Fox, M.S., & Reddy, Y.F., (1982), "Knowledge Representation in Organization Modeling and Simulation: Definition and Interpretation", Modeling and Simulation, Vol. 13, W.G. Vogt & M.H. Mickle (Eds.), pp. 675-683, Research Triangle Park NC: Instrument Society of America.

Knowledge Representation in Organization Modeling and Simulation: Definition and Interpretation[1]

Mark S. Fox & Y.V. Reddy[2]

The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

1. Introduction

In the summer of 1980 we began the study of problems in managing complex organizations such as job-shop factorie Our purpose was to discover where intelligent systems may aid in the achievement of organizational goals. Our analys resulted in the formation of the Intelligent Management System (IMS) Project (Fox, 1981). IMS is a long-term proje concerned with applying artificial intelligence techniques in aiding professionals and managers in their day to day task Research in IMS is proceeding in many areas, including: job-shop scheduling, flexible simulation, process diagnos organization modeling, and user-interfaces. This paper discusses the application of **Knowledge Representation a** its application to Modeling and Simulation.

A commonly occurring problem in management is the inability to answer "what if" questions readily. In a survey questions posed by managers in three plants, many were concerned with the effect of proposed changes in facto organization. Some could be answered based upon previous experience, some by analysis, but many went unanswere Why do these questions remain unanswered when tools exist for analyzing organizations? In particular, simulati systems are used to measure performance of existing or proposed systems which are too complex to be studi analytically. It is the cost of construction that limits their use, and resulting systems are little used except when runni the same or similar simulations again. More importantly, simple "what if" questions cannot be answered readily. manager requires an intermediary, such as a system analyst, to answer them. There is a definite need for mo sophisticated tools for analyzing organizations, and for providing usable tools directly to the managers and professiona

In this paper we describe the technique of Knowledge Representation and how it can be used to create a flexib simulation system (hereafter referred to as KBS) for Organizational Modeling.

Our reasons for creating yet another simulation system are numerous. In particular, issues we have explored in ou research include:

- creating a system modeling language that can simultaneously support multiple applications in addition to simulation. Thus eliminating the need and cost of maintaining multiple models.
- representing the behavior of system entities directly in the model. This admits total flexibility in creating and altering entities and their behavior, without altering the simulation model interpreter.
- allowing the system to be selectively instrumented. This restricts data analysis to areas of interest, and provides support of graphics displays.
- representing the system at multiple levels of abstraction. This allows the user to specify the level of simulation and the detail-level of results.
- consistency and completeness checking. Much time is spent verifying that models are consistent and complete. We have developed a checker which detects model incompleteness and inconsistencies.
- providing interactive access to the model building and simulation system. This appears to reduce model building time, and provide a more intimate understanding of the simulation.

The rest of the paper provides an overview of knowledge representation, and SRL (Fox, 1982), the language used build KBS. It also introduces the concept of simulation as an interpretation of a knowledge base, followed by an examp of a simple model with two machines.

2. Knowledge-Based Modeling and Simulation

A primary focus of our research has been the underlying modeling theory[3], because the model dictates th applicability and ease with which simulations can be constructed. In examining current modeling techniques, a variety simulation modeling theories and methodologies have been introduced over the years. These systems can be classifie as:

- Programming languages,

- Extended Programming Languages, and

- Special purpose packages.

Simscript II (Kiviat, 1969) and Simula (Dahl,1967) fall into the category of extended programming languages since they provide a total programming environment in which the usual programming constructs are augmented with simulation oriented language constructs such as **queues**, **events** and **entities**. These facilities make it easier to specify a model as opposed to a general purpose programming language such as **FORTRAN** or **PASCAL**. But they provide maximum flexibility at the cost of considerable programming effort.

GASP (Pritsker,1974) and **DESPL/1** (Reddy, 1973) also belong to the class of extended programming languages since they extend the facilities of a general purpose programming language by adding preprocessors or subroutine packages to implement simulation features.

