used here based on the user’s requirements. The mean, the standard deviation, the variance and confidence intervals are some measures that we can acquire. For most of our applications, the mean of the replication results serve as the basic “data” point in our response surface or graph, representing the goodness of a plan. Variance can be a measure of predictability or stability when the variance is small. Confidence intervals are useful because given a sample output distribution and a confidence level x , it gives you the interval in which you can say with x % confidence that the real mean lies within the interval.

Again, let X_{ij} be the random variable from the j th replication of the i th system or alternative. Then let

$$\bar{X}_i(n) = \frac{\sum_{j=1}^n X_{ij}}{n} \quad (2.6)$$

be the sample mean which is an unbiased estimator of μ_i (i.e. $\mu = E[\bar{X}_i(n)]$). Also, let the sample variance be

$$S_i^2(n) = \frac{\sum_{j=1}^n [X_j - \bar{X}(n)]^2}{n - 1} \quad (2.7)$$

Then, borrowing heavily from Law and Kelton [25], we calculate for any $n \geq 2$, 100(1 - α) percent confidence interval by

$$\bar{X}(n) \pm t_{n-1, 1-\alpha/2} \sqrt{\frac{S^2(n)}{n}} \quad (2.8)$$

where $t_{n-1, 1-\alpha/2}$ is the upper $1 - \alpha/2$ critical point for the t distribution with $n - 1$ degrees of freedom. These points are given by a students t distribution table which can be found in [25]. The quantity that is added and subtracted from $\bar{X}(n)$ in equation 2.8 is called the *half-length* of the confidence interval. It is a measure of how precise we know μ . As shown in equation 2.8, the half-length is usually decreased by a factor of approximately 2 when the sample size n is increased to $4n$.

Using the confidence-interval approach, there are mainly two ways to compare among k systems or alternatives that are discussed in the literature [25]. We discuss the two methods here and later compare their differences in terms its efficiency and accuracy through a sample implementation in chapter 6. Since we are trying to select the best out of k alternatives, we must detect and quantify any significant pairwise differences in their means. First, let $\mu_i = E(X_{ij})$. An approach is to form confidence intervals for the difference $\mu_{i_2} - \mu_{i_1}$, for all i_1 and i_2 between 1 and k , with $i_1 < i_2$. Since there will be $k(k - 1)/2$ individual intervals, each must be made at level $1 - \alpha/[k(k - 1)/2]$ in order to have a confidence level of at least $1 - \alpha$ for all the intervals together. This is because of Bonferroni inequality which implies that if we want to make c number of confidence-interval statements, then each separate interval should be made at level $1 - \alpha/c$, so that the overall confidence level associated with all intervals' covering their targets will be at least $1 - \alpha$. Thus, in the first “iterative” approach, we iteratively continue the set of n replications while throwing away alternatives that are *significantly different*³ and also worse than all other alternatives. This approach, although quite accurate in terms of the results, can

³two means are considered significantly different if the confidence interval for their difference does not (or misses) contain zero

include unnecessary number of replications due to the fact that a constant number of replications is performed uniformly across the current set of alternatives.

The second approach, which we call the “non-iterative” approach (also called the “Selecting the Best of k Systems” approach) selects one of the k systems as being the best one while controlling the probability that the selected system really *is* the best one [25]. Let μ_{i_l} be the l th smallest of the μ_i ’s, such that $\mu_{i_1} \leq \mu_{i_2} \leq \dots \leq \mu_{i_k}$. The goal is to select a system with the *largest* (in our case) expected response (score), μ_{i_k} . Let CS denote the event of Correct Selection. The inherent randomness of the X_{ij} s makes it hard to say that we are *absolutely* certain of our CS. Thus, we prespecify a probability P^* of CS such that $P(\text{CS}) \geq P^*$ provided that $\mu_{i_k} - \mu_{i_{k-1}} \geq d^*$, where the minimal CS probability $P^* > 1/k$ and the “indifference” amount $d^* > 0$ are both specified by the user. If μ_{i_k} and $\mu_{i_{k-1}}$ are very close together, we might not care if we erroneously choose system $k - 1$ and thus avoiding a large number of replications to resolve this unimportant difference. The procedure stated below has the nice property that, with probability at least P^* , the expected response of the selected system will be no smaller than $\mu_k - d^*$.

The statistical procedure for this approach, involves two-stage sampling from each of the k systems. In the first-stage sampling, we make $n_0 \geq 2$ replications of each of the k systems and acquire the first-stage means and variances using the standard method. For $i = 1, 2, \dots, k$, let the mean for system i be $\bar{X}_i^{(1)}(n_0)$ and the variance for system i , $S_i^2(n_0)$. Then we compute the total sample size N_i needed for system i as

$$N_i = \max \left\{ n_0 + 1, \left\lceil \frac{h_1^2 S_i^2(n_0)}{(d^*)^2} \right\rceil \right\} \quad (2.9)$$

where h_1 (which depends on k , P^* , and n_0) is a constant that is obtained from a table. The table in [25] provides values of h_1 for $P^* = 0.90$ and 0.95 for $n_0 = 20$

and 40. Next, we make $N_i - n_0$ more replications of each system i and obtain the second-stage sample means as

$$\bar{X}_i^{(2)}(N_i - n_0) = \frac{\sum_{j=n_0+1}^{N_i} X_{ij}}{N_i - n_0} \quad (2.10)$$

Since each system i may involve different number of replications, we must weigh the means from each stage accordingly. We define the weights for the first stage

$$W_{i1} = \frac{n_0}{N_i} \left[1 + \sqrt{1 - \frac{N_i}{n_0} \left(1 - \frac{(N_i - n_0)(d^*)^2}{h_1^2 S_i^2(n_0)} \right)} \right] \quad (2.11)$$

and for the second stage $W_{i2} = 1 - W_{i1}$, for $i = 1, 2, \dots, k$. Finally, we obtain the weighted sample means

$$\tilde{X}_i(N_i) = W_{i1} \bar{X}_i^{(1)}(n_0) + W_{i2} \bar{X}_i^{(2)}(N_i - n_0) \quad (2.12)$$

and select the system with the largest $\tilde{X}_i(N_i)$.

2.3 Rule-based Systems and SBP

As part of our effort to answer the question why SBP(Simulation-based Planning) is a worthwhile method for route planning, we compare and contrast our method with rule-based systems. Rule-based systems have been used in several planning systems [35, 22]. However, when there are large degrees of uncertainty and also infinitely many variations of the inputs, thus requiring the system to consider infinite number of situations, rule-based systems seem to be inadequate. We claim that simulation can help in producing better solutions in such environments and in particular for the route planning domain. There are several advantages and disadvantages of rule-based systems. Our intent is to compare and contrast with SBP to find out how the SBP

methodology can be used to support conventional AI methods such as a rule-based systems.

- **Coverage of domain:**

According to Gonzalez and Dankel [7], rule-based systems may not be appropriate for certain domains– “domains that contain so many variations of the inputs where a system needs to consider and represent nearly infinite number of situations.” Covering such domains would require tens of thousands of rules to be developed, verified and maintained. “Developing such a knowledge base would be an extremely difficult process, even while ignoring the problems of its verification and maintenance” [7]. The air traffic control domain, an example domain suggested by Gonzalez and Dankel [7], contains potentially infinite number of situations given the potential number of air planes, their locations, types and speeds etc. Thus, to reason about all possible situations, you need (in the worst case) as many or more rules to represent all possible outcomes of plan executions. Using SBP, you model and execute individual objects, using whatever modeling paradigm that is suitable, in response to every possible situation and gather the effects of execution as the outcome. This is the first advantage of SBP as discussed in section 2.1. We demonstrate this feature by the following example.

EXAMPLE:

This example is from the domain of military mission planning for Air Force mission. Let us assume that the planning system is given a set of alternative routes and the planner’s role is to evaluate and suggest the best route. Let us assume that a route goes through a radar site. In the following, we denote the enemy force as the ‘red force’ and the friendly force as the ‘blue force’. In general, this is bad because it gives the red force a chance to send a red aircraft

after the blue aircraft that is flying on the route. This can be expressed as a rule somewhat similar to the following:

IF route crosses a radar coverage area
THEN hit probability is significant

However, there may be several exceptions to this rule. If, for instance, no red aircrafts are in the vicinity, or if red aircraft's current location and its max speed is such that it cannot catch up to the blue aircraft or if all the red aircrafts on the ground base are inoperable then hit probability is likely to be not high.

To handle this, additional rules or conditions are needed:

IF route crosses a radar coverage area AND
(no red planes within x miles OR
red plane's max speed is too slow OR
its location is too far away OR
no more operable planes on the ground)
THEN hit probability is NOT HIGH

With simulation, all of the above comes as a result of executing each models individually and seeing the output results of the simulations.

- For example, the radar site model will have a logic model where if a red is detected then it notifies any red planes that are reachable.
- If there were any red aircrafts that were able to receive the order to intercept the blue aircraft, then it will use its own internal behavioral/physical model to pursue the blue aircraft.

- Conditions such as the red's max speed being slower than blue and not being able to catch up will show up as a result of the simulation.
- If either all contacted red planes are inoperable or if no red planes exist, by individual planes not being able to respond and therefore resulting in no counter action by the red force, the resulting simulation will show a positive result.

Thus, in SBP, we do not have to explicitly reason or state what the consequences will be for each particular situation and action combination. However, this can come at the expense of modeling the knowledge at a lower level of granularity and perhaps requiring extensive time in computing the results.

The majority of the gain is in the Planner Object (Blue plane). SBP relieves the burden of having to reason about individual consequences beforehand by allowing the consequence to be observed via simulation. Often times, the problem with building an expert planning system for a complex domain like the military mission planning is building the expert knowledge-base itself. With SBP, the expert knowledge and their qualitative model can be obtained for one object at a time which can make the process easier.

Other non central object models are expected to have equally complex knowledge as a rule-base as far as the behavioral aspect of the model goes. However, again, with an addition of a physical model which may include equations for motion, fuel consumption and etc., along with the ability to sample appropriate probability distributions for uncertainty, SBP can reason about plans at a level that is much closer to the execution level, which is one of the main advantages of using simulation to evaluate plans. SBP can also represent abstract behavior and logic by incorporating traditional AI symbolic models at a higher level.

Consequently, if 1) the problem domain does not involve any kind of uncertainty; or 2) the required answer is only at an abstract level; and 3) there is no complex interactions among objects; then a higher level reasoner like rule-based system alone will be sufficient.

- **Planning for hypothetical situations**

Most knowledge-based systems represent *associational* knowledge. Such knowledge is a set of IF THEN rules that is based on an expert's heuristic ability or knowledge that was acquired mostly through observations. The expert may not understand the exact internal process but has the ability to associate the correct input/output pair for most, if not all, situations. As pointed out by Gonzalez and Dankel [7], this type of shallow knowledge is inadequate for solving new and previously unexperienced situations. If the expert had the theoretical (deep) knowledge, he would be able to solve the problem using this knowledge. Then, what do most shallow human experts do in such cases? They often perform what is called "trial and error." By subjecting the real process to a series of trial and error tests, they find a certain input combination that produces the desired output. Much like what we do when we try to find a route through a maze, we try different alternative routes until we find one that succeeds. Because the maze is so complex, we do not attempt to *think* in one step what the best route is. This is very much similar to SBP in that SBP makes simulation trials and records the errors until the best one is found. There is a pitfall here in that the experiment may not be valid if the model does not correctly represent the actual process. And this is an issue that must be addressed in the future. But given that it does, SBP tries different input combinations (sequences of actions) to see what kind of outputs are produced. In addition, SBP captures the uncertainties in the environment (whether it is due to incomplete information

or instability) through well established simulation methods such as probability sampling and replication.

Because of simulations' ability to experiment hypothetical situation—a direct result of 1—, SBP is still able to plan routes for hypothetical situations which the expert has never experienced. Much like how we simulate different hypothesis or experimental systems to find out their properties, we can simulate using each objects model to study the results of a plan execution in hypothetical situations. For example, SBP is an ideal candidate if one wants to plan for an aircraft whose weapon system has just been developed (and therefore no expert has any experience with this system or its effects) and only a simulation model exists. Other possible examples are when either the terrain or the weather is in a state in which the SME has never experienced, as was the case in Desert Storm.

- **Naturalness of knowledge representation**

Naturalness is one of the strong points of rule-based systems. Rules are a natural way for humans to express their knowledge. Based on a given situation, experts tell us what to do through rules. SBP does lack this type of naturalness, the naturalness of being able to express everything in words and logic. However, what may be unnatural to one person may be natural to another once a person becomes well accustomed to certain modeling types. We believe that by using multimodeling a user can choose whatever model type he finds most natural and easy to use which may not necessarily be in symbols or in logic.

Rule-based approaches lack the ability to modify/refine/adapt its knowledge to specific situations. SBP on the other hand lacks “smartness” or “heuristical knowledge” or “symbolic reasoning” capability. One may view SBP as a brute-force

method, in that a naive SBP system will simulate all possible combinations of situations on a particular route in order to evaluate it. Thus, it may waste effort in simulating useless plans. But with the help of appropriate heuristics that will prune out combinations that are unlikely or going to do poorly, and using various experimental design methods, we can make SBP smart enough to be practical in real environments.

CHAPTER 3 A MULTIMODEL DESIGN FOR DECISION MAKING

3.1 A Truck Depot Example

The Truck Depot problem was originally taken from [10]. Since the problem contains both non-intelligent objects (e.g. basin, trucks, valves) and intelligent objects (e.g. robots or people) in equal emphasis, the problem inspired us to find a solution to optimization by combining the two fields of simulation (simulating non-intelligent objects) and AI (simulating intelligent objects) under a unifying modeling paradigm. Unlike the mission planning domain which appears in following chapters, this problem lies on the reactive spectrum of planning, belonging in the realm of intelligent control or decision making.

Figure 3.1 shows the aerial view of the truck depot, which represents the concept model of system. The depot contains one basin with two input pipes P_1 and P_2 , carrying two different chemicals and one output pipe P_3 . Each pipe has a valve (V_1 , V_2 and V_3) which controls the flow. Each valve has a servo motor attached enabling the human operator to remotely close or open the valves. Empty tanker trucks arrive at the depot and wait until they can move under pipe P_3 to be filled with the mixture from the basin. When each truck leaves the basin, its cargo is tested. If the truck has been filled with an acceptable mixture, it leaves the system; otherwise it dumps the cargo and returns to be refilled. All mixture which overflows from the basin or the truck is treated as waste. In our version of the problem, the capacity of each tanker truck is constant. The truck depot's class model is shown in figure 3.2 and its instance model is shown in figure 3.3.

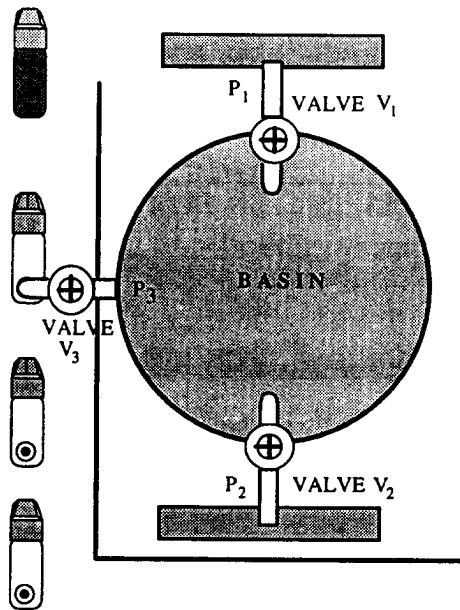


Figure 3.1. Concept model of truck depot (Aerial view)

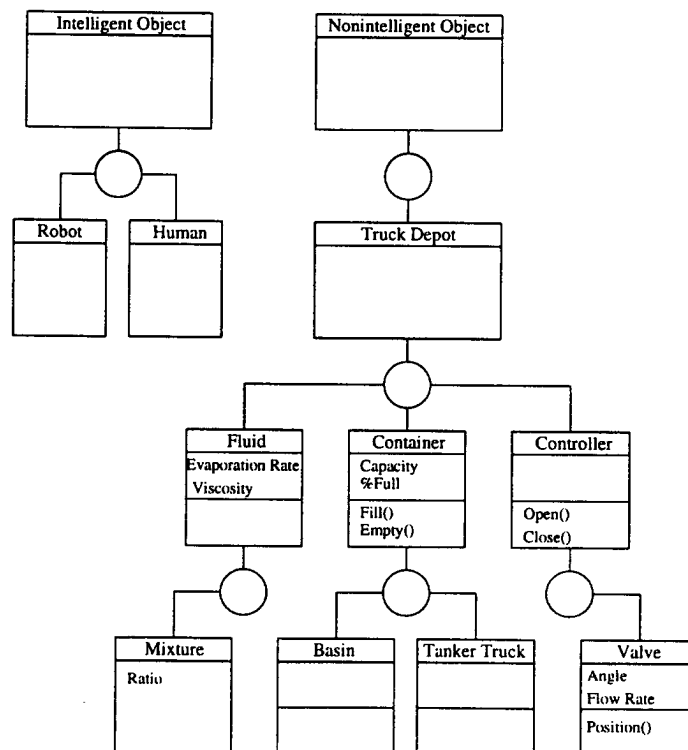


Figure 3.2. Class model of truck depot

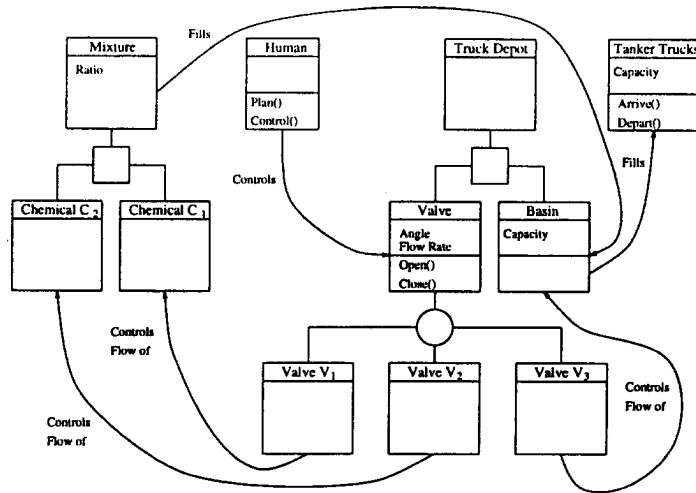


Figure 3.3. Instance model of truck depot

The intelligent objects control the non-intelligent objects in the instance model. In our case, the human operator makes decisions and controls the valves in the depot, while achieving the goal of maximizing the profit of the depot by maximizing the number of trucks filled while minimizing the cost. The depot is charged for the total amount of chemicals that flow through the input pipes during the period in which it is open. The trucks are independent objects which arrive according to an exponential distribution over the period of time when the depot is open.

We also have the notion of time to consider in our simulation. The start of simulation time corresponds to an opening time of the depot and the end of simulation time corresponds to the closing time in the real world. Figure 3.4 shows the control system of our simulation.

Let us examine the basin and the effect of the valves in more detail. Consider the basin containing a mixture (as illustrated in Figure 3.5). The two input pipes P_1 , P_2 carry two different chemicals. Pipe P_3 fills each tanker truck with a mixture of the two chemicals. Each pipe has a valve (V_1 , V_2 and V_3) which controls the chemical flow. Each valve has a servo motor attached that is controlled remotely and

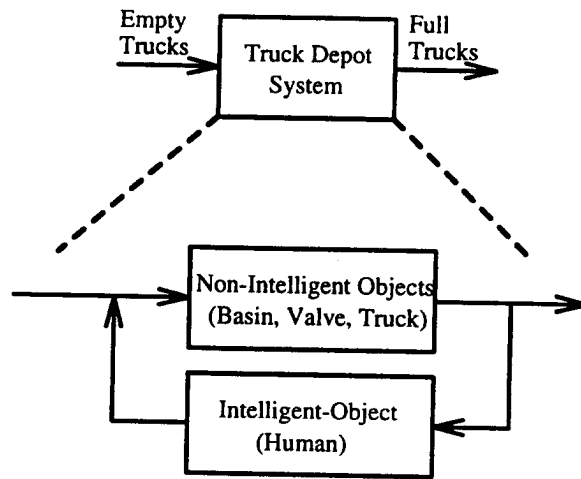


Figure 3.4. Top view of system

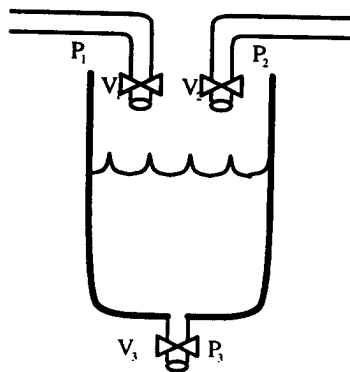


Figure 3.5. Basin containing mixture

simultaneously by the intelligent object. The intelligent object controls each of the valves by opening or closing them.

The primary goal of the intelligent object is to fill the tanker trucks with an acceptable mixture while minimizing the system cost. If we further extend this problem to a real world situation, we can consider that the owner of the depot is paid for each truck properly filled and is charged for the total volume of chemical that flows through P_1 and P_2 . Thus, the owner's goal is to use the least amount of chemicals while filling as many trucks as possible. All mixture which overflows from the basin or the truck is treated as waste. Also, any remaining mixture in the basin after some deadline (e.g. end of the day) will also be waste. And finally, when a truck contains an unacceptable mixture, the contents are dumped and refilled later. The mixture is not acceptable (or *bad*) when the proportions are off by more than 10% from the correct ratio. It is considered acceptable (or *good*) otherwise.

3.2 Intelligent Objects

Because the arrival of the trucks is dynamic and there is no determined number of trucks *a priori*, no type of offline planning is possible. As illustrated in figure 3.6, we have adopted Brook's subsumption architecture [5] to integrate the different level modules. Our hierarchical approach is different from a conventional hierarchical planner in the sense that each of the levels have access to input and output. Since it is possible to have conflicting output commands, we need some type of coordination or mediation [23] among them. Adopting the subsumption architecture's method of mediation, the outputs are suppressed by a higher level when the higher level makes an overriding decision. In the original version of the subsumption architecture, a time period is specified, during which the output will be suppressed. However, because our simulation is discrete, we will allow the output to be suppressed by a higher level for one time step until the next event arrives and causes another output.

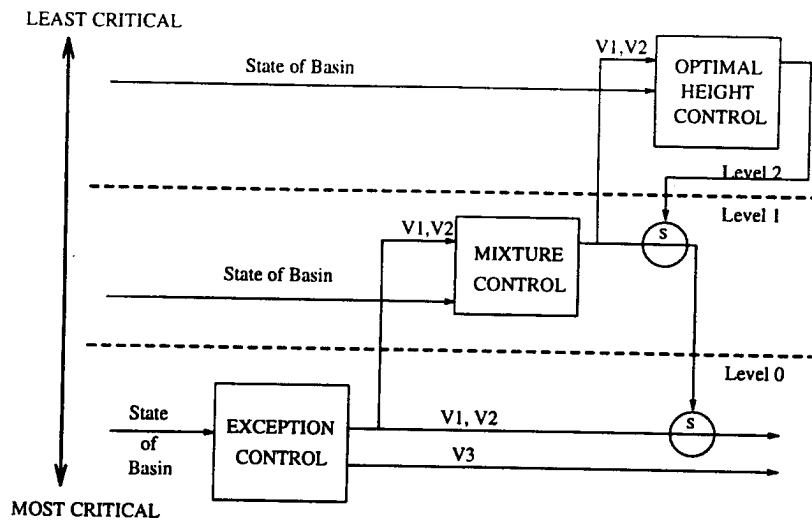


Figure 3.6. Multimodel planner

The multimodel planner has multiple levels that are divided based on how critical the reaction of each level is to the overall success of the planner. The levels also reflect the reactivity of the control in that the lowest level module is the most reactive module whereas the highest level is the least reactive. In general, however, this may not always be the case. The least reactive module may contain the most critical level of control in the system. The following hypothetical example illustrates a typical case where the more reactive modules are also more critical to the overall success. For example, when they plan deliberately to decide where to turn while driving or controlling the car at the same time. If the car reaches a red light before the turn, it is more critical that the driver react and stop at the red light than worry about the turn.

3.2.1 Exception Control

The Exception Control module exists at the lowest level in the hierarchy, which is the most reactive and the most critical. In our problem, there are two critical situations. First, overflow either from the basin or the truck must be avoided since spilled mixture can never be recovered. Second, unless it is close to the end of

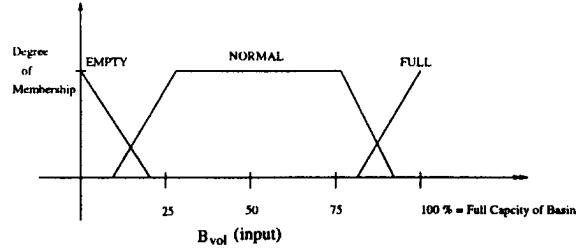


Figure 3.7. Fuzzy set for mixture height

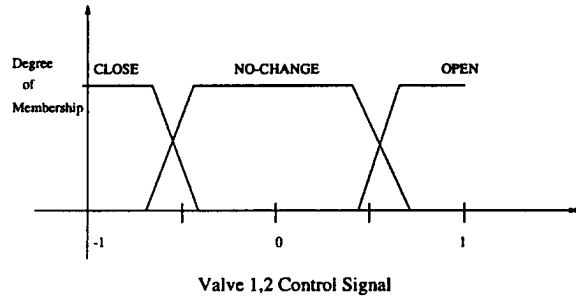


Figure 3.8. Fuzzy set for valve control

simulation time, the planner should avoid having an empty basin since no truck can be filled. The Exception Control takes the state of the basin as input, which is the volume of the mixture in the basin, B_{vol} and the volume of the mixture in the truck, T_{vol} . With B_{vol} , fuzzy logic [45, 44] is used to infer whether the basin is in an OVERFLOW or EMPTY state. With T_{vol} , fuzzy logic is used to decide if the truck is in an OVERFLOW state. Only when these conditions arise, does Exception Control react and sends an output command (CLOSE or OPEN valve V_n). NO-CHANGE—which corresponds to a null signal—is sent otherwise. The fuzzy input and output sets for OVERFLOW CONTROL of the basin are illustrated graphically in figures 3.7 and 3.8.

At this level, the output fuzzy set is identical for both valves V_1 and V_2 . Because when an overflow (or empty) occurs or is about to occur, both of the valves need to be closed (or opened) at the same time. To control the overflow of the truck, the volume of the mixture in the truck is monitored and inferred by fuzzy logic to be

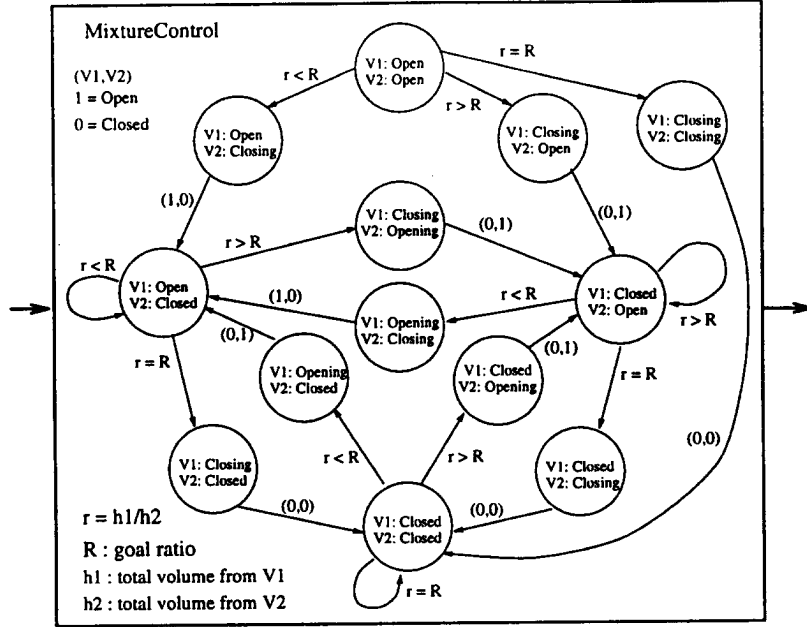


Figure 3.9. FSA for Mixture Control

either in the state NOT_FULL or FULL. The output set for valve V_3 is similar to the one for valves V_1 and V_2 . Fuzzy logic seems well suited since we are not concerned precisely when these events occur, but rather the appropriate time to start monitoring and controlling to prevent overflow. The fuzzy logic model best mimics the actual performance of the human operator at this level.

3.2.2 Mixture Control

The Mixture Control module is responsible for maintaining the *correct* ratio of the two chemicals in the basin. Initially, the planner is given a ratio R which is to be maintained throughout the simulation. The mixture is considered acceptable and to be of the *correct* ratio if the actual ratio r falls between the range $R - 5\%$ and $R + 5\%$. The type of model used for Mixture Control is a finite state automaton as shown in figure 3.9. In the figure, the current state of the valves is represented by the tuple (V_1, V_2) where V_n can be one of the following Open, Closed, Opening, Closing. Depending on the current state, the next state is reached by choosing the appropriate

transition. If the next state reached is a transition state (state that contains one or more valve settings ending with *ing*), an output command is sent that will actually create the event to change the physical state of the valves. For example, at time t , the current state is (Open,Open). Then at time $t + 1$ (i.e. the next even time), the input is $(r, 1, 1)$. If $r < R$, the Mixture Control module will switch the current state to (Open,Closing). The current state of the FSA will stay at this state until the input is $(r, 1, 0)$. There is an implicit self loop (which is omitted in the figure) that transitions to itself when the input is anything other than $(r, 1, 0)$. When the input becomes $(r, 1, 0)$ at some time $t + n$, where n is the time delay for the valve to be totally closed, the current state switches to (Open, Closed). The input $(r, 1, 0)$ is a confirmation from the basin model that the commands were properly carried out and that valve V_2 has successfully closed.

Once the ratio of the mixture becomes acceptable and close to its goal ratio R , Mixture Control will send commands to close all the valves to maintain the steady state. However, the output command may be suppressed by the higher level, therefore never reaching the basin model. This may also be the case for the Exception Control module. When sending the output command, the Mixture Control module should also consider the output coming in from the Exception Control module. If the Mixture Control module decides that its output is more critical, it will suppress the lower level output and replace the command with its own. For more detailed explanation of the suppressor function, refer to [5].

3.2.3 Optimal Height Control

Finally, the Optimal Height Control module controls the height to maximize the profit. Because this module is less reactive and involves more symbolic knowledge and reasoning (heuristics) than the other lower level modules, rule-based reasoning is best suited for the task. The notion of optimal height is time dependent. Since

the simulation has a start and an end time, the intelligent object can have different strategies for maintaining the height at different times. Another consideration that the Optimal Height Control module takes into account is the speed of chemical flow. The flow rate of valve V_3 depends on the height of the mixture in the basin. Since our Optimal Height Control module uses heuristics, optimality is not guaranteed. Included in this module, is the Evaluator which evaluates the overall profit of the system, during and after the end of simulation. The profit is (amount of reward per unit of volume)*(total volume of good trucks) – (amount of money charged per unit of volume) * (total volume of input). This function is included as part of the Optimal Height Control module. The formula used is $Profit = N(V_{gt}) - C(V_i)$ where N is the amount of reward per unit of volume and C is the amount of money charged per unit of volume. V_{gt} represents the total volume of good trucks and V_i represents the total volume of input.

3.3 Non-Intelligent Objects

3.3.1 Model Design

Modeling the basin poses several challenges. The model state includes continuous and discrete variables, constraints, and functional relationships.

The volume of the mixture in the basin changes continuously throughout the simulation. The input signals that control the three valves give continuous outputs, but they change at discrete times. The tanker trucks move through the system as discrete objects and are constrained when waiting to be filled because: the signal for valve V_3 must open the valve; the basin must have enough mixture to fill a truck; and the filling area must be empty.

We chose a Petri net to model the top level, see figure 3.10. The inputs to the model are tanker truck arrivals and control signals for opening and closing each of the valves. The output from the model includes statistics showing the number of trucks

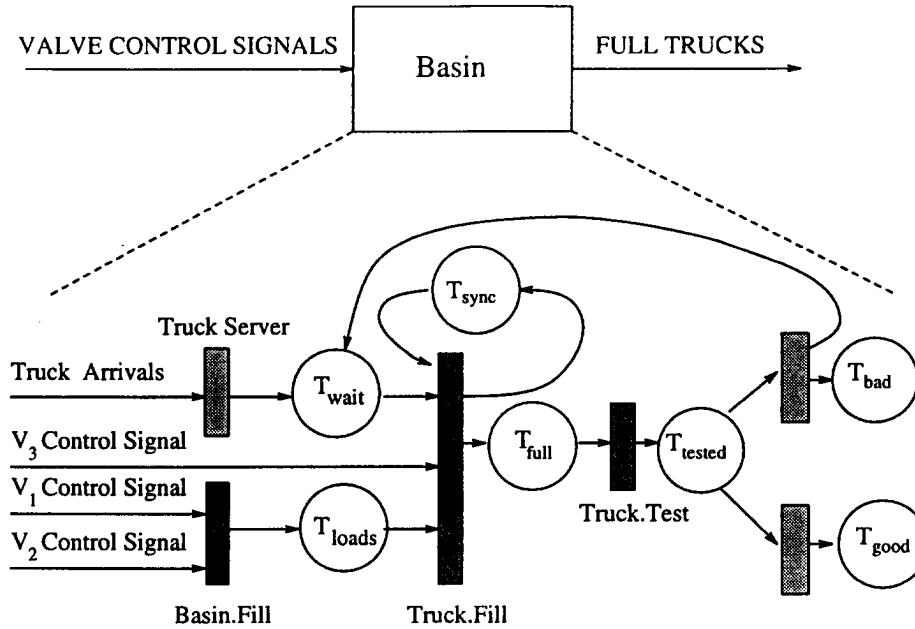


Figure 3.10. Petri net model of the basin

that were filled properly, and the volume of each chemical that was poured into the basin during the simulation.

The *Truck Queue* transition is refined by an S/S/1 queuing model. It was chosen to model the tanker trucks waiting to be filled. The queue is used to maintain the trucks arrival order.

The transition *Fill Basin* is refined by a block model, see figure 3.11. The control signals are passed to the refining models *Valve 1* and *Valve 2* respectively. Each refining model returns a value representing the control to be applied. The function block, *Mixture Volume in Basin* takes the control input and returns a value that states how many tanker truck loads the basin currently holds.

Fill Truck is also refined by a block model (similar to figure 3.11) that shows the relationship between valve V_3 and the *Mixture Volume in Truck* function block. Valve V_3 takes as input the control signals, *OPEN_V3* and *CLOSE_V3*, then returns the amount of control to be applied due to the current state of valve V_3 . The function

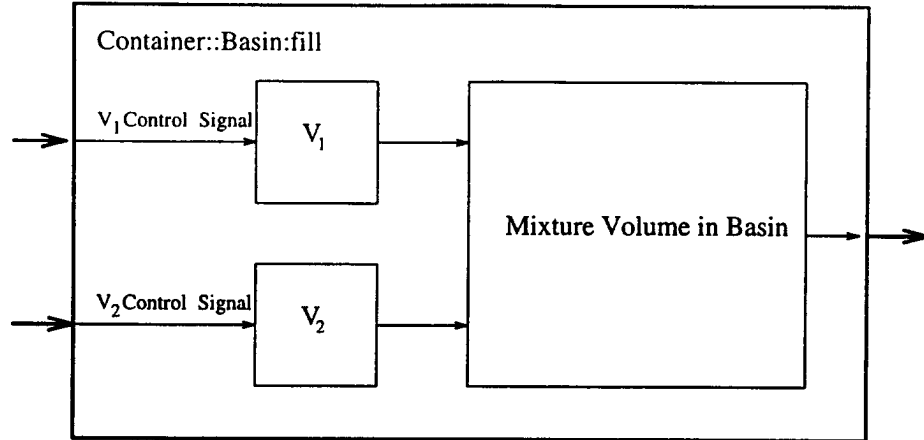


Figure 3.11. Refinement of transition *Fill Basin*

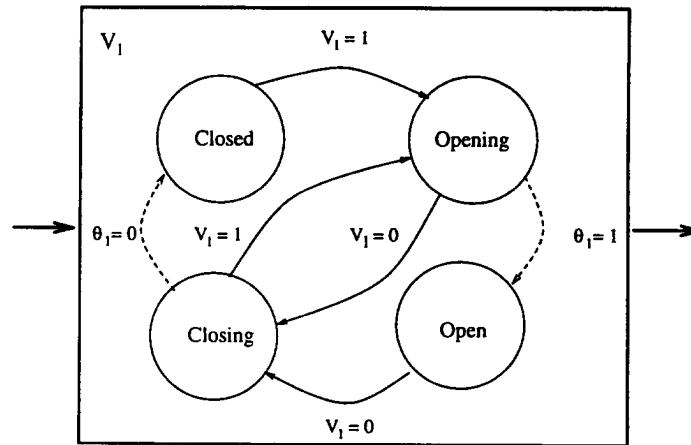


Figure 3.12. FSA model of valve V_1

block *Fill Truck* takes the control value as input and returns a value when the truck has been filled.

Each of the function blocks are refined by a finite state automaton. There are five FSAs, but the three FSAs that refine the valves are similar to the refinement of valve V_1 .

The function block *Valve V_1* is refined by the FSA shown in figure 3.12. The input for this FSA is the control signal for the valve V_1 , ($V_1 = 1$ or $V_1 = 0$). The FSA will change state if an internal transition is detected, ($\theta_1 = 1$ or $\theta_1 = 0$).

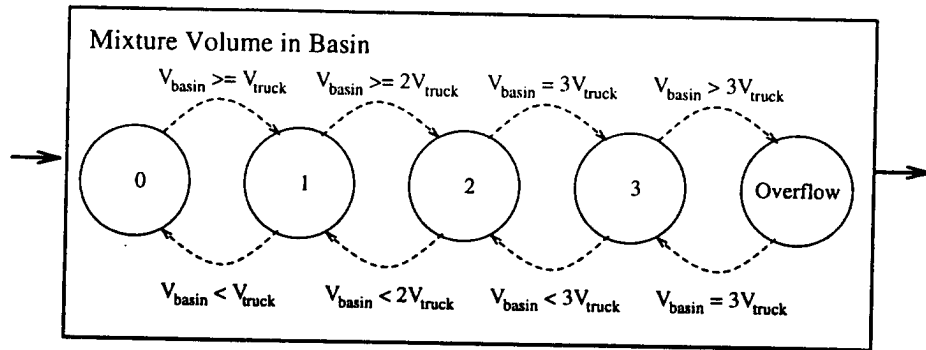


Figure 3.13. FSA model of the volume of mixture in the basin

The *Fill Basin* function is refined by the FSA displayed in figure 3.13. The input for this FSA is the amount of control that is being applied to the system by valves V_1 and V_2 . The output is how many truck loads of mixture the basin contains.

The function block *Mixture Volume in Truck* is refined by an FSA with three states. The input to this FSA is the amount of control applied by valve V_3 , the system output is the state of the truck, whether the truck is *FILLING*, *FULL* or *OVERFLOWING*.

The control equation $\dot{x} = Ax + Bu$ is used to model the continuous state variables in the system. Where x is the subsystem state, u is the control from the valves, and B is the amount of control being applied.

3.3.2 Model Execution

A coordinator is used to create and execute the multimodel system. After each level of the model is created, the levels are connected to their refining models, then each level is initialized. Our initial conditions for the truck depot example are: the basin is empty, the valves are closed, and no trucks are waiting to be serviced. These initial conditions can be changed, however, depending on the goal of the simulation. The coordinator's task during execution is to dequeue events from the FEL and direct the event to the model specified.

CHAPTER 4 ADVERSARIAL ROUTE PLANNING

4.1 Mission Planning for Computer Generated Forces

For planners that work as part of the Computer Generated Force (CGF) simulation [24] using the Distributed Interactive Simulation Environments, uncertainties of enemy's actions have to be handled in real time since the Computer Generated Forces have to fight against the opposing force (which are normally human trainees) in real time. As stated in chapter 1, the CGF planning domain selected here is assumed to have complete information of the environment including the enemy and the terrain. The complexity of the problem lies in the uncertainty of enemy's action and varying properties of the terrain. The mission planner presented in this chapter is an integral part of a larger project of the Institute for Simulation and Training (IST) called "Intelligent Autonomous Behavior by Semi-Automated Forces in Distributed Interactive Simulation" which was funded by the U.S. Army Simulation Training and Instrumentation Command (STRICOM). The goal of the planner is to automatically derive plans for a CGF command entity node, at the company level initially, so that the force will provide an Army trainee with an effective training experience. Therefore, the planner represents a decision making algorithm at the level of a company commander. Planning is only a small part of the overall project, which includes efficient line of site (LOS) determination, terrain reasoning, intelligent target acquisition and behavior representation for CGF entities. The planner takes orders from the battalion level and translates these orders, with a tight coupling with the terrain analyzer, into efficient plans for the CGF platoon entities. In addition to planning for its subordinate units, the planner must also be able to monitor the execution

of the plan, react to unexpected situations and replan if necessary. We present a Simulation-Based Planner that can meet such challenges.

4.2 Planner Architecture

Figure 4.1 displays the architecture of our planner in relation to the IST CGF Testbed [24]. Each commander in the IST testbed is simulated by a Command Entity (CE). The Planner performs major functions of this entity. The planner has two phases: the Reactive phase and the Planning phase—where a phase is a group of states that collectively display a behavior. Only one phase is active at any given time and the starting phase is the Reactive Behavior. The decision as to which phase becomes active is made by the current active phase based on the inputs. There is no single “main” algorithm that controls the whole process. Thus, the decision is made in a distributive manner. In particular, the decision to give up planning and report to a higher level unit is made by the Planning Behavior. The inputs are either OPODs¹ or SITREPs² and depending on the type of SITREPs and OPODs, different decisions will result. Each Planner in the CE is made up of the following components.

4.2.1 World Database (DB)

The World Database contains information about the battlefield. This is not a complete spatial representation of the battlefield (the Terrain Analyzer (TA) has this information) but a simplified database which mainly contains information that is known to the CE (and not known to the TA). Since the TA does not have any information regarding the location of enemy or friendly units and does not keep track of the locations, the planner needs to keep track of these locations and the status of the units in the World Database. This database is created as soon as the CE starts to

¹Operations Orders - refer to section 4.3 for more details.

²Situation Reports - refer to section 4.3 for more details.

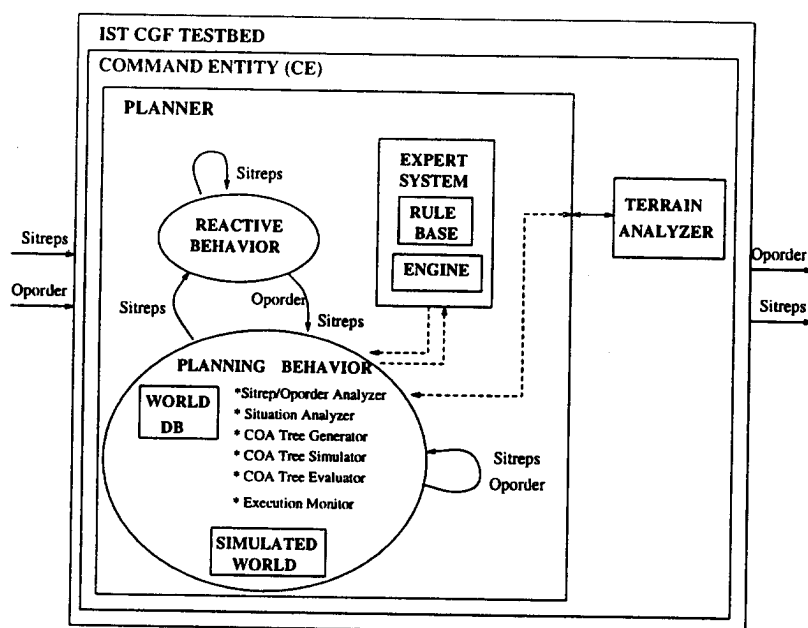


Figure 4.1. Planner Architecture

exist. Initially it contains its own location and will be updated with new information as it becomes available to the CE via SITREPs or OPORDs.

The battlefield is a spatial model, divided into rectangular regions represented as the elements of a matrix. This is a very low resolution of the battlefield because the purpose of this matrix is mainly to speed up the look up time of unit locations and other information by organizing the linked lists into regions. Each element of the matrix is linked to a linked list that contains all the information the CE has about that region. Each node will have more exact locations along with other available information such as status of the unit.

4.2.2 Reactive Behavior

The Reactive Behavior module displays reactive behavior necessary for survival when immediate action is required. This behavior may be different for different types of entities. The module is initialized with a generic set of behaviors at the start and may be modified with any reactive behaviors provided by an OPORD.

4.2.3 Planning Behavior

The Planning Behavior module has the ability to generate orders for its subordinate entities from an OPORD given by a higher level entity. This module is made up of the following smaller modules where the order in which they are presented actually coincides with the algorithm steps of a typical planning process.