On the other hand, systems such as **GPSS** (IBM,1970) provide a flowchart type of facility which is easy to use but provides limited flexibility, since it is not embedded in a rich programming environment. Systems such as **QGERT** (Pritsker,1977), **RESQ** (Sauer,1978), **BORIS** (Wendt,1980) and **INS** (Roberts,1980) are designed to provide model building facilities without extensive programming knowledge. These systems have their own limitations in that they take a particular approach such as a queueing network or Petrinets (Zisman,1978). **DEMOS** (Birstwistle,1980) is an extreme example of building a simulation system in that it extends another simulation language, **SIMULA**, to make it easier to build models.

A central theme of these systems is "How to make model building effortless?". However, they suffer from a number of drawbacks, such as lack of flexibility in expressing a model structure and requiring extensive programming effort. For example, in GPSS the programmer is restricted to the concepts of *facility, transaction* and *queue* and the model is to be constructed as a flowchart using these blocks. Also the there are no facilities for the selective collection of statistics. In programming systems such as SIMSCRIPT II and SIMULA the structure of the model is embedded in the program which realizes the model and thus any structural changes to the model require program modification.

Also, most of the current modeling systems are *batch* oriented which puts severe limitations on the model optimization process. A model is generally conceived by management personnel who have little programming expertize and thus requires the services of a programmer to translate the model into a program. Often the programmer has little understanding of the system being modeled. Because the various modeling assumptions are *hardwired* into the code, the model builder cannot be expected to verify whether all the assumptions have been faithfully translated into code. In addition, even small structural changes to the model turnout to be major programming projects.

The goals of IMS include the integration of functionality such as simulation, scheduling, and diagnosis into a single, distributed system, and making all functions accessable to managers and professionals. In order to accomplish integration we sought to create a "single" model of the organization (system) that is accessable by all IMS functions, including simulation. To achieve this, we had to develop a method of modeling that is

- rich in the modeling concepts it can represent, hence easing the mapping from domain to model.

- easily extendable if the modeling system does not fit the domain.

- understandable by all functions that wish to access the model. That is, the semantics of the model are embedded in the model, and not the programs that manipulate it.

The approach taken was to use an Artificial Intelligence (AI) knowledge representation system in which a library of entities can be created and instantiated, defining both attribute and behavioral descriptions. In order to answer "what if" questions, the knowledge base should contain various facts about each *entity* in the system and its relationship to other entities, and process knowledge about the effect of actions in the system. It should also include knowledge about the relationship between entities and consistency specifications. For example, the knowledge base should contain the fact that to perform a certain operation on a work-piece, we need a machine/operator capable of performing that operation. Another piece of information may be that the state of a machine changes from "busy" to "free" when it unloads its current work-piece. In addition to this general knowledge, the knowledge base should contain specific information about system such as "operation o-150x is done by machine m-nc-drill1".

In order to achieve user accessablity, we sought to add to the knowledge representation system functions that provide the following characteristics:

- Creation of models should require little programming effort. The modeling system should have or allow the creation of entities that match the concepts of the domain being modeled.

- The model creation and alteration interface should be interactive.

- The model should be selectively instrumentable in order to gather and analyze data, and to provide run-time output.

- The model should be alterable *during* the simulation run to allow the real-time testing of hypotheses.

- The model should be automatically examined for consistency and completeness. This reduces the amount of model debugging.

A number of knowledge representation languages such as KRL (Bobrow & Winograd, 1978), Klone (Brachman, 1978), NETL (Fahlman, 1978) have been used in various artificial intelligence systems. This system is implemented in SRL (Schema Representation Language) (Fox, 1982), which runs under the VAX FRANZ lisp system (Foderaro, 1980). The rational for choosing an AI knowledge representation language is two fold. First, research in knowledge representation has been concerned with the representational semantics of knowledge in general. Thus the meaning of information in the model is embedded in the model, and not in the functions that access it. Second, knowledge representations are both flexible and extendable. Alterations to existing information in models does not necessarily require massive reorganization of the model structure. And new information (e.g., entities, relations, etc.) can be added, again without major alteration.

The approach taken in our research is similar to another AI-based simulation system called ROSS (Klahr & Fought, 1980). Both KBS and ROSS are object oriented modeling system which contain attribute and behavioral descriptions, and provide interactive access and display. They differ in that our approach separates the model from its interpreter. The model is the kernal of IMS, and must support a variety of functions including factory monitoring, scheduling, and question-answering, in addition to simulation. Hence, our system is an interpreter which accesses the model, providing simulation, model checking, and data analysis capabilities.

2.1. Modeling Entities and Relations

The IMS modeling system provides the following features:

- The model is composed of declarative objects and relations which match the users conceptual model of the organization.

- The modeling system provides a library of objects and relations which the user may use, alter, and/or extend in their application.

- The model incorporates a variety of representational techniques allowing a wide variety of organizations to be modeled (continuous and discrete). And it is extensible, allowing the incorporation of new modeling techniques.

- The user interactively defines, alters, and peruses the model.

- The model can be easily instrumented. For example, the model can be diagramatically displayed on a color graphics monitor at different levels of abstraction. The complete organization, or parts thereof, can be viewed with summaries (e.g., queue lengths, state).

- The modeling system is simple to learn to use because the modeling tools match the concepts people use to think about problems.

The basic unit for representing objects, processes, ideas, etc. is the **Schema**. Physically, a schema is composed of a schema name (printed in the bold font) and a set of slots (printed in small caps). A schema is always enclosed by double braces with the schema name appearing at the top. The **Machine** schema (figure 2-1) contains eight slots, some which define physical limitations of the machine, i.e., CAPACITY, some which define its current status, i.e., OPERATOR, and some which define event behavior, i.e., LOAD. Slots can have simple values (figure 2-2). Schemata can be more complex. Each slot has a set of associated facets (printed in italics) (figure 2-3). The *Restriction* facet restricts the type of values that may fill the slot. The *Default* facet defines the value of the slot if it is not present. And each filler of a *facet* may have one or more pieces of meta-information termed characters (printed underlined) (figure 2-4). The Filler character defines the value of the facet. Creator defines who created the filler, and Creation-Date defines when the filler was created.

{{ **Machine**

CAPACITY:
OPERATOR:
CONTENTS:
LOAD:
UNLOAD:
INPUT-Q:
OUTPUT-Q:
SERVICE-TIME: }}

Figure 2-1: Machine Schema

{{ **Machine**

CAPACITY: 3
OPERATOR: joe
CONTENTS: lot-29
LOAD:
UNLOAD: }}

Figure 2-2: Machine Schema with values

An important aspect of SRL is that schemata may form networks. Each slot in a schema may act as a relation tying the schema to others. The schema may *inherit* slots and their fillers along these relations. Consider the schema for a **Continuous-machine**. Figure 2-5 defines a CONTINUOUS-MACHINE which works much like a pizza oven, it can be continuously filled up to capacity. A **Continuous-Machine** IS-A **Machine**. The IS-A relation between the two schemata allows **Continuous-Machine** to inherit attributes (slot names) and their values from the **Machine** schema. The LOAD slot defines the behavior of the machine when a load event occurs. The loading rule tests whether the machine has capacity, if so the object is placed in the machine, otherwise it is queued.

Another type of inheritance relation used is **part-of**. The part-of inheritance relationship may be used to define spatial

{{ **Machine**

CAPACITY:
Value: 3
OPERATOR:
Value: joe
CONTENTS:
Restriction: (TYPE is-a product)
LOAD:
Restriction: (SET (TYPE is-a rule))
Default: load-rule
UNLOAD:
Restriction: (SET (TYPE is-a rule))
Default: unload-rule
}}

Figure 2-3: Machine Schema with facets

{{ **Machine**

CAPACITY:
Value:
Filler: 3
Creator: shop-supvervisor
Creation-Date: 22-OCT-79
OPERATOR:
Value:
Filler: joe
CONTENTS:
Restriction:
Filler: (TYPE is-a product)
LOAD:
Restriction:
Filler: (SET (TYPE is-a rule))
Default:
Filler: load-rule
UNLOAD:
Restriction:
Filler: (SET (TYPE is-a rule))
Default:
Filler: unload-rule
}}