1. **SITREP/OPORD Analyzer** parses the SITREP/OPORD to update the World DB.

- **OPORD**: is further parsed to generate a list of task(s) to be achieved. The Situation Analyzer is called next with this list.
- **SITREP**: is analyzed to decide if any immediate action is required, if any replanning is required, or if any SITREP needs to be generated. The Execution Monitor is called with the decision.

2. **Situation Analyzer (SA)** is a collection of rules that analyze the given situation using the World DB to generate the appropriate constraints for the ROUTE or the DEF_TACT_POS call to the TA. The decision as to which of the two calls to make first depends on problem size reduction. In other words, in a given situation, the call ROUTE may produce many routes whereas a DEF_TACT_POS call may have produced few tactical positions. In such a case, we can first make a DEF_TACT_POS call to acquire a set of tactical positions and then call ROUTE with these tactical positions which will reduce the number of returned routes due to the given constraints. Thus, the SA will perform some alternate calls of ROUTE and DEF_TACT_POS to produce an appropriate number of alternate routes. The COA Tree Generator is called with these alternate routes.

3. **Course of Action (COA) Tree Generator** uses the set of alternate routes produced by the SA to generate a COA Tree where the 1st level contains alternative subunit combinations. and 2nd level contains alternative route combinations. The following levels can contain other alternatives such as varying the role of platoons in different formations. We can extend the tree as much as we want with any other possible alternatives at each level. Also, if time permits, we may want to add an adversary tree to the 2nd level. This adversary tree can represent possible reaction behaviors by the enemy given the current set of action by the friendly unit. Also note that some alternatives may be omitted at this stage and later generated during the Simulation/Evaluation step.

Once such a tree is generated, the tree is pruned using various methods and rules before it is passed onto the COA Simulator. Many alternatives can be pruned away by using a military expert knowledge system. However, we must not prune away too much since many alternatives should be left to be explored via simulation. The purpose of using simulation may be lost if the choices have been made already. Next, the COA Tree Simulator is called with the COA Tree.

4. **The COA Tree Simulator** is invoked to simulate the set of COA trees that have been generated. This is done by creating a Simulated World and performing the simulation of friendly and enemy units by time slicing between actions (move, look, fire) and observation by each unit. It is also time sliced between friendly and enemy units. Different methods can be used in simulating friendly and enemy units. One method is to allow the enemy units to have the same planning capabilities as the friendly units but with different tactics. This method would be quite realistic, but it can be quite time consuming. In general, a complete simulation will be more time consuming than a temporal

projection using rules. If computing capabilities are limited, we can perform simulation at different levels of abstraction [26, 16] where each higher level will use less computational power. Another solution is to let the enemy units follow a less sophisticated planning process allowing limited intelligence. This is the approach we are taking for this particular application.

The actual simulation algorithm is as follows:

```
While (planner active) do
    Update entity state variables
    Perform line of sight (LOS) check
    Engagement check
    Update current clock time by  $\Delta T$ 
End While
```

The simulator takes each level of the COA subtree and simulates each route and calculates a score for each friendly subunit per each route. In the current version, the enemy unit is simulated in a very limited manner. The enemy unit is assumed to remain stationary and only engage in combat when an opposing force unit has been sighted. Currently, our method employs the Aggregate Combat Model [8]. Two important assumptions made in general is the 3 to 1 rule and that each of the units are equivalent in size. The 3 to 1 rule states that the break-even point is at a force ratio of 3 to 1, (3 being the defender and 1 being the offender) which is a reasonable assumption in most cases since a defender is prepared to defend in a favorable terrain. To satisfy the second assumption, the demonstration mission involves only units in the platoon level.

Since the planner does not currently have access to the TA, the Line of Sight check is being done by a simple function LOS_check which checks the distance between two units. At present time, the terrain is assumed to be flat and open, not affecting the line of sight calculation. With the TA connected, the planner should be able to have access to a more accurate LOS check which will account for different types of terrain that may obscure the view. The speed of the units are currently determined only by the terrain type: ROAD has 100% mobility, GROUND has 80% and FORD has 50% mobility.

For each course of action, the simulator operates as follows. State variables defining an entity's position and orientation are updated at each time slice. In low mobility areas or areas with a steep terrain gradient, the movement is slower. Also, for some terrain features, as with fords or chokepoints, a simple queuing model can be executed to keep track of entities that must wait for entities that are blocking the path. Service times and speed values are obtained by sampling from a probability distribution appropriate for the blocked area. A line of sight (LOS) check and range calculation is done between the entity being simulated and known enemy locations. If the enemy is within range of certain weapons (such as a HEAT or Sabot round), an engagement will ensue. We are unsure as to the level of detail required to simulate the engagement for planning purposes. However, these may be extended as behaviors such as "seeking cover" are integrated into the planner. The simulation proceeds, while updating the simulation time by ΔT until either the plan has been fully simulated, or the planner is interrupted.

Initially, the Simulated World is created from the World DB and then the status of the world is updated as entities are being simulated. The simulator uses the

TA to update the Simulated World. The outcome of the simulation is then fed into the COA Tree evaluator.

5. **The COA Tree Evaluator** is invoked to evaluate the simulation result by calculating scores using the following formula:

$$\text{score} = \text{strength of unit} + \text{proximity to OBJ}(\%)$$

If at any point in time during the simulation the strength of a unit is below a certain threshold (5 %), it is considered to be destroyed and no further simulation will be run on that particular branch of the COA tree. Thus, the overall simulation strategy is branch and bound. Depending on the order of the calls, however, it is possible to simulate the COA tree in a somewhat depth-first manner. It is possible to simulate the 2nd level of the SPLIT tree before the 1st level of the NO_SPLIT tree. The each unit's score is stored in every route_node of the COA tree representing the simulation result of that particular branch.

Once the evaluation is done, different interpretations can be made on the scores themselves. To follow the military's operational concepts of acting with an *initiative*, incorporating unpredictability of actions into the planners is a major task. A possible solution is to allow the planner to choose nondeterministically among those plans that have evaluation scores above some threshold. Another important extension is to allow the enemy to react nondeterministically during simulations so that the evaluation scores will come out differently at different runs.

6. The Execution Monitor

The Execution Monitor is the main driver of the Planning Behavior module. Its actions are:

- (a) Issue the set of chosen subtasks in the plan to each units in an OPORD format.
- (b) Execute its own subtask if any.
- (c) If any SITREP is received,
 - i. Call the SITREP/OPORD Analyzer.
 - ii. If the decision returned calls for
 - immediate action, it is handled by the Reactive Behavior module which accesses the mini Expert System to react accordingly. In doing so, the Reactive module also takes into account any Engagement Criteria given in the OPORD.
 - replanning, the SA is called to start a planning process with the newly updated World DB.
 - giving up planning at the current level, the CE sends SITREP to its higher unit reporting of its current status and waits for further orders.

4.2.4 Expert System

The mini Expert System module contains rules to aid the planning process in making decisions such as choosing routes, choosing best COA tree, performing analysis of situations, OPORDs and SITREPs. In the prototype version, very simple rules have been used and they exist as if-then statements inside the implementation. As more depth military expert knowledge are acquired, it may become reasonable to implement the expert knowledge as a separate Expert System.

4.3 Interface between Planner and other Command Entities

Similar to how Operations Order (OPORD) are sent as directives to subordinates in military Command and Control (C^2), the CGF Command Entity is also expected

to receive and send OPORDs. The standard military format for OPORDs contain considerable amount of overlap throughout the different sections. Since our OPORDs will be sent as messages within the program and each message has a limited size of 300 bytes, there was need to modify the military standard format to make it more concise.

A standard military OPORD consists of five paragraphs: Situation, Mission, Execution, Service Support and Command and Signal [12]. For our application, we are mainly concerned with the first three where Situation describes the situation and missions of enemy forces (if known) and friendly forces, Mission states the mission the unit issuing the OPORD is trying to achieve, and Execution expands on the Mission statement by describing the specific tasks to be accomplished by each subordinate units.

In our OPORD format, the Execution paragraph takes the form of a Synchronization Matrix where the rows represent the tasks for each subunit and the columns represent different phases of the mission. The Synchronization matrix is issued by the commander to its subordinate units where individual unit locates the row that contains the orders for that specific unit and executes the tasks issued to them. Each unit transitions to the next phase of the mission based upon its own transition code. It can initiate its own transition: On Own Initiative or it can only transition after receiving an order from its commanding unit: On Order. A corresponding matrix to our demonstration order is shown in figure 4.2. This same matrix is sent to all three companies and each company extracts the applicable row of tasks to be accomplished. Since the id number of our company is 1, the orders are contained in the first row. The second and third rows do not directly affect the planner since they are orders for company 2 and company 3, respectively. Unless some coordination was required among the three companies, a company is only concerned with the orders

	Phase 1	Phase 2	...	Phase n
Company 1	SEIZE OBJ at 32500, 28750 ×	DEFEND 32500, 28750	...	
Company 2	SEIZE OBJ at 38700 40500 ○	MOVE to 32500, 28750	...	
Company 3	MOVE to 50000 40000 ○	SEIZE OBJ at 54750, 40100	...	

Transition Code: ○ On Order
× On Own Initiative

Figure 4.2. Synchronization Matrix for Demo OPORD

given specifically to the unit. Also note that since the prototype assumes there is only one phase to a mission, phase 2 is also ignored in the current version of the prototype.

Situation Reports (SITREPs) are another type of order that is sent to the superior unit by a subordinate unit to report either scheduled situations (e.g. when a mission has been accomplished, when a unit hits a check point, etc.) or unscheduled situations (e.g. when a threat is identified, when overall combat strength falls below a predetermined level, etc.).

4.4 Interface between Terrain Analyzer and Planner

As mentioned earlier in section 1, the mission planner is a part of a larger project at IST. The planner must integrate with many other components in the IST project, but for the most part it must work with the IST's Terrain Analyzer.

4.4.1 Terrain Analyzer

The Terrain Analyzer is the planner's only source of information about terrain and thus the planner uses the TA quite extensively during the planning process. The TA is responsible for route planning, finding tactical positions, computing Line of Sight and answering questions about terrain features. The interface between the TA and the planner is established by four types of calls; ROUTE, DEF_TACT_POS, LOS and TERRAIN_FEATURE.

- ROUTE: Given the maximum number of routes, start and end position, the unit boundary, the minimum percentage of concealment, the mission type and the direction of approach to the OBJ (located at end position), the TA returns multiple routes that satisfy the given constraints. Note that the direction of approach does not apply to cases when the end position does not have an enemy unit and the mission type is not a SEIZE mission. Any intermediate enemy position that needs to be avoided can also be given along with a radius and the TA will generate routes avoiding these locations radius distance away from the avoid locations. Each returned route has a route id, length and percentage of concealment relative to the route. The actual route is represented as a piecewise linear curve made up of a set of line segments. Each line segment contains not only its begin and end points but also the percentage of mobility, passable width, probability of LOS to the OBJ and the terrain_type (ground, road, ford).
- DEF_TACT_POS: Given the type of mission, this call requires the TA to provide locations for a given type of tactical position(s). The current position of the unit and the OBJ must also be given. There are three different types of position: SUPPORT_BY_FIRE, DEFENSE, and ATTACK. We have two types

of mission: SEIZE and DEFEND. Finally, the enemy location must also be provided to the TA for SUPPORT_BY_FIRE and ATTACK positions.

- ALOS: This call directs the TA to perform an area to area Line_of_sight determination. Locations 1 and 2 are given, each with Radius1 and Radius2. The Radius1 describes the circular area centered at 1 and Radius2 describes the circular area centered at 2. 1_#_PTS constrains the number of points where LOS should be tested within circle centered at 1. 2_#_PTS constrains circle at location 2 in the same manner. If the Radius is 0 for both locations then a simple point to point LOS is performed. The returned data is the probability of sightings over the #_PTS tested within the two areas.
- TERRAIN_FEATURE: This call requires the TA to return all the terrain features that are included within the radius of a given point. The planner supplies the TA with a center point and a radius and the TA returns one or more of the following types as a terrain_feature: ROAD, FORD, GROUND, CHOKE_POINT.

4.5 Demonstration Mission Scenario

Figure 4.3 illustrates the demonstration scenario. The friendly company 1 (the company entity receiving the OPORD) is situated at Assembly Area (AA) located at (50000, 52500). Company unit 1 is made up of 3 platoons: platoon A is made up of 4 M1 Abrams Main Battle Tanks and each of the remaining 2 platoons B and C are made up of 4 Bradley Infantry Fighting Vehicles. There are two enemy platoons: OPFOR(opposite force) platoon A is made up of 4 M1 tanks located at (32500, 28750) and OPFOR platoon B is made up of 4 M2 fighting vehicles located at (45000, 46250). The OPORD given to company unit 1 is to “SEIZE the Objective at (32500, 28750)”. Typically, an area called unit boundaries are given along with

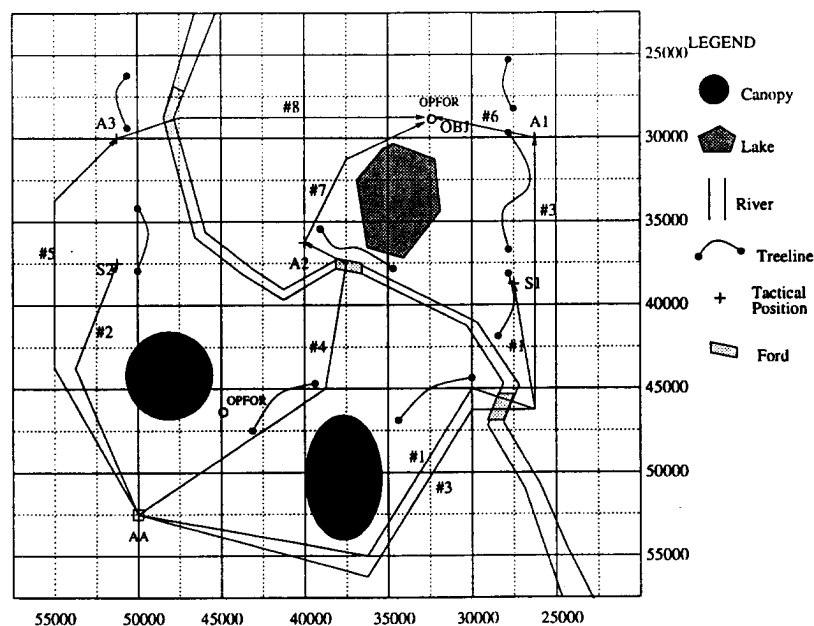


Figure 4.3. Company Mission and Routes

the OPORD. The unit boundaries define the area in which the unit is expected to operate. The company unit boundaries for company 1 are given as the rectangular area in the figure 4.3.

The goal of the command entity is to accomplish the mission with minimal loss of strength. To find a mission plan that will satisfy this criteria, the planner simulates several alternative plans, compares the results and chooses the best one. However, these alternative plans need to be generated by the planner first. The planner has to be careful not to generate too few or too many alternative plans since too few may not include a potentially good plan and too many may take too much computation time. Figure 4.3 also shows the various tactical positions and routes that can be obtained through calls to IST's Terrain Analyzer. Thus, the alternative plans are mainly based on two alternatives: different subunit combinations and different route set combinations. In other words, the alternative plans depend on which unit or units go on which route.

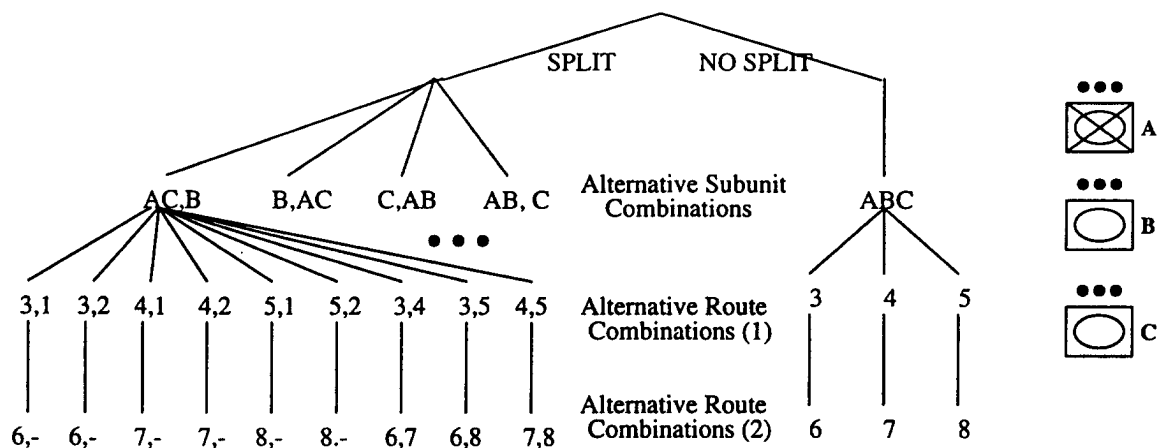


Figure 4.4. COA Tree without Adversary Tree

Figure 4.4 shows the COA tree generated by the prototype. The SPLIT subtree describes the course of action for a company where the company will be divided. Given that a company has 3 platoons, the number of routes needed is always greater than or equal 1. If two platoons go one route and one platoon go another, two routes are needed in total. If each platoon goes on a different route, three different routes will be necessary. The 1st level of this subtree will contain all the possible combinations of splitting a 3 platoon company. Currently, the set of combinations are read from a data file. The justification is that some heuristics must be applied in subdividing the company and since there are no expert systems to aid this process in the prototype, an expert can be consulted in creating the data file using his expertise and knowledge of the Company's composition. The heuristic used in this scenario is not to allow Platoon A to travel alone at any time since M1s are considerably slower and lower power than M2s. This restricts the SPLIT combination to 4 sets: (AC,B) (B,AC) (C,AB) (AB,C).

From these combinations, the Create.COA.TREE generates possible combinations of route sets. Since the mission is SEIZE, at least one route should lead the platoons to an ASSAULT_POS. These routes are 3,4,5 according to 4.3. Thus, the possible set of route combinations are (3,1), (3,2), (4,1), (4,2), (5,1), (5,2), (3,4), (3,5),

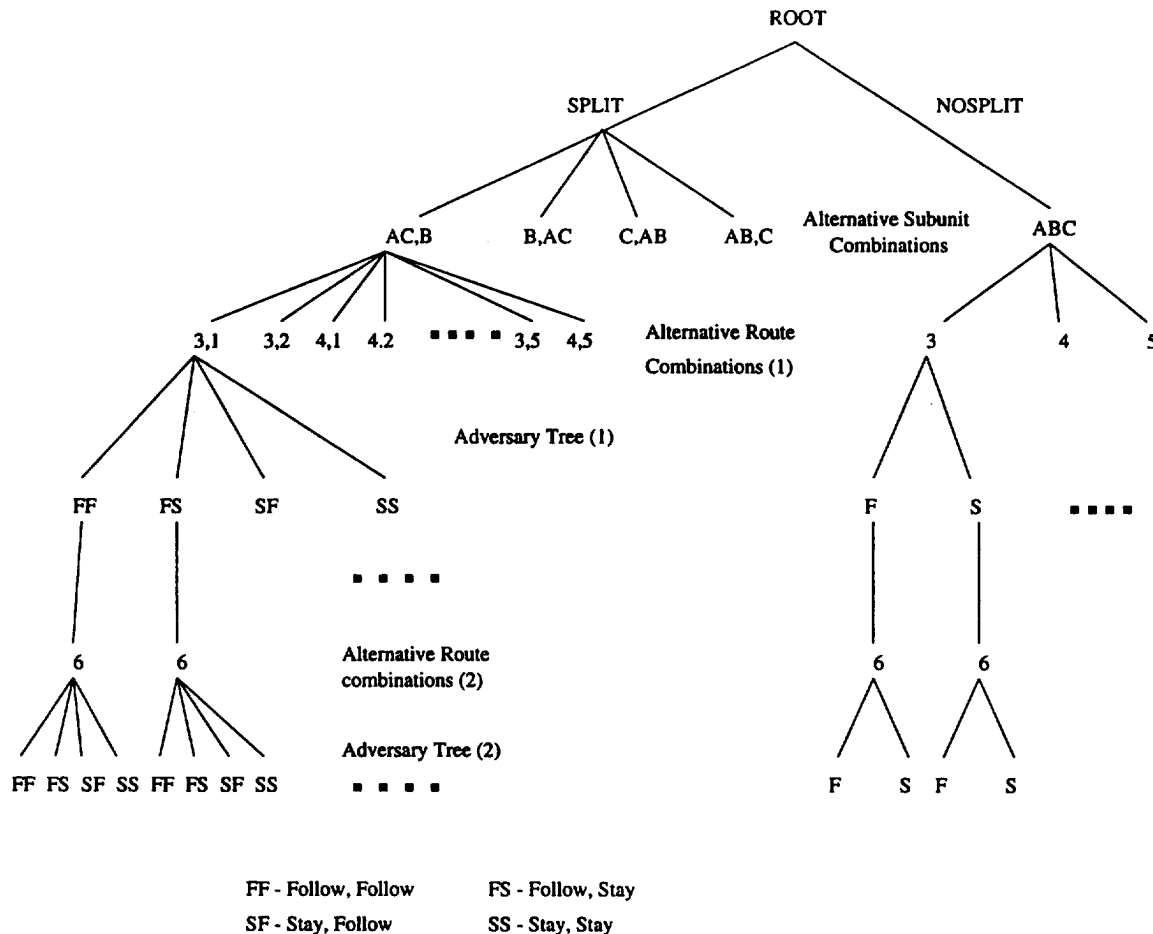


Figure 4.5. COA Tree with Adversary Tree

(4,5). The second level of route combinations in the COA tree contains the routes that connect each of the 1st level routes to OBJ if possible. For example, route 6 extends route 3 to OBJ. However, route 1 is not extended to the objective because it's a SUPPORT_BY_FIRE position.

The NO_SPLIT subtree has a single alternative subunit combination (ABC) since no split up is allowed in the unit combination. Therefore only a single route set alternatives that lead to an assault positions (3,4,5) are possible. The second level routes are (6,7,8) respectively. No pruning is being done in the current implementation.