Figure 2-4: Machine Schema with characters

elationships such as layout of a factory. Figure 2-7 redefines **nc-drill-1** as being both an instance of an **nc-drill** (figure

{{ **Continuous-Machine**
{ IS-A **Machine**
USED-CAPACITY:
LOAD: {{ INSTANCE **# rule**
IF: (< USED-CAPACITY CAPACITY)
THEN: (fill USED-CAPACITY (+ 1 USED-CAPACITY))
(add object CONTENTS)
ELSE: (add object QUEUE) }}
}
}}

Figure 2-5: Continuous-Machine Schema

{{ **nc-drill**
{ IS-A **machine**
LOAD: nc-load
UNLOAD: nc-unload } }}

Figure 2-6: nc-drill Schema

2-6) and part of the work area **drill-room**.

SRL provides the model builder with the ability to define new schemata and slots, and to define the inheritance semantics of slots which act as relations. This includes defining what information, i.e., slots and their values, is inherited, not inherited, and altered when inherited.

2.2. Rules of Behavior

The LOAD and UNLOAD slots represent events that can take place at an **nc-drill**. Both LOAD (figure 2-8) and UNLOAD exist as schemata whose relations define them as both **slots** and **events**.

{{ **nc-drill-1**
{ INSTANCE **nc-drill** }

{ PART-OF **drill-room** } }}

Figure 2-7: nc-drill-1 with PART-OF

{{ **load**
{ INSTANCE **slot** }

{ IS-A **event** } }}

Figure 2-8: Load Schema

The behavior that is to be displayed by an entity when the event occurs is defined by the fillers of the associated even slot. The filler of the LOAD slot is a **rule** which defines the object's event behavior. A rule has two parts, IF which tests th applicability of the rule, and a THEN slot whose contents are executed when the rule is applicable. The contents of thes slots are either other schemata (i.e., rules or functions), or lisp code. Figure 2-9) defines how a machine is loaded. It i important to note that a rule provides a behavioral description of an event at the level of detail defined by the entity. N more, no less. If the entity is an abstraction of a more detailed description, then the rule is also an abstraction. Entities events, and their behaviors can be successively refined into more detailed descriptions. The implication of thi refinement process on simulation will be discussed later in the paper.

2.3. Model Libraries

Using SRL, we can define a set of schemata representing various types of generic objects, processes , behavioral rule and scheduling algorithms relevant to many domains, and store them in a model library. Once this library is create specifying an individual model consists of instantiating relevant schemata from the library. The user may also add t and/or alter library schemata, depending on their simulation needs.

```
{{ load-rule
   { INSTANCE rule
     IF:   state = free &
           contents of input-source not empty
     THEN: select object to be loaded &
           update contents &
           change state to busy &
           execute statistics-rule.
     ELSE: do nothing.   } }}
```

Figure 2-9: **load-rule** Schema

```
{{ event23
   { INSTANCE event-notice
     EVENT-TIME: 2.8
     EVENT-NAME: load
     EVENT-FOCUS: m-nc-drill1
     EVENT-PARAMS: order23   } }}
```

Figure 2-10: An Example of an Event Notice

The schemata for various objects and processes are arranged in a hierarchy where, each schema may inherit the slots and values from schemata directly above them in the hierarchy. For example, schemata representing various machines and "agents" (such as operators) and inspectors can form a schema hierarchy where, at the highest level there is a schema: **agent**. This schema represents anything that performs an operation. At the next level, there are three schemata: **machine**, **manual-agent** and **man-machine-agent**. These represent refinements of the **agent** schema and represent an operatorless machine, an inspector/manual operator and an operator assisted machine respectively. Each of these schemata, inturn can be refined to represent specialized objects such as numerically controlled machines and semiautomatic machines. Similarly, we can create a hierarchy of schemata representing various types of storage areas such as queues and random access storages[4].