Figure 4.5 shows the COA tree generated with the adversarial actions included. For now, we assume that there are only two possible actions Follow or Stay. Follow

means that the enemy unit will exhibit a behavior of active pursuit, given that it is within range. In other words, if the enemy unit is not close enough to actually detect and follow, the subtree with the Follow behavior will not be any different from the subtree with the Stay behavior. Stay refers to the behavior where the enemy will remain stationary in its current location but will still fight back if fired upon. The different behaviors can be best seen by comparing figures 4.6 and 4.7. Figure 4.6 shows the case when the enemy is simulated with the Stay behavior. Note that this is also the basic behavior of the enemy for the simulations of the COA tree without adversary tree. Figure 4.7 depicts the situation where the enemy unit is simulated with the Follow behavior. Assuming that the enemy will continue to pursue the friendly unit that is traveling on route 4, the shaded polygon near route 4 which shows the area where combat will continue to occur is extended. Thus, the area of combat in figure 4.7 is larger compared to the shaded area that appears in figure 4.6 and therefore the score is expected to be lower in the first case. For the tree in figure 4.5, four pairs of behaviors (FF, FS, SF, SS) are generated in the SPLIT subtree. A pair is generated because one F or S corresponds to each route and there are always two routes in each branch of the tree under the SPLIT subtree. Accordingly, in the NOSPLIT subtree, only a single behavior (F or S) is specified for each branch.

4.6 Planning Results

From figure 4.3, one should be able to observe that any friendly units traveling on route 4 is likely to engage in combat with the opfor unit stationed at (45000, 46250). The friendly unit may not be totally destroyed but considerable amount of strength may be lost during combat and therefore will result in a lower score. Thus, any plan that includes route 4 will have lower scores than other plans. Due to space limitations, we only include some results of interest. We first show the results obtained using the

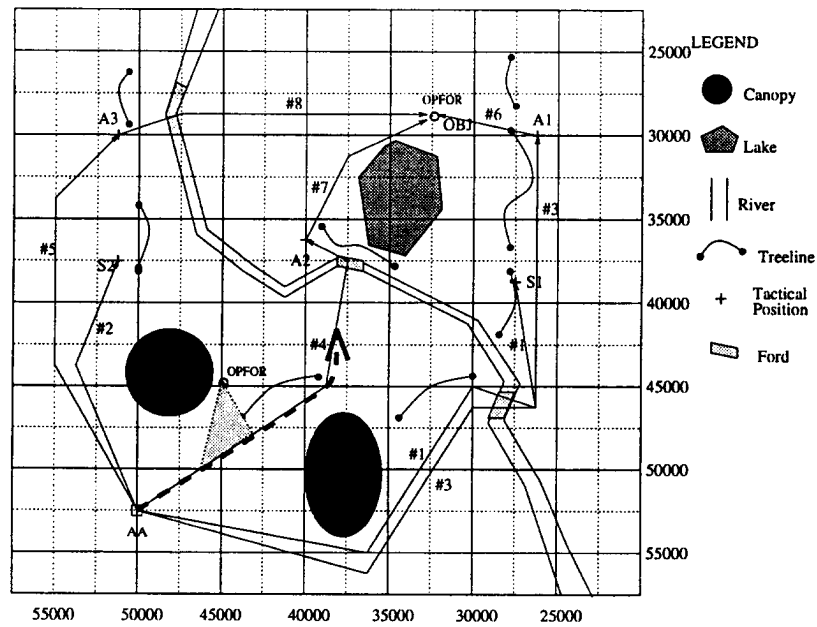


Figure 4.6. Simulation of Stay behavior on Route 4

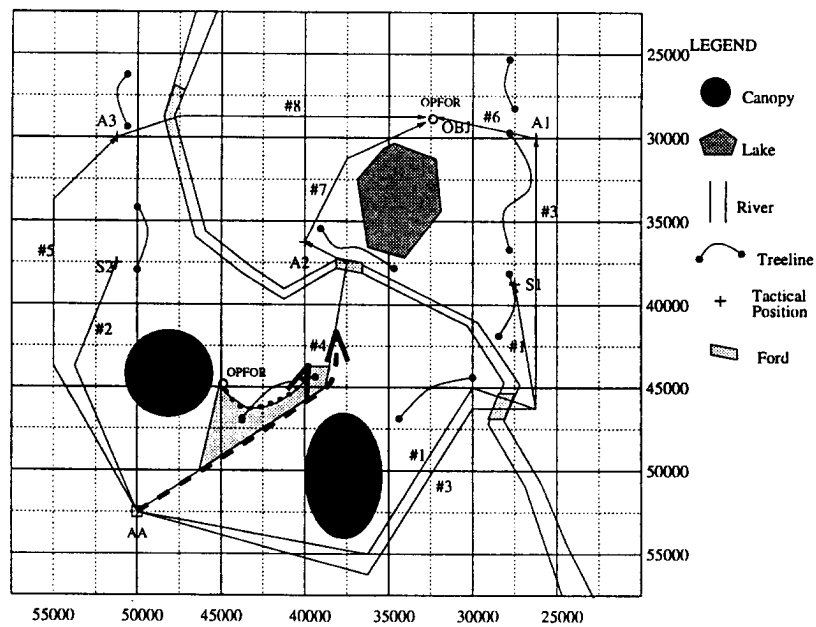


Figure 4.7. Simulation of Follow behavior on Route 4

COA tree without an adversary tree. This result is for the alternative plans where the company stays together as a company (NOSPLIT) and travels on a single route to the objective: (Note that routes 1 and 2 are not considered since they end at Support_By_Fire positions.)

Result of NOSPLIT routes at phase 1:

[A-B-C 3] : 165.007523

[A-B-C 5] : 131.897141

[A-B-C 4] : 89.443901

Result of NOSPLIT routes at phase 2:

[A-B-C 3 6] : 128.000000

[A-B-C 5 8] : 128.000000

[A-B-C 4 7] : 95.140198

Next, we show the evaluation scores with the adversary tree. The results support our prediction that route plans which include route 4 is considerably lower in score than the other alternative route plans. Moreover, of the two subtrees that includes route 4, the subtree where the enemy was simulated with the Follow behavior is even lower. Since all the other routes 3, 5, 6, 7 and 8 do not have an enemy unit nearby, there is no difference in result whether the subtree includes a Follow or Stay adversarial branch.

Result of NOSPLIT routes at phase 1:

[A-B-C F 3]:	165.007523	* F: FOLLOW
[A-B-C S 3]:	165.007523	S: STAY
[A-B-C F 5]:	131.897141	
[A-B-C S 5]:	131.897141	
[A-B-C S 4]:	89.443901	
[A-B-C F 4]:	65.971626	

Result of NOSPLIT routes at phase 2:

[A-B-C FF 3 6]:	182.000000
[A-B-C FS 3 6]:	182.000000
.	
.	
[A-B-C SF 5 8]:	182.000000
[A-B-C SS 5 8]:	182.000000
[A-B-C SF 4 7]:	119.666664
[A-B-C SS 4 7]:	119.666664
[A-B-C FF 4 7]:	114.666664
[A-B-C FS 4 7]:	114.666664

There are basically four different types of information that is given as input to the planner at different stages of the planning process. When the planner is first started, it is given an OPORD, a set of subunit combinations to be used in the generation of the COA tree, and the initial strength of the individual subunits. During the plan generation process, the planner will make various calls to the TA and get back from the TA information such as tactical positions and routes. Once these plans

are generated, the planner simulates them and outputs the final evaluation scores of alternative plans in decreasing order. The CGF planner prototype currently runs on an IBM 486 PC and the Sun Unix Workstations.

CHAPTER 5

NON-ADVERSARIAL ROUTE PLANNING

The CGF Mission Planning problem, presented in chapter 4, did not possess a lot of uncertainty in the way it was defined. To show how SBP can be used in problem domains that possess significant amount of data uncertainty—including uncertainty due to noise—we have selected the Mars Rover route planning problem as our second application domain.

5.1 Rover Route Planning

A major difficulty with the Computer Generated Forces Mission Planning was the overwhelming amount of domain knowledge that we had to learn which left little room for experimentation and further refinement of our methodology. In realizing the problem, we decided to apply our methods to a canonical problem. Route (or path in robotics terms) planning with mobile robots is a well known problem and thus is a good candidate. When there is little uncertainty involved, as is sometimes the case in many robot route planning problems, the existing approaches such as potential fields [2] do quite well. But, when uncertainties exist in the environment, these methods alone cannot produce good results. The main algorithm of producing a graph of traversable paths and searching the graph for a best route still remains the same, however. The part that is different is how we measure the *goodness* of a route. If the goal is to select a route that is the shortest in distance, we can use any of the standard algorithms that exist for finding shortest paths in a graph. But, if the problem is in an environment that is unknown or uncertain, we must use a different method which we claim is SBP.

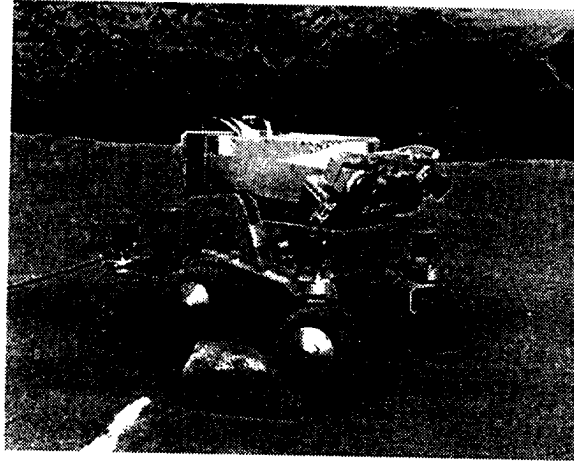


Figure 5.1. Mars Microrover. Provided by permission of Jet Propulsion Laboratory [28]

In 1996, NASA plans to launch a spacecraft to Mars to explore the environment of the planet [28]. The spacecraft will carry an 11 kg rover, called the Microrover, that will move around the vicinity of the landing site to explore the territory for a duration of approximately 1 to 4 weeks. Figure 5.1 shows the Microrover traveling over a rock.

Because the Martian surface is not completely known, JPL is undergoing a process of performance evaluation of the rover's autonomous navigation system with varying terrain characteristics. The Microrover testbed contains the Microrover vehicle and an indoor test arena with overhead cameras for automatic, real-time tracking of the true rover position and heading. In the arena, they have created Mars analog terrains by randomly distributing rocks according to an exponential model of Mars rock size and frequency created from Viking lander imagery. JPL has decomposed the rover navigation task into four functions: 1) goal designation; 2) rover localization; 3) hazard detection; and 4) path selection. Goal designation is expected to be performed by mission operators on Earth. Hazard detection is largely connected with problems in sensors such as stereo camera pair and light stripe ranging sensor. And thus, although the four functions are integrated, we will focus mainly on rover localization

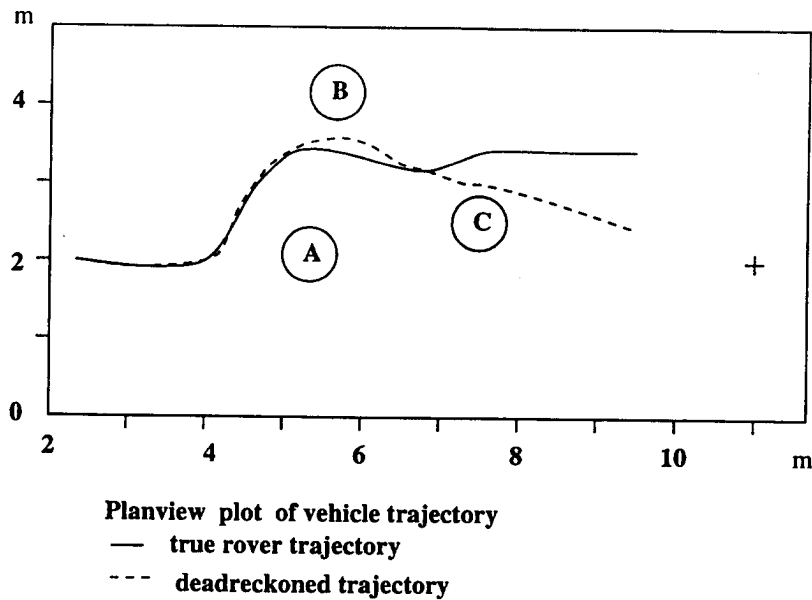


Figure 5.2. Rover trajectory for a run with three rocks

and route selection. Rover localization is presented as a major issue since dead reckoning error often prevents the rover from properly recognizing when it has reached the goal location. Figure 5.2 shows a planview plot of the vehicle trajectory for a run with three rocks. The solid line represents the true rover trajectory and the dashed line represents the deadreckoned trajectory. The rover scraped its hubs against rocks B and C. According to JPL's report, the results of this and other trial runs imply that scraping against rocks introduce large heading errors.

Currently, path selection is achieved by a simple behavior control algorithm which is reactive and does not take any excess knowledge—such as maps—into account. Thus, our intent is to build a simulation-based route planning system that will enable the rover to simulate various alternative path traversal behaviors and/or parameter settings such as turn rate and sensor reading rate to estimate the optimal settings at which the rover is likely to have least amount of deadreckoning error and, therefore, most likely to succeed.

5.2 Simulation-Based Rover Route Planning System

Figure 5.3 illustrates the basic components of our Simulation-Based Rover Route Planning (SBRRP) system. Initially, the planner takes the goal location of the route as input and selects a route plan for output. This selected plan is the input to the Control Subsystem which performs a supervisory control of the process. The output of the rover process is the actual sensory output of the rover. The sensory output will include camera images, hazard detections and position information. Along with the plan, a simulation log (the simulation data that was produced previously during the plan evaluation process) of the chosen plan is provided as input to the Control Subsystem. This can be used to serve as a reference model to track the state of the execution in order to monitor its progress towards the goal. The monitoring information can be used further to tune the system towards the goal (i.e. correct its route or position estimation) or to generate a failure signal to the planner as soon as it decides that the current route is unlikely to succeed.

5.2.1 Planner

The planner has two major modules:

1. The **Plan Simulator** simulates alternative traversal configurations for different terrains. In terms of the rover configuration, two factors are varied: 1) unit distance (the amount of distance traveled in between hazard detection) and 2) unit angle (the number of degrees turned in between hazard detection). The Path selection behavior algorithm is another possible factor to vary but is not considered at this time. For the terrain, we also vary two factors: 1) rock sizes and 2) rock frequency. Thus, the simulation is based on the physical and empirical models of the rover and the terrain.

Table 5.1. Power Usage for different subsystems

Subsystem	Power Usage(W)
Driving	8.28
Steering	6.70
Hazard Detection Scan	7.33

- The **Mars Rover Model** includes a physical model which include specific characteristics given by the JPL document: translation (max 0.67 cm/sec), rotation speed, step climbing ability (max 19.5 cm), and specification for the hazard detection sensors (range of view is about 120 deg. with 30 cm max distance). In addition, it's power usage is also documented and is shown in table 5.1.

Since we are simulating in a 2 dimensional space, we can represent the position of the rover with respect to a fixed global frame at time k as a vector

$$\mathbf{x}(k) = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} \quad (5.1)$$

where $x(k)$ and $y(k)$ describe the cartesian location and $\theta(k)$ represents the heading measured counterclockwise from the positive x-axis.

The robot's motion is controlled by the control input

$$\mathbf{u}(k) = \begin{bmatrix} T(k) \\ \Delta\theta(k) \end{bmatrix} \quad (5.2)$$

where $T(k)$ is the speed (unit distance) per time step and $\Delta\theta(k)$ is the rotational value per time step.

The actual position at time $k+1$ is obtained by the state transition function f ,

$$f(\mathbf{x}(k), \mathbf{u}(k)) = \begin{bmatrix} x(k) + T(k)\cos\theta(k) \\ y(k) + T(k)\sin\theta(k) \\ \theta(k) + \Delta\theta(k) \end{bmatrix} \quad (5.3)$$

To model the complete state of the rover system, we must include the power usage $p(k)$ at time k and also keep track of the deadreckoned position. Let $\hat{\mathbf{x}}(k) = (\hat{x}(k), \hat{y}(k), \hat{\theta}(k))$ represent the deadreckoned position and heading at time k . Then $\hat{\mathbf{u}}(k+1)$ is obtained by the state transition function $\hat{f}$,

$$\hat{f}(\hat{\mathbf{x}}(k), \mathbf{u}(k)) = \begin{bmatrix} \hat{x}(k) + (T(k) + e_d(k))\cos\hat{\theta}(k) \\ \hat{y}(k) + (T(k) + e_d(k))\sin\hat{\theta}(k) \\ \theta(k) + \Delta\theta(k) + e_h(k) \end{bmatrix} \quad (5.4)$$

$e_d(k)$ represents the position error and $e_h(k)$ represents the heading error at time k sampled from triangular distributions presented in the following text. Thus, the complete rover system state is represented by a vector

$$\mathbf{r}(k) = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \\ \hat{x}(k) \\ \hat{y}(k) \\ \hat{\theta}(k) \\ P(k) \end{bmatrix} \quad (5.5)$$

and

$$P(k+1) = P(k) + p_i(k) \quad (5.6)$$

where i represents one of the three subsystem functions: Driving, Steering and Hazard Detection Scan. $p_i(k)$ represents the power usage of subsystem function i occurring at k . The power usage values, as shown in table 5.1, are not dependent on time and are constant throughout the simulation.

Table 5.2. Error Distribution of 40 runs

	Heading error(deg)	Position error(m)
Min	0.2	0.22
Mode	8.0	0.69
Max	37.2	2.45

What is time dependent is the value of i since due to power constraint of the rover only one of the three functions can occur at time k .

Empirical data gathered from JPL's trial runs are used to model $e_d(k)$ and $e_h(k)$. Also, hazard detection error has a failure rate of 1 in 1000 given that hazard frequency is 1/100. The error models are based on the dead reckoning and heading error statistics gathered by JPL for 40 runs on artificial Mars *nominal* terrain. The error statistics form a triangular distribution given in table 5.2.

These errors represent cumulative errors gather at the end of each run. The mode of the true distance traveled is 8.39 m and thus we can naively scale this to form a new position error distribution values U_{error} for each unit distance traveled using the following equation:

$$U_{error} = \frac{\frac{P_{error} * D_{goal}}{8.39}}{\frac{D_{goal}}{U_{dist}}} \quad (5.7)$$

P_{error} is the position error as shown in the above table. D_{goal} is the total distance between the start and goal locations in a given route planning problem. U_{dist} is the unit distance selected for the simulation. Intuitively, the numerator is the amount of total position error scaled to the current route distance and the denominator is a rough estimate of the total number of samplings (i.e. simulation time steps) likely to occur during the simulation. And thus, the equation provides a rough estimate of the position error for each unit distance traveled. Scaling the heading error distribution

Table 5.3. Scaled Sample Error Distribution

	Heading error(deg)	Position error(m)
Min	0.001	0.18879
Mode	0.0315	0.59213
Max	0.186	2.0999

is not as straightforward as scaling the position error distribution due to the fact that it is not really possible to predict the total number of degrees the rover will turn during the simulation of a route traversal. So, based on the assumption that the rover is likely to make a turn for every simulation step, we have arbitrarily scaled the heading errors down by $\frac{D_{goal}}{U_{dist}}$.

Table 5.3 shows a scaled triangular error distribution for simulations whose U_{dist} is 7.2 cm and D_{goal} is 766.94 cm.

- The **Mars Terrain Model** creates Mars analog terrains using a rock size-frequency distribution developed by Moore [29]. Currently, JPL is experimenting their Microrover by performing test runs on terrains which has been created artificially by randomly placing rocks according to this Moore [29] model . Moore's model is based on data obtained from images taken by Viking Lander 2. Since a similar rock density is expected for the Microrover experiment, the same model can be used. The original Moore's model for rocks down to a diameter of 0.14 m is represented by $N = 0.013D^{-2.66}$, where N is the cumulative frequency of rocks per square meter with diameters of D and larger. This model predicts that about 18.8% of the landing site area is covered by rocks. However, the model used by JPL so far in creating the actual test terrains is based on the modal value of the surface rock cover over the whole planet, which is estimated to be at 6%. The terrain created from this model is called *nominal*. A computer simulation is under development at JPL in order to test terrains

with rock frequencies ranging up to 19%. Thus, our simulated terrains will also be *nominal* in terms of total rock cover percentage although the sizes and frequency will vary.

Using the models of the rover and the Martian surface, we perform multiple simulations of each route. In general, there are several ways to proceed in the simulation. If a set of routes are given beforehand— as in the case of our previous work in mission planning—one can divide each route into phases and proceed the simulation in a breadth-first manner (each level being one phase of the route). This way, the planner will at least have some idea of the goodness of each route, should the time limit the planner to make a decision before it has performed a complete set of simulations. If, however, no pre-determined routes are to be used during execution as in the case of the rover, then a natural way is to proceed in a depth first manner. Another possible approach is to divide the simulation artificially into several segments and proceeding breadth-first, simulating different segments from different simulation trials. This would involve significant overhead since we need to keep a record of all the states of each trials in between each simulation. Thus, our approach here is to perform a complete simulation, trial by trial.

To reduce the amount of computation, we can use the A* search method if we can build a heuristic function which can estimate the cost of the remaining route. Another possibility is the branch and bound method used in the area of Operations Research. Due to uncertainties which exist in the models, simulations must be performed multiple times using the available stochastic information to capture the overall effect and also to reduce the variance of the outcome variables.

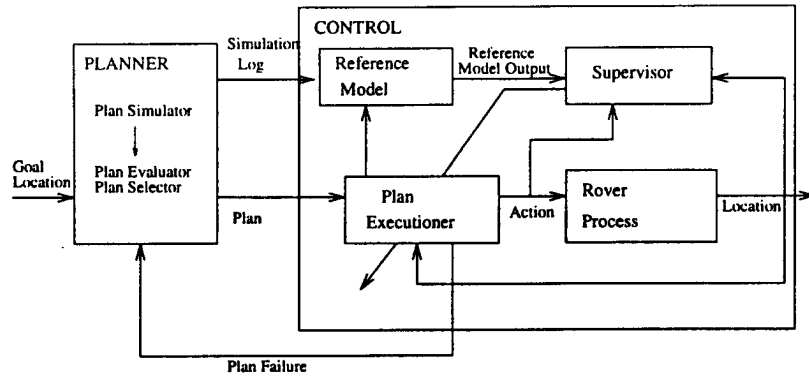


Figure 5.3. Simulation-Based Rover Route Planning System

2. The **Plan Evaluator/Selector** evaluates the results of the simulations and suggests a plan for execution. Currently, three elements are considered: 1) closeness to the goal location; 2) the total time elapsed; 3) power usage. These elements are combined into a single score using the following equation:

$$Score = W_d * dist + W_t * time + W_p * power \quad (5.8)$$

where

dist = is the distance between the rover's location and the goal location.

time = is the total virtual time elapsed between the start and end of the simulation

power = is the total power usage of the rover

W_d , W_t and W_p represent weights attached to each factor so that they can be adjusted appropriately to reflect the needs of the user. Since we want all of the above to be minimum, the score should be as low as possible.

5.2.2 Experimental Design

Our SBP method's experimental design approach to the rover problem is to vary the terrain and the rover configuration. More specifically, we vary three factors: rock

(distribution of rock sizes and frequency) of the landing site to design a non-uniform sampling distribution. By using visual information of the landing site, we can build rock distributions that are similar to the actual terrain characteristics. The placement of these smaller (less than 23cm in diameter) rocks will be random. As time permits, multiple simulations with different small rock placements will be performed. On a higher level, the percentage of rock cover can be varied to be between 6% to 19% but it is fixed at 6% for now. Thus, a simulation environment is set by the tuple (D, A, T) where:

$D = [4.0, 12.0]$ representing the unit distance of travel (in centimeters)

$A = [3.0, 7.0]$ representing the unit angle of rotation (in degrees)

T = is the terrain determined by (s, f, p) where $s = [0.05, 0.23]$ (in meters) is the diameter of the rocks, $f = 6$ is the percentage of rock coverage and p is the placement strategies of the rocks which is done randomly in the current implementation but may be based on other probability distributions using available data. Since there are infinite number of combinations of these factors, our approach is to sample D and A at certain intervals while creating T by randomly selecting the values for s . Since f and p are fixed in this prototype, we will refer to the terrain simply as $T(s)$. Because it is possible to determine f from observations of the surrounding terrain either by satellite imagery or by local camera images, it is reasonable to assume that we can fix f to be a certain value for a particular planning situation. It is also possible that we can narrow down the range of s in a similar way.

For the simulation, we use discrete time step simulation with the following algorithm (repeated for each route):

While (Goal is not reached) do

Sample sensor data

Follow route by executing action on rover

Update rover state variables

Update current clock time by ΔT

End While

Until the rover's state variables indicate it has reached the goal location, the planner continues the above loop and continues on its route. Sampling sensor data involves sensing the rocks on the terrain using the sensor model built from JPL's description. Depending on the size and location of a rock, the planner may also invoke the hazard detection sensors—indicating that the rover will have to maneuver around the obstacle. Smaller rocks (with diameter less than 23 cm) are ignored by the hazard detection sensor but is likely to cause the dead-reckoning error to increase. For the rover route planning problem, this is the distinguishable aspect of SBP which is handled differently (or sometimes not at all) by other planning methods.

The effect is captured by incorporating a dead-reckoning error model into the simulation which is modeled as a white process. A white process is a random process where the distribution of the expected value of the members of the collection of functions over all frequency components in the full range is uniform. This is an idealized concept which does serve as a very useful approximation where noise is wideband compared with the bandwidth of the system. Finally, the state variables (location x,y) are updated accordingly using control dynamics of the rover along with the clock.

5.3 Prototype Environment

For the prototype, a graphics user interface to the planner has been developed by our MOOSE (Multimodel Object-Oriented Specification Environment) group. Figure 5.5 shows the interface we have built and the routes traversed using our algorithm. Through this interface, the user is able to define various parameter values for the terrain, the rover, the experimental design, and the output analysis. A user can either

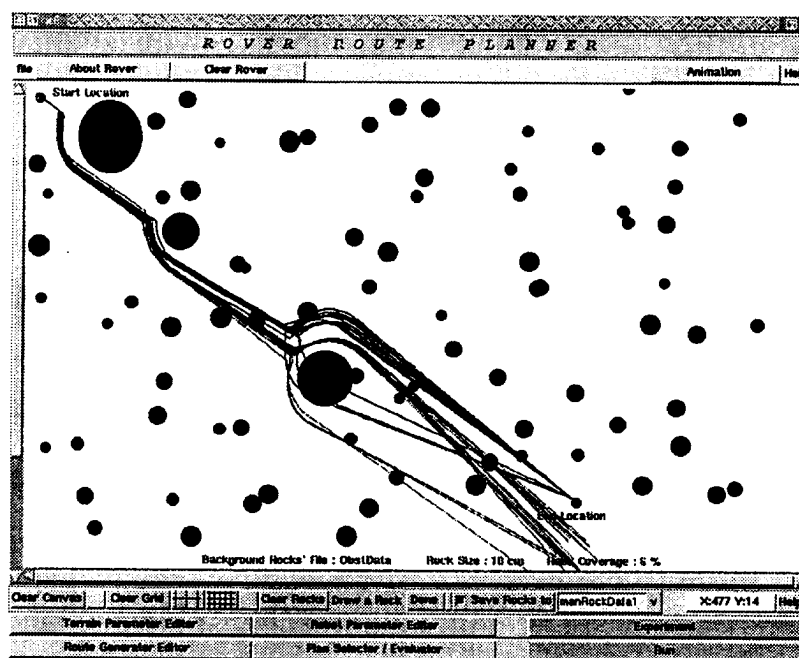


Figure 5.5. Rover Route Planning Runs

set the parameters to certain values or select certain ranges in which parameter values will be randomly sampled and used in the simulations.

The shown paths in figure 5.5 are 20 replicated simulations on a terrain where the rock sizes are 10cm in diameter or smaller and the overall area of rock coverage is 6% (i.e. 6% of surface area are covered by rocks). The start location is at (10,10) and the goal location is at (600,500). The distance to the goal is 766cm. Since the distance of the real runs tested in the Mars Rover experiment was 7.6m, our test distance is an appropriate choice. The paths that are darker and end exactly at the goal location are dead-reckoned paths (paths the rover thinks it is traversing in) and paths that are lighter and stray away from the goal location are actual paths the rover traveled.

5.4 Planning Results

In the rover experiment, a test run was considered successful if the rover's dead-reckoned position estimate implied that it had reached the goal. Based on this definition, all 20 of our rover route planning runs in figure 5.5 were successful at unit

distance of 6.5cm. This is a good match to the rate of success of the real rover test runs, which was 39 out of 40 with unit distance of 6.5cm. Figure 5.6 displays the response surface graph of the average evaluation scores obtained from 20 replications of each simulation environment $(D, A, 10)$. 10 implies that the terrain is made up of rocks whose sizes are 10 cm in diameter or smaller. Thus, for such a terrain, we can observe from figure 5.6 that the area of optimal configuration will be near the area where unit distance is 7 cm and unit angle is 3 degrees. Figure 5.8 shows the confidence interval half-widths at 95% of the mean evaluation scores produced in figure 5.6. The area where the confidence interval width is the smallest coincides with the area of near optimal configuration in figure 5.6 which means the near optimal configuration area is also quite stable.

Figures 5.7 and 5.9 are scores and confidence interval half-widths for $(D, A, 20)$. The overall shape of the response surface is somewhat similar to the one in figure 5.6. However, there is a significant difference in the area where score is minimum. In figure 5.7, the area of close to optimal configuration is when unit distance is set to 7.3 cm and unit angle is set to 3 deg whereas in figure 5.6, the optimal area is where unit distance is near 3.6 cm and unit angle is near 4.8 deg.

As the difference of the optimal configuration point is significant depending on the terrain, having the ability to search for the near optimal configuration given the current terrain environment will be crucial to the overall success of the mission.

In this chapter, we were able to observe how simulation-based planning can play a role in predicting near-optimal configurations for terrain traversal in a given environment. This exhibits the potential for on-line adaptive planning. One can imagine that, once on Mars, the rover can first gather information about the environment as much as possible and then using this current, local information, the rover can perform simulation-based planning to figure out which configuration (unit distance and

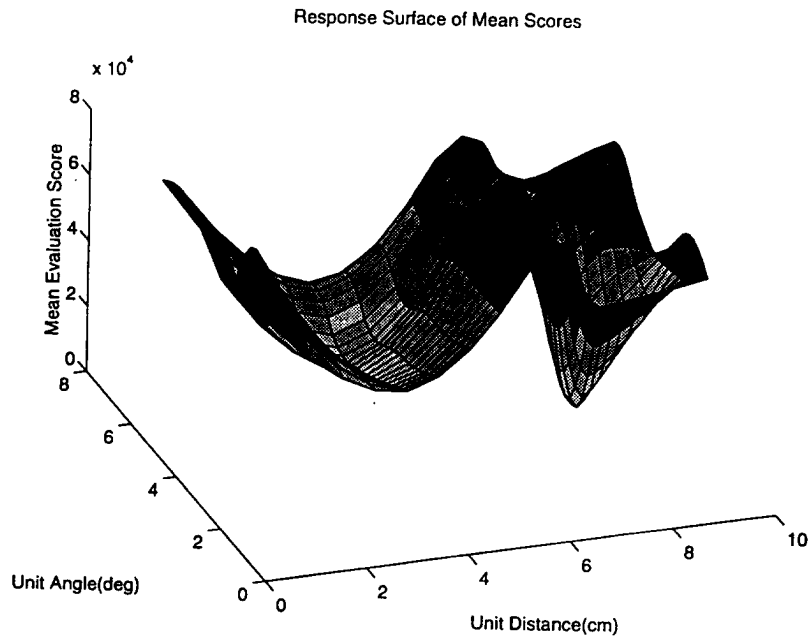


Figure 5.6. Response Surface of Mean Evaluation Scores (Rock Size ≤ 10 cm)

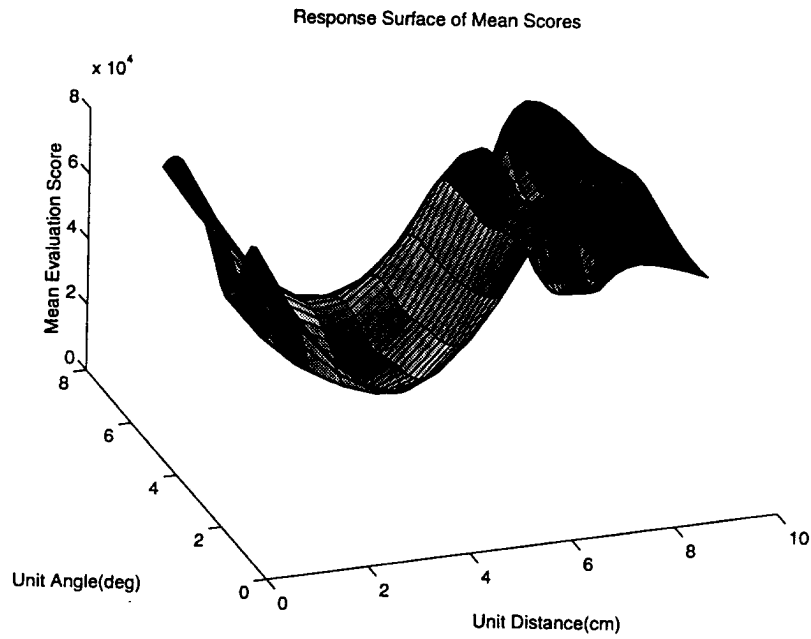


Figure 5.7. Response Surface of Mean Evaluation Scores (Rock Size ≤ 20 cm)

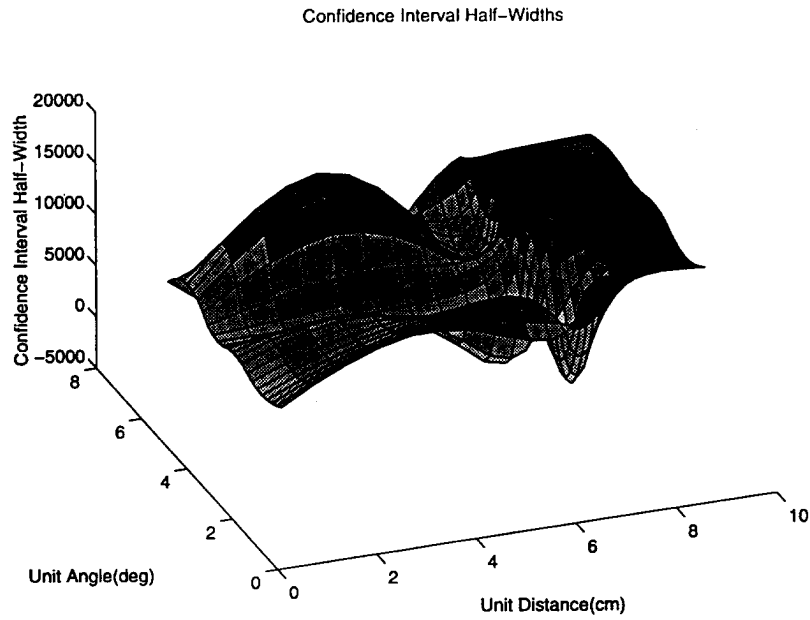


Figure 5.8. Response Surface of Confidence Interval Half-Width at 95% ($RockSize \leq 10cm$)

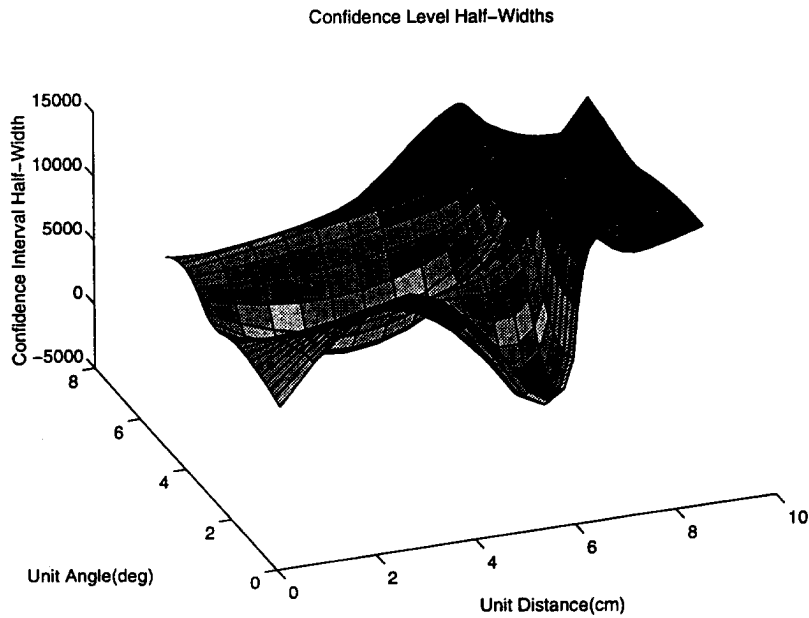


Figure 5.9. Response Surface of Confidence Interval Half-Width at 95% ($RockSize \leq 20cm$)

angle, traversal algorithm etc.) is likely to produce the outcome that will be closest to the goal.

In addition to unit distance, unit angle and terrain, which we have varied in the experiment, we can also simulate other factors such as the time of day—to predict the effects of light on the energy, for instance.

CHAPTER 6 NONDETERMINISTIC ADVERSARIAL ROUTE PLANNING

6.1 Air Force Mission Planning

Similar to the CGF mission planning problem discussed in chapter 4, the Air Force Mission Planning is a good candidate for SBP since it is adversarial and also nondeterministic. To illustrate, here is an example air combat scenario in Figure 6.1.

This figure defines a scenario with dynamically moving objects. The mission of the blue force fighter aircraft is to pave a path for the bomber to disable the factory. There are three radar locations, (R1,R2,R3), each with different effective detection ranges. On the surface, the problem of guiding the blue force around the radar coverage, and toward the factory, seems like a simple problem in computational geometry. In actuality, this is the way in which most route planning operates. A rule might be formed “To locate a path, avoid radar fields and storm fronts” to aid in the decision making. Consider the following available knowledge at some point during the mission:

1. *Uncertain location and range:* Radar locations R2 and R3 are permanent fixtures and have been identified by satellite imagery. However, R2 may have been upgraded to have a wider range. Moreover, R1 may not exist. A land-based scout report suggests that it may.
2. *Uncertain enemy mission:* Red force aircraft are known to fly a routine reconnaissance route toward the north. Is the red force on a routine mission, and therefore out of the way, or has the enemy indicated the blue force strength, providing the red force with an intercept mission?

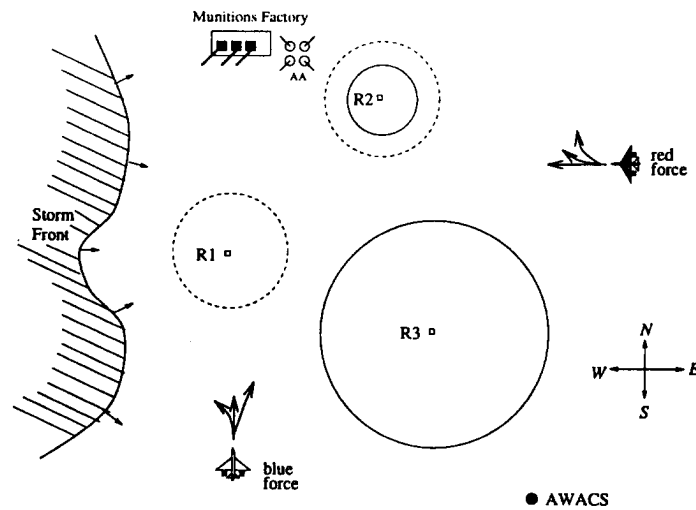


Figure 6.1. Air combat scenario with dynamics and uncertainty

3. *Uncertain force strength:* An AWACS monitoring plane has just identified a blue force (as denoted just south of the AWACS) but it is unclear as to the number and type of aircraft. The factory has four anti-aircraft (AA) guns but the intelligence reports are not complete; the range and accuracy of the guns is largely unknown.
4. *Weather conditions:* Weather stations report a storm front to the West, moving East, but the cloud cover situation is unclear; only a fuzzy estimate can be made of the height and severity of the storm.

These types of uncertainties may well be the rule rather than the exception.

As the example shows, large part of the mission is dependent on which particular route or air corridor you will fly through and it basically becomes a route planning problem with many uncertainties on the way. Of course, some higher-level expert system can be used and should be used as a meta reasoning system on top of the SBP planner to provide guidelines as to-given the current situation-what general heuristics we are to use in generating alternative air corridors for simulation.

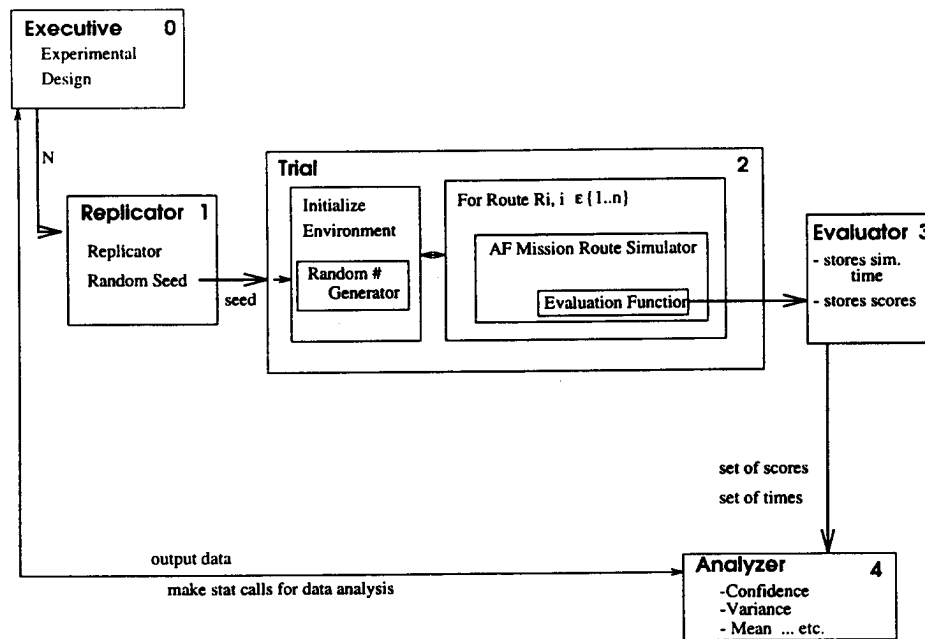


Figure 6.2. Top Level Architecture of Air Force Route Planner

In this chapter, we present the design, implementation and some preliminary results of the prototype built for the air force mission planning domain.

6.2 Planner Architecture

Figure 6.2 shows the specific top level architecture for the Air Force Route Planner. The generic Route Simulation Module in Figure 2.1 is now instantiated to the Air Force (AF) Mission Route Simulator, which is described in the following sections.

6.2.1 Multimodel of AF Mission Route Simulator

In this section, we describe the multimodel for the Air Force Route Simulator which resides inside the **Trial** block. First, the class hierarchy of our Air Force multimodel appears in figure 6.3. Assuming that a set of alternate routes and environment data are given, the following models are simulated and evaluated for each route R_i . The simulation process is replicated and its output results are accumulated and then analyzed by the **Analyzer**.

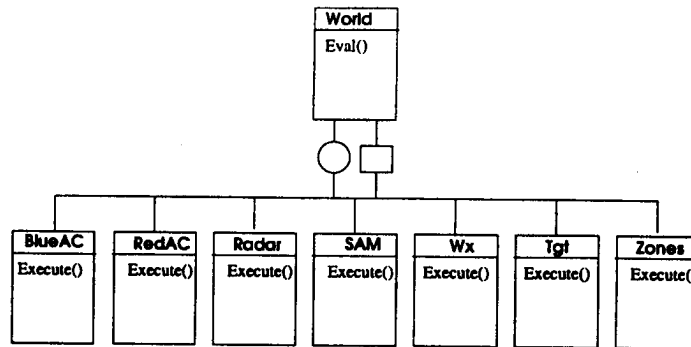


Figure 6.3. Class hierarchy of Air Force Model

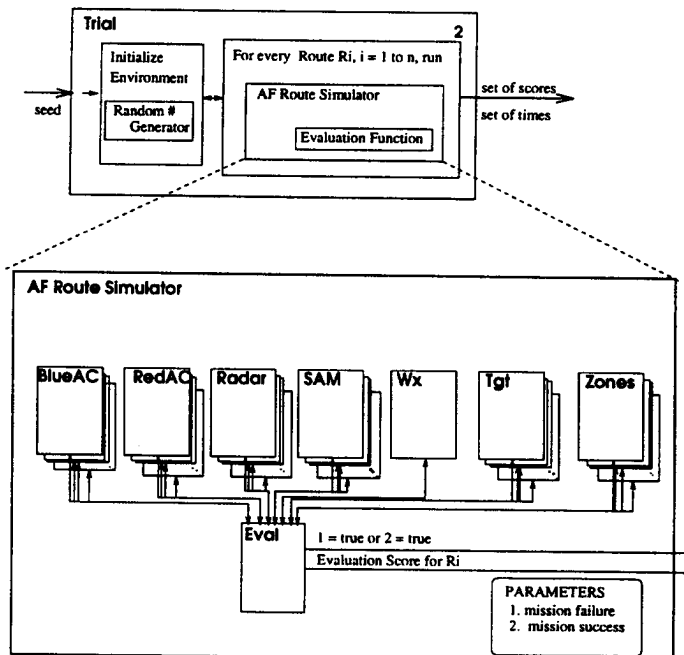


Figure 6.4. General Simulator Module

Figure 6.4 shows a general layout of all the objects that are currently implemented in the simulator module. We assume there are seven types of PHYSICAL objects: BlueAC, RedAC, Radar, SAM, Wx, Tgt, Zones and one ABSTRACT object called Eval. The class BlueAC stands for Blue Aircraft and RedAC similarly stands for Red Aircraft. Radar represents a ground radar site. SAM represents a ground SAM missile site. Wx represents the weather. Tgt represents the target that needs to be destroyed. In the current prototype the target is the headquarters for the red force. Zones represent the area of defense zone. For the zones, we assume that there are a set of radars strategically located inside the zone such that when an enemy aircraft (BlueAC) flies inside the zone, it is detected. Any number of each object type may exist depending on the specific mission situation. If the user defines the environment to have 3 radars then, by the nature of the object-oriented paradigm, 3 Radar instances of the Radar class are automatically created. Having multiple weather objects is unimaginable but a weather object made up of other sub-weather objects such as wind, cloud and sun is certainly plausible. During a typical simulation loop, every object updates its local state and/or perform action(s) which in turn may affect other objects in the following time slice. The last object to be called within a simulation loop is Eval. We describe the role of each object models in more detail below.

Eval

Figure 6.5 shows a functional block of Eval which contains the various check functions that check the status for each object class. Notice that if there were n objects of class Radar, then the Check Radar function will check all n radar objects.

1. Although it is not explicitly shown in the figure, Eval is responsible for maintaining a consistent and aggregated “current state” of the world. In other words,

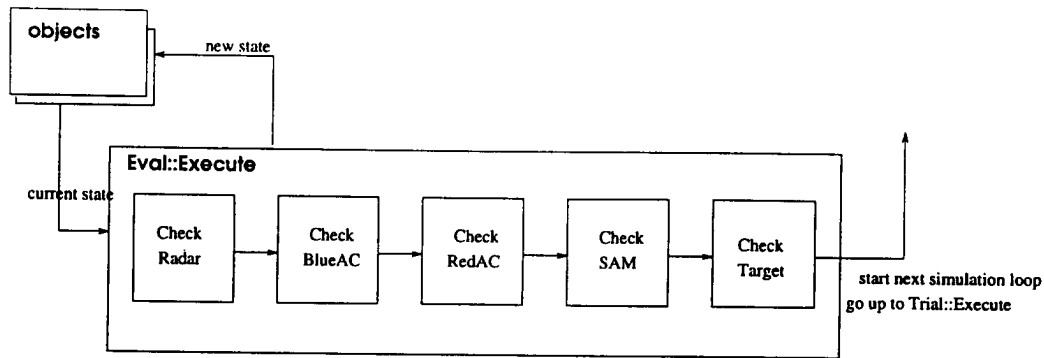


Figure 6.5. Eval(Evaluation) model

using the notation defined in section 2.2.1, Eval is responsible for maintaining $Q(t)$.

2. Eval decides the outcome of interactions between objects. In a way, it plays the role of God. Every time Eval is executed, it looks for certain condition of each object such as Missile_Fired or Engagement. If object O_i has fired a missile targeting object O_k , for example, the outcome of the event (missed_target, destroyed_target) is determined by Eval. Our reason for having Eval determine such events is because we wanted to maintain symmetry and self containment among objects. It is not symmetrical to have either any of the two objects decide the outcome of an event that is outside of their control. Determining the outcome can be as simple as sampling from a given probability distribution or simulating a lower level model which would contain a physical model of the missile.
3. Eval evaluates each situation for every time slice and maintains a score that represents the goodness of the plan. The evaluation score is constantly affected by events during the simulation. In the prototype, we have assigned arbitrary values to each event; a positive(good) event has a positive value and a negative(bad) event has a negative value. The following explains our initial set of

events which have already been implemented. We list the set of implemented events in association with each object type that is interacting with the BlueAC.

RedAC

- **NO_KILL** : neither the RedAC nor the BlueAC has been destroyed. The score is unaffected by this event.
- **RED_KILLS_BLUE** : RedAC has destroyed BlueAC. The value here is set the maximum negative value since it implies that the mission has failed.
- **BLUE_KILLS_RED** : BlueAC has destroyed RedAC. The value is positive but is not set to the maximum value since it does not immediately imply that the mission is a success.
- **RED_AND_BLUE_KILL_EACH_OTHER** : BlueAC and RedAC has destroyed each other. The score is first increased by a positive value since a RedAC was destroyed but is decreased by a larger negative value because the mission has ended in a failure.

SAM

- **NO_KILL** : Neither the SAM nor the BlueAC has destroyed each other. The score is unaffected by this event.
- **RED_KILLS_BLUE** : red SAM missile was fired and has destroyed BlueAC. The score is decreased by the maximum amount as in the case of RedAC.
- **BLUE_KILLS_RED** : BlueAC has destroyed the red SAM missile site. The score must be increased by a positive value but should not be increased by the maximum amount since it does not immediately imply that the mission is a success.

- RED_AND_BLUE_KILL_EACH_OTHER : BlueAC and red SAM has destroyed each other. The score is first increased by a positive value since a SAM was destroyed but is ultimately decreased by a larger negative value because the mission is a failure.

Tgt

- TGT_DESTROYED : red Target was destroyed by the BlueAC which implies the mission was a success. The maximum positive value is added to the score to represent the attractiveness of this outcome.

Radar

- BLUE_DETECTED_BY_RADAR : BlueAC has been detected by a radar. A small amount of the score is decreased every time the BlueAC is detected. Unlike previous events, which occur only once, this event can reoccur for each time step as long as the condition holds true. The logic behind this is that the longer the BlueAC is exposed to the radar, the higher the danger is of being attacked and thus the score should reflect this property. More accurate results will be obtained if the radar is modeled at a lower level where other red aircrafts are contacted to intercept and possibly destroy BlueAC.

Finally, when BlueAC has successfully destroyed the target, the remaining fuel level is calculated and added to the score in order to reward alternatives that have used less fuel. Thus, in the current prototype the evaluation function is based on the common sense judgement where the remaining fuel level, the level of safety and the level of success (in terms of destroying the target) is combined to represent the goodness of a route plan. As better expert knowledge becomes available from an

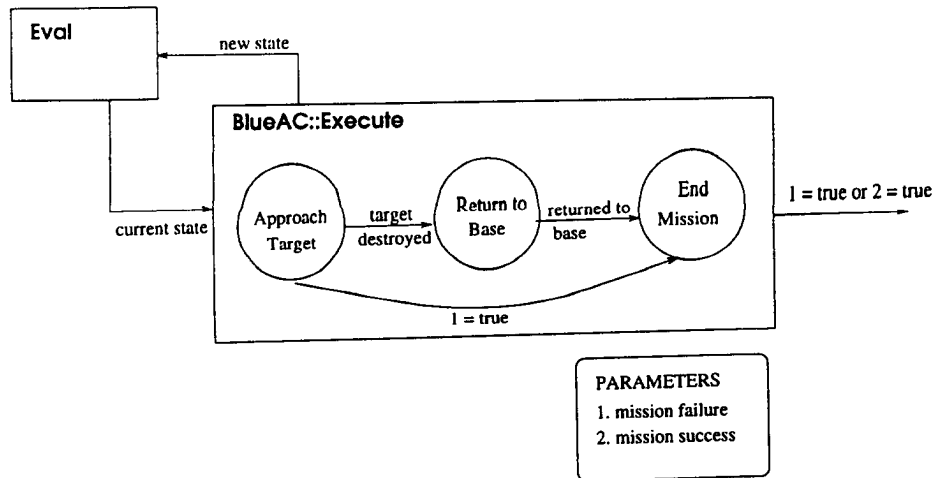


Figure 6.6. Blue Aircraft (BlueAC) object model

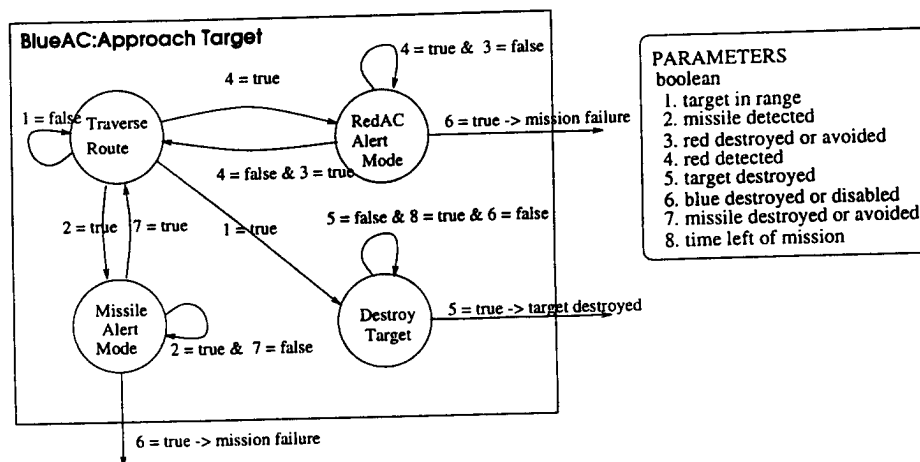


Figure 6.7. Interdiction FSA for BlueAC

SME(Subject Matter Expert), we expect that finding a good evaluation function will play a major role in making SBP a viable methodology in military planning.

BlueAC

The objects BlueAC and RedAC are Dynamic Objects (as defined in section 2.2.1). The top level model of the BlueAC object is shown in Figure 6.6. In our prototype we have uniformly named the main top level method of each class as **Execute**. At the highest level of the BlueAC object, for an Interdiction mission, it is modeled as an FSA with three phases: Approach Target, Return to Base and End Mission.

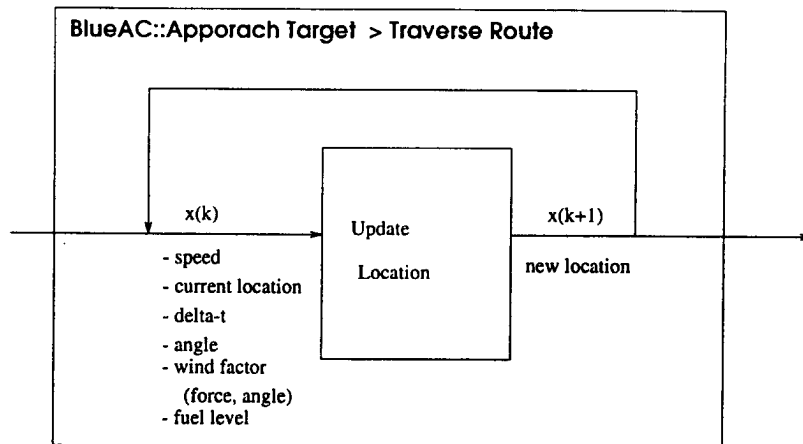


Figure 6.8. Traverse Route function for BlueAC::Approach Target

Figure 6.7 shows the refinement FSA for Approach Target phase appearing in figure 6.6. In normal circumstances, BlueAC traverses the given route. However, whenever an enemy aircraft or a missile is detected, it must switch into an alert mode. Once the danger is no longer present, BlueAC can go back to the Traverse Route phase. Finally, when the target is within range, the FSA transitions to the Destroy Target phase. After the target or the BlueAC has been destroyed, the Approach Target FSA exits with the appropriate condition. This exiting condition is used at the higher level to transition to the appropriate next state in the FSA which is Return to Base (Figure 6.6).

Going another level down from the Traverse Route phase, figure 6.8 illustrates the functional block model for updating the location while traversing the route. It takes as input, the aircraft's speed, current location, angle, fuel level and wind factor. Except for the wind information, all the other information are maintained in the vector $\mathbf{x}(k)$ which represents the state of BlueAC at time k (current state). The UpdateLoc function produces the next state $\mathbf{x}(k + 1)$. We can express the state vector $\mathbf{x}(k)$ as follows :

$$\mathbf{x}(k) = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \\ s(k) \\ f(k) \end{bmatrix} \quad (6.1)$$

where $x(k)$, $y(k)$ and $\theta(k)$ describe the current location of the aircraft. $s(k)$ denotes the current speed and $f(k)$ denotes the fuel level at time k . Currently, the speed is set to a constant. It is certainly possible, however, to explore different constant speeds or varying speeds during the simulation to experiment other possibilities. Based on the current speed, we calculate the unit distance $T(k)$ for Δt using

$$T(k) = s(k) * \Delta t * scale_factor \quad (6.2)$$

The term *scale_factor* is used to either scale down or up the simulation granularity of the terrain grid. Finally, the fuel level $f(k)$ is modeled by equation 6.3.

$$\begin{aligned} f(k + \Delta t) = & f(k) - (s(k) * \Delta t * scale_factor) * fuel_consumption_rate \\ & - ws(k) * \cos(|\theta(k) - w\theta(k)|) \end{aligned} \quad (6.3)$$

where $ws(k)$ is the wind speed and $w\theta(k)$ is the direction of the wind at time k . The *fuel_consumption_rate* refers to the amount of fuel that is consumed per basic unit of distance(e.g. per mile or per kilometer). In addition to the current speed, the wind can have an effect on the fuel consumption depending on the angular difference. Based on this difference and the wind speed, the fuel consumption will either increase or decrease. BlueAC's fuel level is updated every Δt .

In the current prototype, only the Traverse Route and Destroy Target behavior have been fully implemented. The RedAC Alert Mode can, however, be further refined as another FSA as shown in figure 6.9. The Missile Alert Mode will also take on a similar FSA model as in figure 6.9. However, if the FSA in figure 6.9 were to be

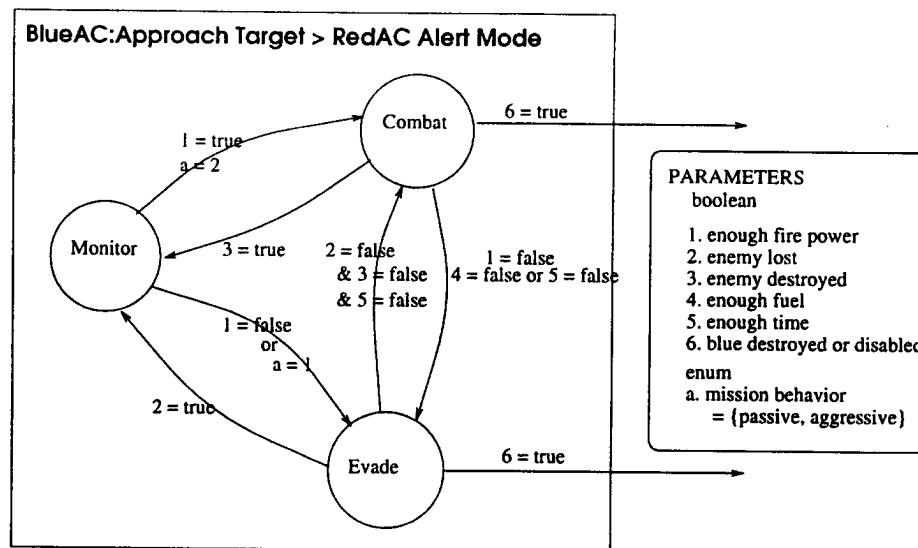


Figure 6.9. Red Aircraft (RedAC) model for BlueAC object

further refined, it will have a different refinement model since different tactics and physical models will apply in monitoring, combatting and evading a missile or an aircraft.

The return trip of the BlueAC to the base is not currently implemented in the prototype. However, this can easily be taken into account. The simplest way is to add a function such as the one shown in figure 6.10 that calculates whether there's enough fuel for a return trip. And if so, reward the alternative highly and if not, penalize it severely. If the return flight is likely to involve significant risks and the safe return of the BlueAC is of major concern, the return flight should be simulated and scored in much the same way as it was done for the Approach Target Phase of the mission.

The Red Aircraft model shown in figure 6.11 have a special mission. They are assigned to certain defense zones where their duty is to defend their zones from any intrusion by an enemy aircraft. When the models are set, the user describes the starting state of RedAC: its location, fuel level, current speed and any zone assignment. Once the simulation starts, the RedAC starts from its initial location

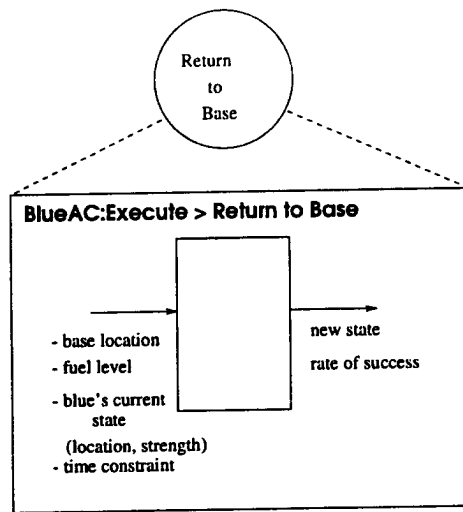


Figure 6.10. Return to Base functional block for BlueAC

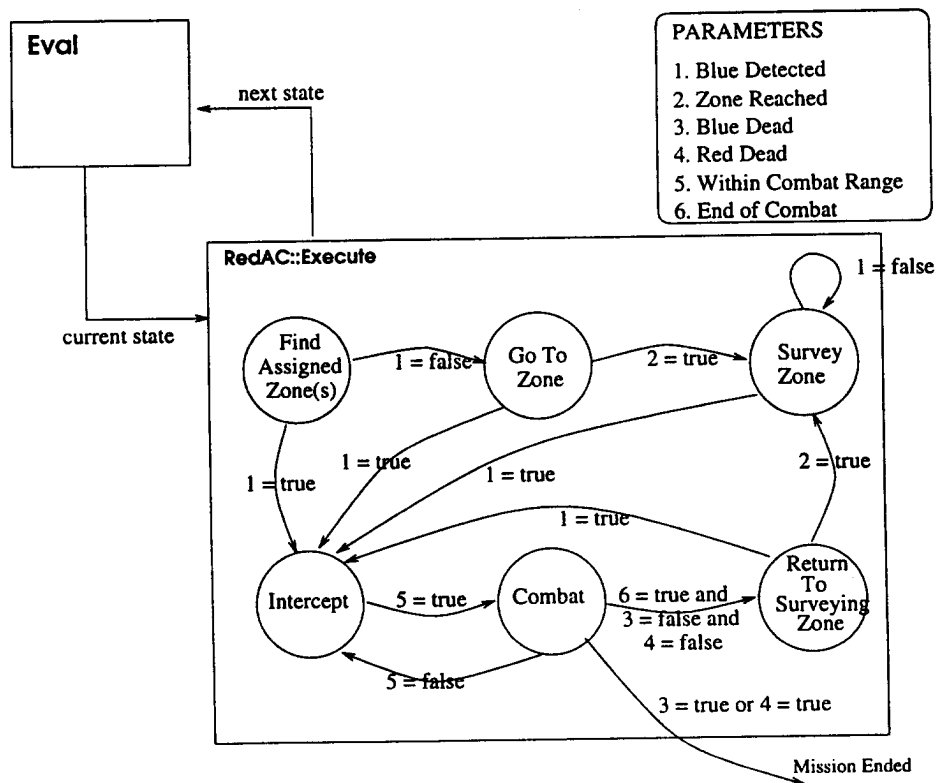


Figure 6.11. RedAC (Red Aircraft) model

and flies to its assigned zone. After reaching the zone(s), it goes into a surveying mode—flying back and forth following the zone from one end to another. If RedAC is assigned to more than one zone, it is reasonable to assume that it is assigned to consecutive zones. So, if it is assigned to zone 1 and 2, for example, it will first fly to the northwest corner of zone 1 and then once it has reached zone 1, it will fly towards the northeast end of zone 2. When it has reached the east end of zone 2, it will turn around and start a flight back towards the west and so on.

While performing the survey mission, whenever a BlueAC crosses an assigned zone, the assigned RedAC will go into an Intercept mode and pursue the BlueAC and attempt to destroy it. Once the two aircrafts come within range, Combat may occur. RedACs having other types of specialized missions should also be modeled and simulated in the future.

The next two objects belong to the Static Objects group as defined in section 2.2.1, since we are assuming both the missile and radar site are immobile. The SAM missile has three ranges: Track range, Missile range and Arm range. The track range represents a range where an aircraft can be detected and tracked. The missiles can actually be guided towards the target in preparation for launch within the Missile range. Finally, if an aircraft comes within Arm range we assume the SAM missiles will be automatically launched to destroy the target. We have employed a generic SAM missile site model here but a more specific and sophisticated model can be used if necessary.

We model ground radar sites as a simple one level FSA as shown in figure 6.13. When BlueAC is detected by a radar, a possible action by the radar station would be to contact other RedACs to intercept and destroy the blue aircraft. Currently, we capture this effect by having decrementing the score by a certain amount for every

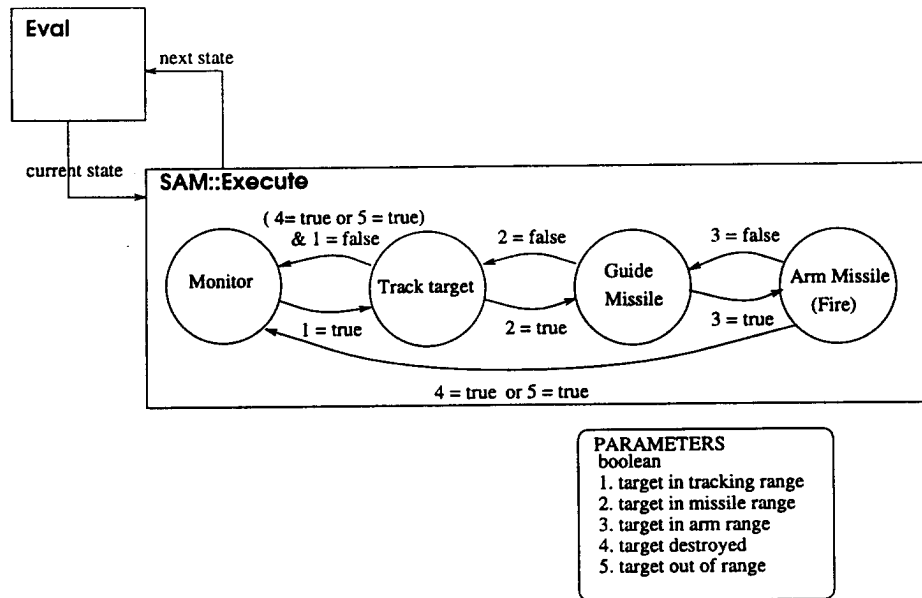


Figure 6.12. Model of SAM missile site

time slice BlueAC is detected by a radar. Thus, the longer the BlueAC is exposed to the enemy radar, the lower the score will be.

Finally, the weather object **Wx** is modeled as having the wind factor. The wind has a constant speed and a constant direction.

The model presented here is by no means complete nor has it been validated by an SME. Since building sophisticated and realistic models is not the issue in our current research, simple yet sensible models were built to prove our SBP approach.

6.3 Demonstration Mission Type

Our demonstration air mission is interdiction. Interdiction mission is a typical air mission where the purpose is to destroy, delay, or disrupt existing enemy surface forces while they are far enough from friendly surface forces that detailed coordination of aerospace and surface activities is not needed. The objective of interdiction entails the execution of carefully conceived, comprehensive plan designed to isolate an area and to stop all support from reaching the area of conflict.

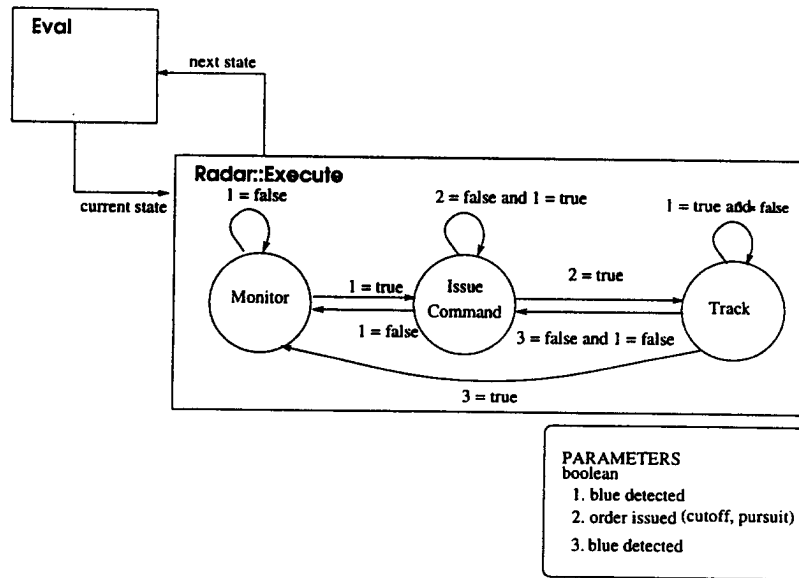


Figure 6.13. Radar model

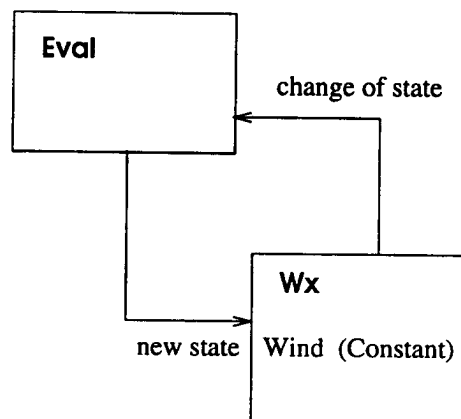


Figure 6.14. Weather model

Therefore, the task of the attack aircraft can be defined as “hitting the target swiftly and accurately with whatever munitions are carried, and return to base safely” [39]. To achieve this task, the enemy defense must be penetrated. However, difficulties arise because methods of penetration can vary according to the strength and sophistication of the enemy’s detection, reporting, command and control network, and how much intelligence is available about its capabilities. A balance between fuel and munitions in determining the load to be carried is also important in mission planning. Considering these uncertainties and constraints, selecting the best route is a very difficult task.

6.4 An Air Interdiction Scenario

As one of the applications of the SBP, we have chosen a typical air interdiction scenario, and developed its Simulation Based Planner (C++) and GUI (Tk/Tcl) under our Multimodeling Object-Oriented Simulation Environment (MOOSE). In order to show the usefulness of the SBP approach, consider an air interdiction scenario in Figure 6.15. This Figure defines a scenario with dynamically moving objects. The mission of the blue force aircraft is to destroy a red force munitions factory. There are three Radars ($R1$, $R2$, $R3$) and two Surface-to-Air Missile (SAM) sites($S1$, $S2$), each with different effective detection ranges. Two red force aircrafts ($A1$, $A2$) are located in air defense zones Zone2 and Zone3 respectively, while one red force aircraft ($A3$) is located outside of the air defense zones. At a first glance, the problem of guiding the blue force around the radar, SAM and air defense zone coverages, and toward the factory seems like a simple problem in computational geometry. In fact, this is the manner in which most route planning is done. A typical rule might be formed “To locate a path, avoid radar and SAM fields, and avoid fighting against enemy fighters.” However such a rule based reasoning becomes more onerous when uncertainty and dynamics are present.

To see what kind of uncertainty and dynamics are involved, consider the following available information at some point during the mission.

- Uncertain location and range : Radar $R1$ and $R2$ have been identified as permanent fixtures, but an intelligence report suggests that $R3$ may have mobility. All the ranges (target, missile, arm range) of SAM site $S1$ is well known, but only the arm range of $S2$ is known and it has been reported to have a better guidance system including swift mobility making its location uncertain.
- Uncertain enemy mission : red force aircraft $A1$ and $A2$ are known to be on a Combat Air Patrol (CAP) mission, since they are always detected around zone2 and zone3. But $A3$'s mission type is unknown.

For each simulation trial, the uncertainty of $S2$ is handled by first sampling a random location for $S2$ within the boundaries of the circle drawn around $S2$ in figure 6.15. Taking this location as the center point of the SAM site, a boundary circle is drawn representing the arm range of the SAM site. The uncertainty of the radar $R3$ is handled in a similar manner. The location is first determined by sampling the point within the uncertainty circle drawn by the user. Using the sampled point as the center point of the radar, a boundary circle is drawn representing the detection range.

For the objects in our air force mission planning domain, we can categorize the uncertainty into several types:

1. uncertainty of existence: the object may or may not even exist.
2. uncertainty of location: an area of uncertainty of the object's location is available but it is not certain of the exact location of the object.
3. uncertainty of range: the exact detection range or firing range is not known.

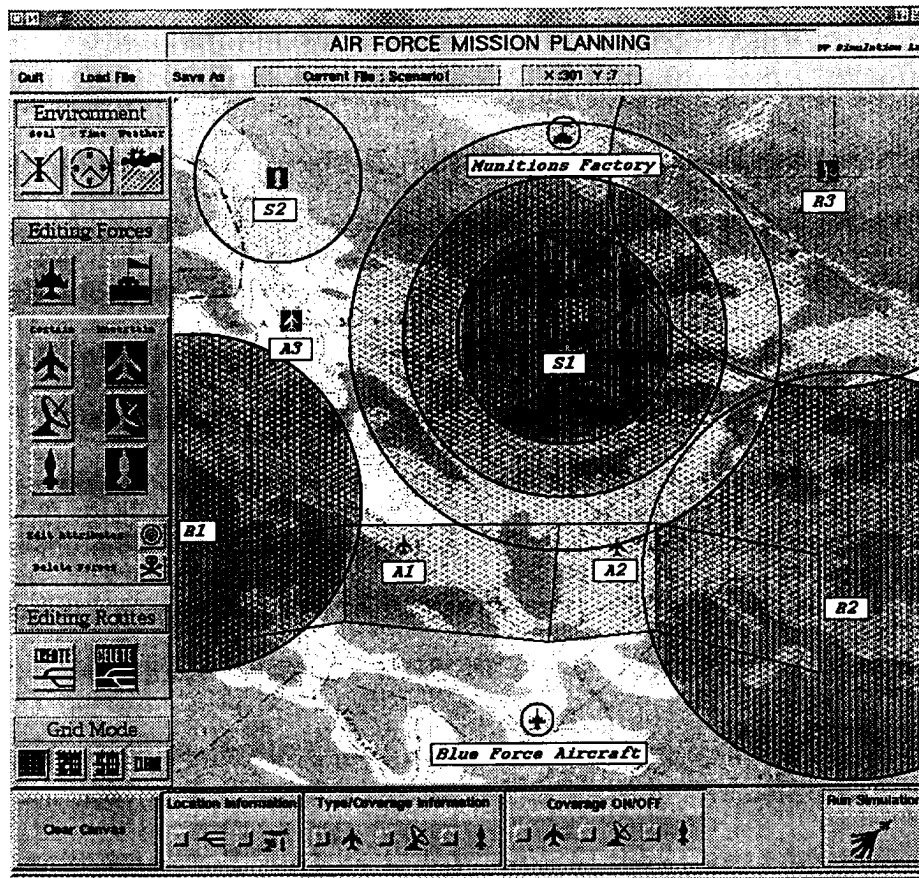


Figure 6.15. A Typical Air Interdiction Scenario

4. uncertainty of mission: the exact mission type of an object is unknown.
5. uncertainty of fire power: the destruction capability of the object is uncertain.

For our current prototype, we have concentrated mainly on location and range uncertainties.

6.5 Planning Results

Figure 6.16 shows two possible routes (*Route1*, *Route2*) under the environment defined in figure 6.15. The goal of blue force aircraft is to destroy the red force munitions factory while satisfying 3 constraints: time or fuel level, safety, and destruction of the target. Given the possible routes, the role of the simulation-based planner is to choose the best route minimizing time and fuel consumption, and maximizing safety

and target destruction. In figure 6.16, *Route1* is more attractive than *Route2* if we value mission time above all others, but seems less safe since it is vulnerable to an attack by red fighter *A1*. *Route2* might be considered more safe and achieve higher target destruction than *Route1* by avoiding the attack from fighter *A1* and SAM site *S1*. However, it will be detected by radar *R2*, increasing the probability of losing blue force aircraft or damage to blue force aircraft. Moreover, there is a big chance of being detected by radar *R3* even though its location is uncertain. The table at the lower left of figure 6.16 shows the result of the simulation. We display the mean score and the confidence interval half width of each mean at a 90% confidence level. As can be expected, *Route2* is more successful since it avoids direct attacks from the highly destructive enemy fighter and the SAM site (mean score of *Route2*: 69, mean score of *Route1*: -54).

If we delete *Route1* and consider another route based on the result of the previous situation, we have two routes that we want to analyze. Figure 6.17 illustrates these two candidates. *Route3* was chosen to avoid direct attack from *A1*, but for a short time period it will be detected by *R1*. *Route3* also takes the blue aircraft into the track range of *S1*, but not into its arm or missile range. Being detected in the track range of *S1* is not very dangerous since only tracking functions may be performed by *S1*. Overall, we expect the success rate of route 3 to depend largely on the result of the samplings for uncertainty factors: specifically, the location and guidance capability of SAM *S2* and the mission type of *A3*. If the powerful guided system of SAM is sampled close to this route, or *A3* has an intercept capability, then the chance of success will be very small. Otherwise, the chance of mission success will be very good. These nondeterministic and stochastic characteristics are resolved by multiple simulations using different samplings of the uncertainty factors. The confidence interval of the mean score of *Route3* is wide in comparison to that of *Route2* due to the reason

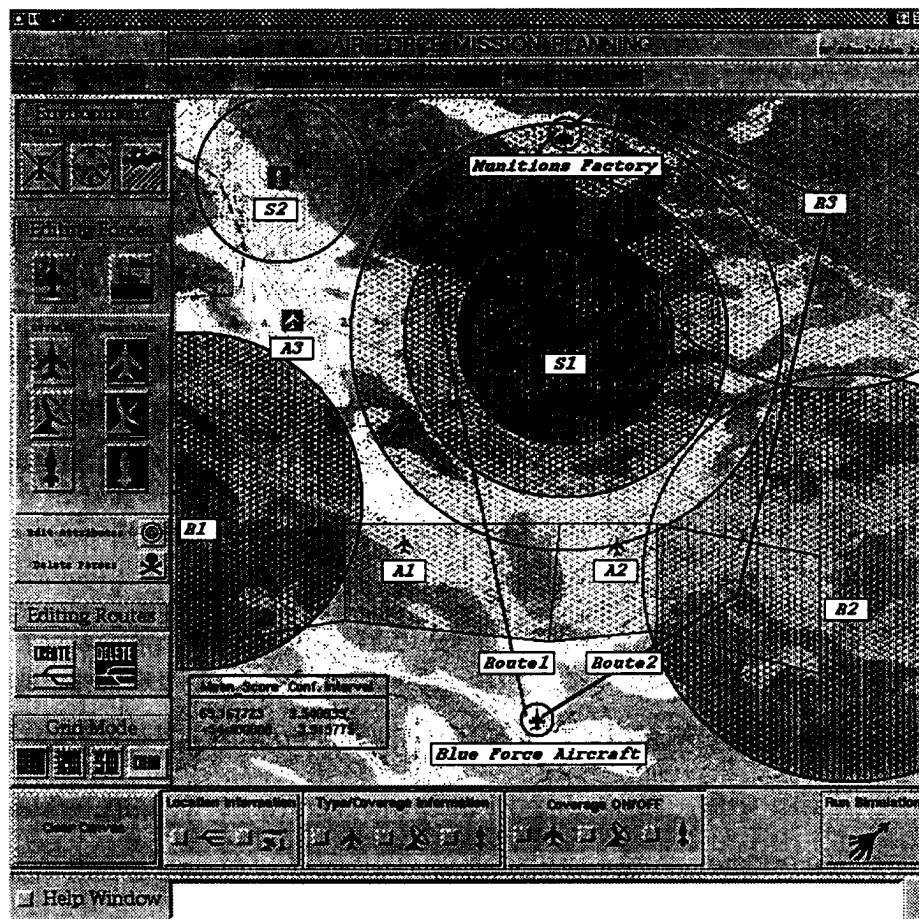


Figure 6.16. Two possible routes

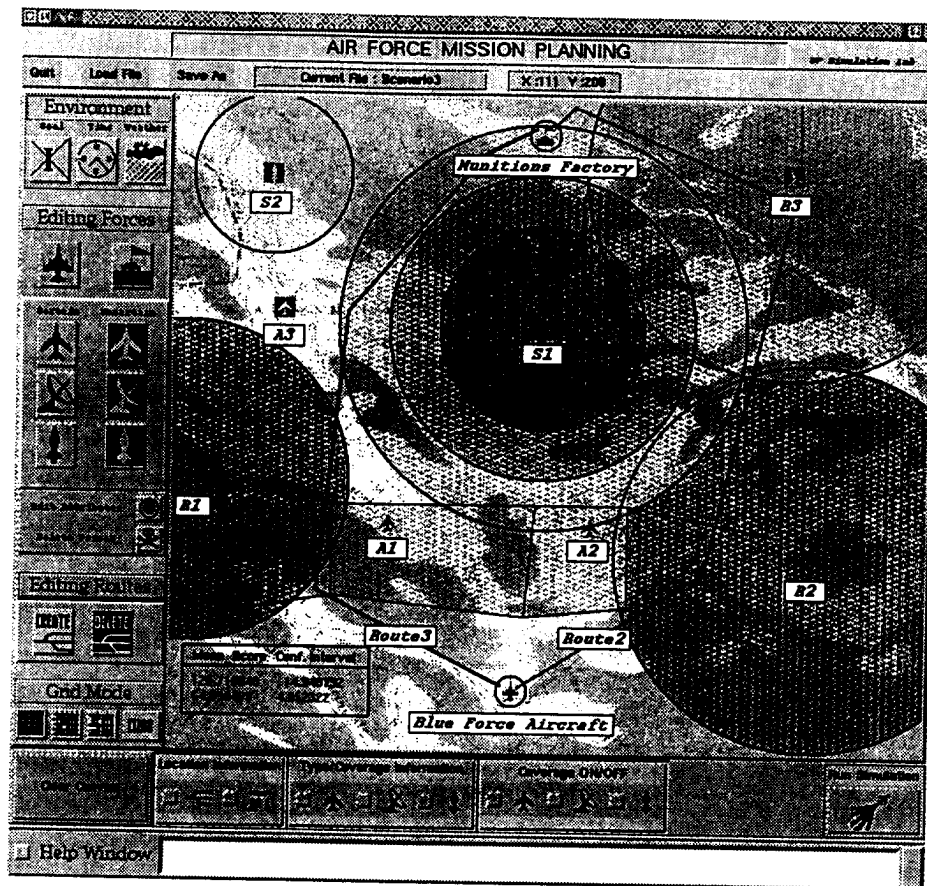


Figure 6.17. Deleting Route1 and inserting Route3

previously discussed; however, the overall mean score is better than that of *Route2* because of the small chance of being detected by *S2* or intercepted by *A3*.

We can now delete *Route2* and insert another route, *Route4*, which is carefully chosen to minimize the amount of time that a blue force aircraft falls within the detection ranges of *R2* and *R3* as in figure 6.18. The result of the SBP shows almost the same mean score for *Route3* and *Route4* (*Route3* : 110.36, *Route4* : 103.08) with *Route3* being slightly better¹. Intuitively, *Route 4* seems like a better route since it only involves radar sites whereas *Route 3* has a SAM site *S2*, although its location may be uncertain. With simulation-based planning, however, we discover that *Route 3* is a slightly better. But depending on our objective, we may select *Route4* as the

¹The goal is to maximize the mean score for determining the better plan.

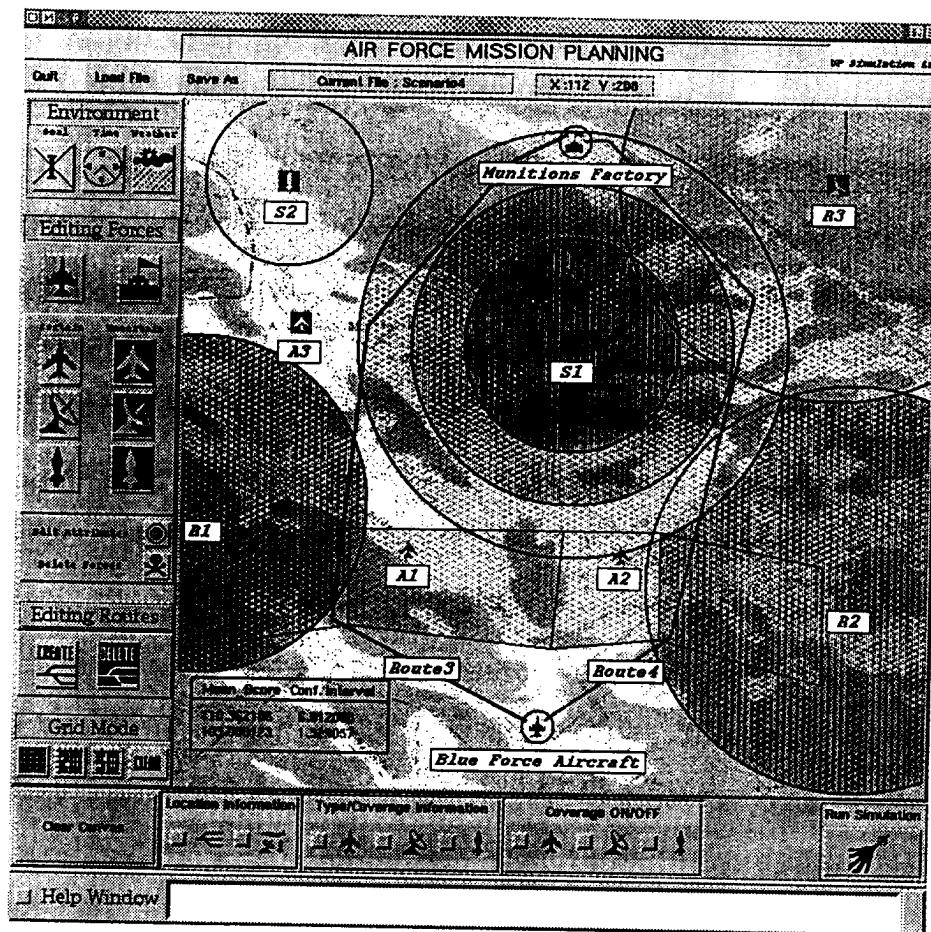


Figure 6.18. Deleting Route2 and inserting Route4

best overall route based on its narrower confidence interval (*Route4* : 1.3, *Route3* : 6.0).

Next, in order to reduce the total number of replications in the simulation, we compare two different methods of output analysis. These two methods are discussed in detail in Chapter 2. The first one, which we refer to as the “iterative” method, attempts to quantify significant pairwise differences among the k means within a given confidence interval α . We call it “iterative” because of the fact that the algorithm iterates—performing for every iteration, a set number of b replications and analyzing data to see if there are any significant differences. Whenever a route is found who is significantly worse than all the other routes, it is eliminated. The iteration continues

until only two routes remain and a difference exists between the two of them. Note that since we have 4 routes in our experiment, each confidence level for the pairwise differences must be made at $1 - \alpha/6$ ². The second method, which we refer to as the “non-iterative” method (referred to as “Selecting best of k systems” in section 2.2.3), is a method that avoids making unnecessary number of replications to resolve what may be an unimportant difference. When two alternatives are actually very close together, in terms of their goodness, we might not care if we erroneously choose one system (the one that may be slightly worse) over another (the one that is slightly better). Thus, given a “correct selection” probability P^* and the “indifference” amount d^* , the method calculates how many more replications are necessary in order to make a selection—a selection where with the probability at least P^* , the expected score of the selected alternative will be no smaller than by a margin of d^* . In the following experiment, we have chosen $P^* = 0.95$ and $d^* = 13$. A smaller d^* will produce more accurate results but with many more replications.

In addition to the two routes that appear in figure 6.18, we add two more routes are added to test how much the number of replications reduce and also if the identical selection is made. The routes are shown in Figure 6.19 and are renumbered. *Route 2* and *Route 0* represent two alternatives that are very close together and is likely to require many number of replications to quantify a significant difference between them. As expected, Routes 0 and 2 do exhibit similar responses as shown in the following plots. Figures 6.20 and 6.21 are results with just 3 routes: 0, 1 and 2. Figure 6.20 shows the mean score change of the routes using the simple iterative method. After the first 20 replications, the planner decides that route 1 can be eliminated since its scores are in general significantly lower than the other two routes. It then performs 120 replications for both routes 0 and 2 before it decides that there is

²Refer to section 2.2.3 for detailed explanation

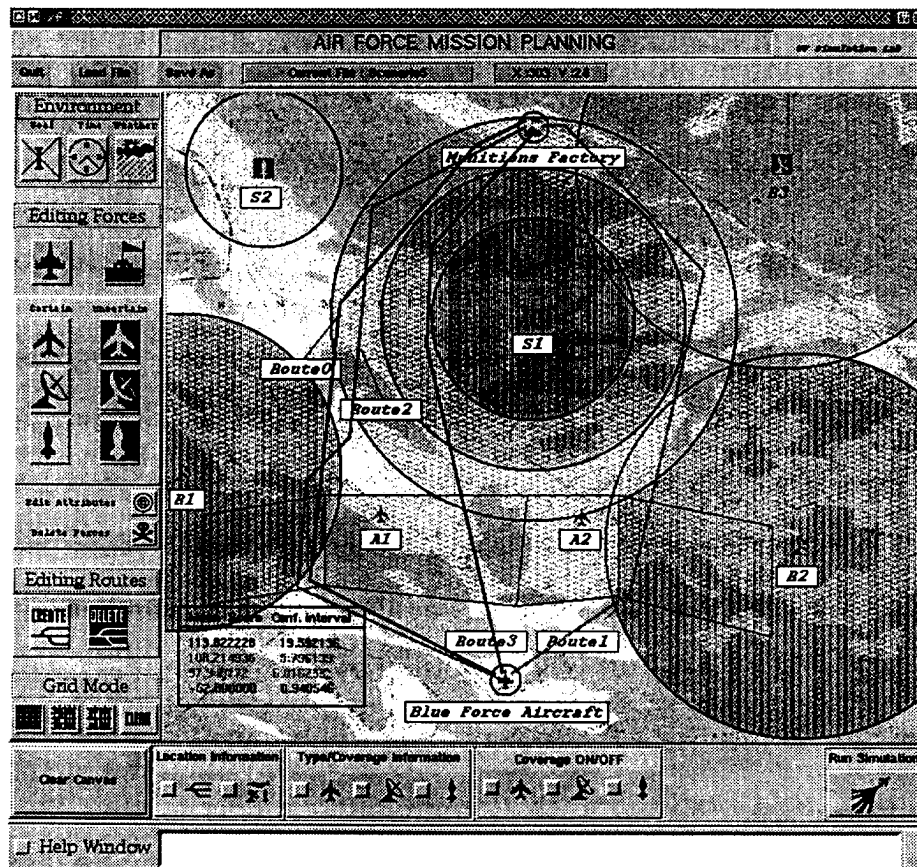


Figure 6.19. Inserting two new routes

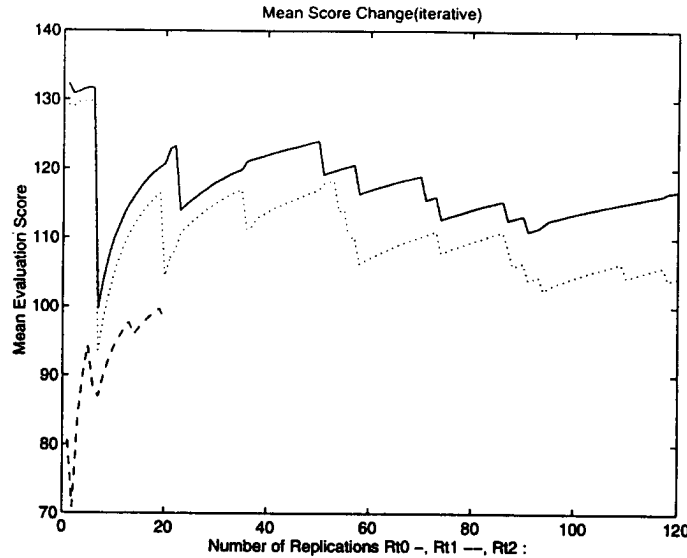


Figure 6.20. Plot of mean score changes of 3 routes using iterative method

significant pairwise difference in their means to make a selection. Using this method, we perform in total $120 + 120 + 20 = 260$ replications. With the non-iterative method, although the method decides that 10 more (130) replications are needed to make a decision on route 2, less number of replications are made in terms of the total number— $130 + 54 + 20 = 204$. Note the weighted means calculated by the non-iterative method which is used in making the final decision as to which of the two remaining routes is the best one. In this particular scenario, both the iterative and the non-iterative methods select route 0 as the best alternative.

Now, we add a 4th route, *Route 3* which is the shortest but perhaps the most dangerous route of the route. Figures 6.22 and 6.23 show the mean score changes for these 4 routes. Route 1 and 3 are eliminated after 20 replications since the average scores are significantly lower than routes 0 and 2. With another route added, the mean score change plots are somewhat different than in the case where there were only 3 routes. And this difference pushes the iterative method to continue replicating 60 more times (180 in total) for each of the two routes before it makes a decision. Consequently, it chooses route 0 as the better route—a different selection than when

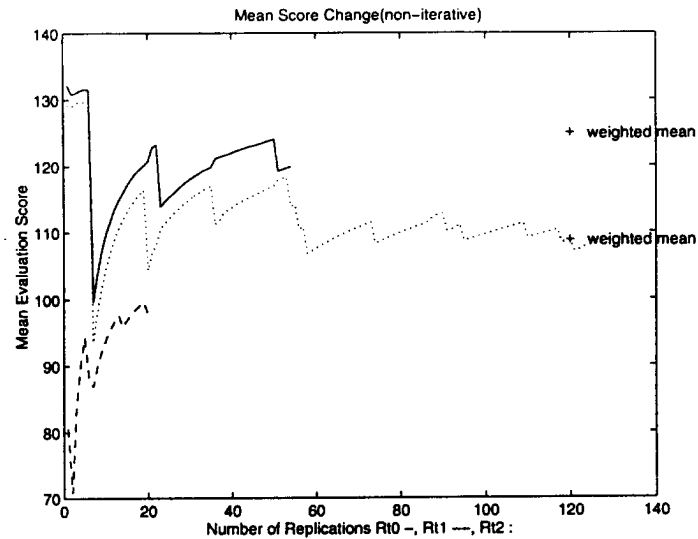


Figure 6.21. Plot of mean score changes of 3 routes using non-iterative method

only 120 replications were performed. In the non-iterative method, it makes only 69 replications for route 0 and 21 replications for route 2 before it makes a selection. It chooses route 2 to be the best route in this particular case. As discussed in chapter 2, this can occur because the non-iterative method only ensures that it makes a correct selection within a given probability P^* . And since the indifference amount d^* was chosen to be 13, it basically decides that route 0 and 2 are indifferent and many more replications is really not necessary. Overall, the iterative method performed 400 replications whereas the non-iterative method only did 130 replications.

To find out whether the mean scores of the two routes 0 and 2 reach some kind of steady state or just continue to oscillate, we performed 500 replications. Figure 6.24 shows this result. As we can see in the graph, the two routes do reach a steady state and route 0 does seem to have a higher average than route 2.

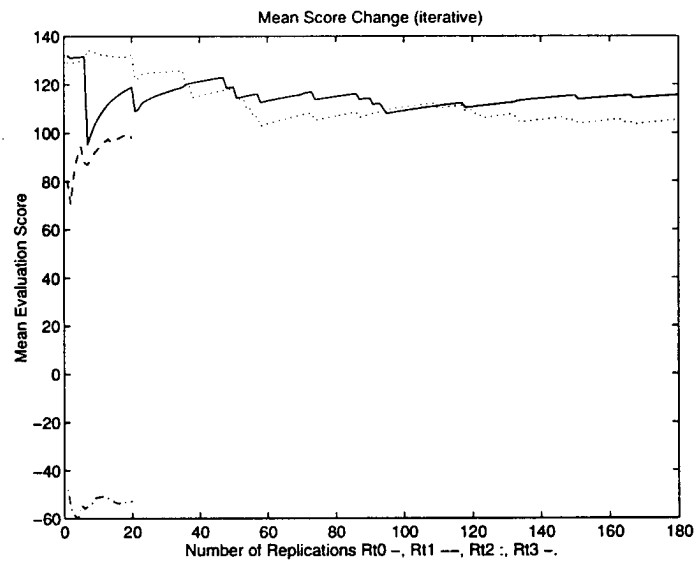


Figure 6.22. Plot of mean score changes of 4 routes using iterative method

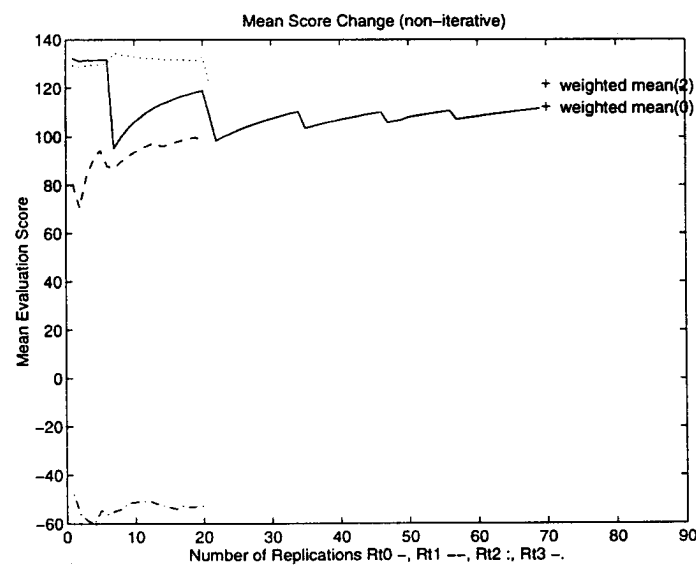


Figure 6.23. Plot of mean score changes of 4 routes using non-iterative method

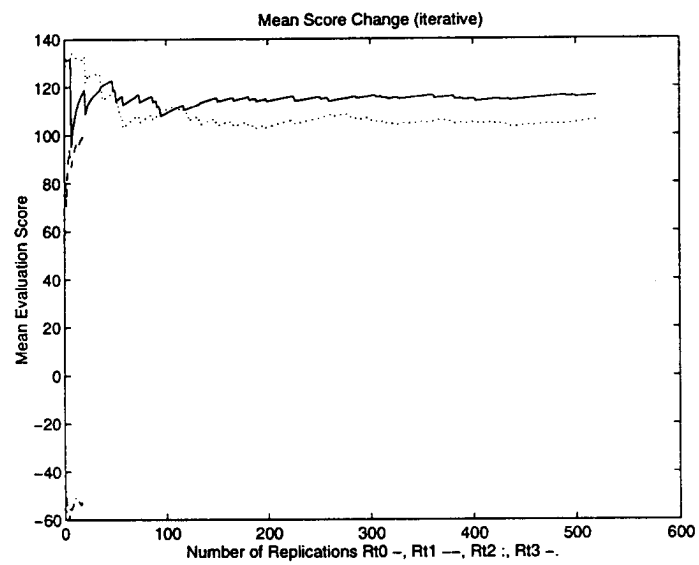


Figure 6.24. Plot of mean score changes of 4 routes for 500 replications

CHAPTER 7 CONCLUSIONS

We have presented the method of simulation-based planning as a new approach to route planning under uncertain and complex environments. Our claim is not that SBP should completely replace all other forms of planning, but that this approach be used in conjunction with already existing, higher level planning approaches. This way, given a set of alternatives to consider, SBP is able to extend the evaluation horizon in mainly three aspects: probabilistic uncertainty is handled through replicated simulation of models rather than solving them analytically using probability theory; the level of reasoning is extended to a finer level of granularity, producing plans that are closer to the level of execution while discovering subtleties that may be missed by a higher level planner; and finally, SBP breaks down the complexity of multiagent adversarial planning by employing object-oriented multimodel simulation.

There are several advantages to using Simulation to predict the results of plan execution. We list them here in addition to the ones that have been stated in section 1.3.

1. Simulation provides a uniform method without resorting to adhoc solutions. In simulation, each entity in the environment is simulated in a uniform and consistent manner by using models that represent both the physical and behavioral properties. Thus, simulating a plan is a natural consequence of simulating each of the entities by itself without having to worry about the global state change as a result of each entities action. Using object-oriented programming methods, each object is simulated using its own model.

2. Because there is no central reasoning node for the simulation but many individual simulation models for different entities, scalability is a natural consequence. Extendibility is another advantage simulation provides. For example, the effects of adding a new type of entity will be clear, only the behavior models of each entity must be updated to recognize and reason about this new entity.
3. Similar to how simulation is used for visualization, simulation can be easily used to perform visual playback of how a plan was simulated to explain the planner's decision. This can be very useful for the military since much of the military training is done through "after action review"—which is reviewing and analyzing the actions that were performed during battle.
4. Once a plan is chosen for execution, the simulation data that was generated during the planning process can be used to match with the current real world state. This can be compared to a common technique used in adaptive control theory where a reference model is compared with the actual performance data in order to tune the controller to a desired state [1].
5. Once the simulation results have been produced, the data can be analyzed and interpreted in several ways to choose the "best" plan. For instance, we can choose the plan which has not only a good average score but also the minimum variance to ensure that it is the safest plan possible. We may also decide to choose a plan that has the most number of highest scores even though the confidence interval width may be large in order to select a plan that has the best potential in spite of risks involved. We can even decide to choose a plan at random (given that the scores are above some threshold) which will produce nondeterministic planning. This is particularly useful for mission planning—opposing forces should not be able to predict your plan.

A common drawback of simulation is that it can be quite time consuming. Before the advent of fast low-cost personal computers, few researchers would consider simulation of a fairly extensive experimental design to be a possible candidate for real-time mission planning. However, as the speed of low-cost computers increases, the simulation-based planning technique is becoming more and more attractive. In order to build SBP planners that can satisfy time constraints, we have presented various ways of reducing the simulation time in terms of output analysis, experimental design and multimodeling. We have also presented a heuristic algorithm that records the average cpu time of each route simulation, uses the data to predict time usage and then designs the experiment accordingly to produce the result within the time constraint.

We have also identified some problem areas where SBP will be most useful. It is expected that for areas where there is little uncertainty involved or the level of reasoning required is only at a higher level, SBP is not likely to do any better. Many factors affect the success of an SBP planner and we briefly mention some of them here. As with any simulation, SBP will be only as good as the models that we build to represent the world that we are planning in. In most cases, building valid models is not a straightforward task. Validating that models indeed accurately represent the world is an issue that we have not addressed in this thesis. Building good evaluation functions that correctly represents the need of the user is also an important aspect and is still somewhat of a trial and error process. All these and many other factors must be carefully researched and some guidelines must be developed in order to ensure that a user will build a useful planning system.

CHAPTER 8 FUTURE WORK

The main focus of our work was in building a framework for SBP. There are many areas that we have not addressed in much depth and we discuss them here as our future work.

Validation of simulation models—making sure that the models we build appropriately represents the actual object—is an important aspect that we must address. A common approach is using sensitivity analysis or an inspection by an expert. An interesting work related to this issue is validating and checking consistencies of a higher rule-based systems. Using the predicted outcome of an existing higher rule-based systems and the prediction of an SBP system (perhaps at a lower level of detail), we can compare the two predictions. Based on the comparison, we can validate one system against another and find any inconsistencies, subtleties that may have been missed by the higher level system. Taking it another step further, we may modify and improve the higher level system using the information obtained from the comparisons. Currently, research is underway in our simulation group that focuses on studying effective consistency measures which will rectify differences in rules produced empirically (through knowledge acquisition) and rules generated automatically from multiple low-level simulations.

More detailed, sophisticated models should be built to obtain better results in terms of answer quality and also test the degree of cpu time consumption in respect to the model's complexity. An immediate future work would be to extend the implementation of the Air Force models to include all the levels of abstraction as shown

in chapter 6. Larger number of objects should also be simulated to further study the scalability and the rate change of time consumption.

Extending the multimodeling paradigm to enable model execution at any level of abstraction is also currently underway and SBP can greatly benefit from the success of this work since it will allow reduction of model execution time. Possibilities exist for future work in finding other ways of meeting real-time constraints: a hybrid approach of using quantitative and qualitative (fuzzy) simulation, developing additional heuristics to aid in optimizing the simulation process are some ideas that we plan to research in the future.

Finally, to further extend the study of the SBP methodology, additional experiments in other application areas should be performed. Also, building and comparing two planning systems, one built using the SBP approach and one built using another planning approach, should prove to be useful in further improving the SBP approach.

REFERENCES

- [1] P. J. Antsaklis and K. M. Passino. *An Introduction to Intelligent and Autonomous Control*. Kluwer Academic Publishers, Norwell, MA, 1993.
- [2] J. Barraquand, B. Langlois, and J. Latombe. Numerical Potential Field Techniques for Robot Path Planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2):224-241, 1992.
- [3] T. Basar. *Dynamic Noncooperative Game Theory*. Academic Press, San Diego, CA, 1995.
- [4] G. Booch. *Object Oriented Design with Applications*. Benjamin Cummings, Redwood City, CA, 1991.
- [5] R. A. Brooks. A robot layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14 - 23, 1986.
- [6] M. Czigler, S. Downes-Martin, and D. Panagos. Fast Futures Contingency Simulation: A "What If" Tool for Exploring Alternative Plans. In *Proceedings of the 1994 SCS Simulation MultiConference*, San Diego, CA, 1994.
- [7] D. D. Dankel and A. J. Gonzalez. *The Engineering of Knowledge-Based Systems*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [8] P. K. Davis. An Introduction to Variable-Resolution Modeling and Cross-Resolution Model Connection. Technical report, RAND, Santa Monica, CA, 1993.
- [9] T. L. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning With Deadlines in Stochastic Domains. In *Proceedings of the AAAI-93*, Washington, DC, 1993.
- [10] T. L. Dean and M. P. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.
- [11] M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan. *Deterministic and Stochastic Scheduling*. Reidel, Dordrecht, 1982.
- [12] Department of the Army. *The Tank and Mechanized Infantry Battalion Task Force, FM 71-2*. Department of the Army, 1987.
- [13] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3:251-288, 1972.
- [14] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Some new directions in robot problem solving. In *Machine Intelligence 7*. Edinburgh University Press, 1972.

- [15] P. A. Fishwick. Heterogeneous Decomposition and Coupling for Combined Modeling. In *1991 Winter Simulation Conference*, pages 1199 – 1208, Phoenix, AZ, December 1991.
- [16] P. A. Fishwick. An Integrated Approach to System Modelling using a Synthesis of Artificial Intelligence, Software Engineering and Simulation Methodologies. *ACM Transactions on Modeling and Computer Simulation*, 2(4):307 – 330, 1992.
- [17] P. A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds*. Prentice-Hall, 1995.
- [18] P. A. Fishwick and B. P. Zeigler. A Multimodel Methodology for Qualitative Model Engineering. *ACM Transactions on Modeling and Computer Simulation*, 1(2):52 – 81, 1992.
- [19] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of Jop-Shop*. Ellis Horwood Limited, 1982.
- [20] D. Harel. On Visual Formalisms. *Communications of the ACM*, 31(5):514 – 530, May 1988.
- [21] D. Harel. Biting the Silver Bullet: Toward a Brighter Future for System Development. *IEEE Computer*, 25(1):8 – 20, January 1992.
- [22] D. Hille, M. R. Hieb, and G. Tecuci. CAPTAIN: Building Agents that Plan and Learn. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, pages 411–422, Orlando, FL., 1994.
- [23] L. P. Kaelbling. An architecture for intelligent reactive systems. In *Reasoning About Actions and Plans*, pages 395 – 410. Morgan Kaufmann, Los Altos, CA, 1987.
- [24] C. R. Karr, R. W. Franceschini, K. R. S. Perumalla, and M. D. Petty. Integrating Aggregate and Vehicle Level Simulations. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pages 231–239, Orlando, FL., 1993.
- [25] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 1991.
- [26] J. J. Lee, W. D. Norris, and P. A. Fishwick. An Object-Oriented Multimodeling Design for Integrating Simulation and Planning Tasks. *Journal of Systems Engineering*, 3:220–235, 1993.
- [27] R. D. Luce and H. Raiffa. *Games and Decisions*. John Wiley and Sons, New York, NY, 1957.
- [28] L. Matthies, E. Gat, R. Harrison, B. Wilcox, Volpe R., and T. Litwin. Mars microrover navigation: Performance evaluation and enhancement. Technical report, Jet Propulsion Laboratory, Pasadena, CA, 1995. To appear in *Autonomous Robots Journal*.
- [29] H. J. Moore and B. M. Jakosky. Viking Landing Sites, Remote-Sensing Observations, and Physical Properties of Martian Surface Materials. *International Journal of Solar System Studies*, 81:164–184, 1989.

- [30] H. Praehofer. *Theoretic Foundations for Combined Discrete Continuous System Simulation*. PhD thesis, University Linz, Austria, 1991.
- [31] P. Rogers and R. J. Gordon. Simulation for Real-Time Decision Making in Manufacturing Systems. In *1993 Winter Simulation Conference*, pages 866–874, Los Angeles, CA, December 1993.
- [32] D. W. Rolston. *Principles of Artificial Intelligence and Expert Systems Development*. McGraw-Hill, Inc., 1988.
- [33] J. Rumbaugh, M. Blaha, W. Premerlani, E. Frederick, and W. Lorenson. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
- [34] S. J. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*. Prentice-Hall, 1995.
- [35] M. Salisbury and H. Tallis. Automated Planning and Replanning for Battlefield Simulation. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pages 243–254, Orlando, FL., 1993.
- [36] S. M. Sanchez. A Robust Design Tutorial. In *1994 Winter Simulation Conference*, pages 106–113, Lake Buena Vista, FL, December 1994.
- [37] M. Schoppers. Universal Plans for Reactive Robots in Unpredictable Domains. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 1039–1046, Milan, Italy, 1987.
- [38] R. Shannon. *Systems Simulation: The Art and Science*. Prentice Hall, 1975.
- [39] M. Spick. *An Illustrated Guide to Moden Attack Aircraft*. Prentice Hall Press, 1987.
- [40] A. Thesen and L. E. Travis. *Simulation Model for Decision Making*. West Publishing Co., 1992.
- [41] M. P. Wellman. *Formulation of Tradeoffs in Planning Under Uncertainty*. Pitman, London, 1990.
- [42] M. P. Wellman, M. Ford, and K. Larson. Path Planning under Time-Dependent Uncertainty. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, Montreal, Canada, August 1995.
- [43] S. D. Wu and R. A. Wysk. An Application of Discrete-Event Simulation to On-Line Control and Scheduling in Flexible Manufacturing. *Internation Journal of Production Research*, 27:1603–1623, 1989.
- [44] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, June 1965.
- [45] L. A. Zadeh. Fuzzy logic. *IEEE Computer*, pages 83 – 93, April 1988.
- [46] B. P. Zeigler. *Object Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press, 1990.

BIOGRAPHICAL SKETCH

Jin Joo Lee received the B.S. degree in computer science from Ewha Womans University, Korea in 1988 and the M.S. degree in computer science from Brown University in 1991. After receiving the M.S. degree, she was a research engineer at Human Computers Inc., Korea until 1992. She entered the Ph.D. program in computer science at University of Florida in the spring of 1992 and plans to graduate in August of 1996. Her research interests are in simulation modeling, AI planning and intelligent control.