In addition to the various schema hierarchies representing different concepts in a factory domain we need a set of schemata to describe various events representing the behavior of the model. These schemata are represented as rules some of which we have already encountered in earlier sections. For example the model library may contain the following behavioral rules for a factory modeling environment: **discrete-load-rule** and **continuous-load-rule** may be refinements of the schema: **rule**, and represent the concept of loading discrete and continuous machines respectively. In addition to these behavioral rules, we can have a number of rules representing various scheduling philosophies. For example, there may be a rule to represent a **global scheduler** for handling output from all machines in the model.

2.4. Simulation Via Model Interpretation

The simulation model is driven by a **clock**. The clock is advanced to the current time each time an event is executed. The occurrence of an event is represented by an **event-notice**. Event notices representing future events are stored in a **calendar** ordered by their expected time of occurrence. For example an arbitrary event: **event23** can be specified as shown in figure 2-10. This schema represents an event called load to occur when clock shows 2.8 units of time. The event is related to the object called **m-nc-drill1**. It may be interpreted as:

```
load m-nc-drill1 with order 23 at 2.8
```

What happens when the above event occurs is defined by the rules in the LOAD slot of **nc-drill1**.

The calendar is represented by a **calendar** schema. *Scheduling* an event simply consists of inserting the event notice in the EVENT-LIST slot of **calendar** in the appropriate place. *Execution* of an event involves execution of the rules in the slot denoted by EVENT-NAME in the schema represented by EVENT-FOCUS slot of the event notice.

3. An Example

In this section we describe the realization of a simple simulation model using the knowledge representation approach. The model consists of two machines, **machine1** and **machine2**, and two queues, **queue1** and **queue2**. **machine1** (figure 3-2) is of type **discrete-machine** (figure 3-1) which is a sub-type of the schema **machine**. This type is used to represent machines which can process one object at a time. The LOAD and UNLOAD rules associated with this type are also refinements of the schema **load-rule**. **queue1** (figure 3-4) and **queue2** are of the type **fifo-queue** which is a refinement of the type **queue**. The model represented by these schemata is a "two-stage single server queueing system". Objects that enter **queue1** are served by **machine1** and passed on to **queue2** where they are processed by **machine2** and passed on to the next stage (if one exists).


```
{{ discrete-machine
   { IS-A machine
     LOAD: discrete-load-rule
     UNLOAD: discrete-unload-rule } }}
```

Figure 3-1: **discrete-machine** Schema

```
{{ machine1
   { INSTANCE discrete-machine
     INPUT-Q : queue1
     OUTPUT-Q : queue2
     SERVICE-TIME: 0.5 } }}
```

Figure 3-2: **machine1** Schema

There also exists a **system** schema to provide information *about* the simulation model itself. The actions to initialize the model are specified by the value of the slot: PRIME in the schema **two-stage-queueing-model** shown in figure 3-5. The value of the slot: START-SIM specifies the required actions to start the execution of the model.

The various actions that take place during the course of model execution cause "event-notices" to be created and deposited in the **calendar** schema. Execution of events specified by the event-notice are realized by the following steps:

- Identify the schema which is the focus of this event-notice.
- Extract the rules from the slot with the name of the event-type.
- Evaluate the rules.

Having defined the various schemata representing the components of the two stage queueing model and its associated rules, we are now ready to describe the detailed operation of the model.

```
{{ machine2
   { INSTANCE continuous-machine
     INPUT-Q: queue2
     OUTPUT-Q: queue3
     SERVICE-TIME: (FUNCTION stime)
     CAPACITY: 5  } }}
```

Figure 3-3: **machine2** Schema

```
{{ queue1
   {INSTANCE fifo-queue
     SOURCE: nil
     DESTINATION: machine1
     ARRIVES: arrival-rule1 } }}
```

Figure 3-4: **queue1** Schema

Initially, the EVENT-LIST slot of **calendar** contains **prime-event**. Hence, the first action taken by the system is to execute the **prime-rule** (figure 3-7). **Prime-rule** has a true condition, hence the THEN slot is evaluated. It contains the function read-orders which reads orders in from a file, creates a schema for each, and possibly schedules an event for each order. In this example, each order results in an arrival event at **queue1**.

At this stage the **calendar** contains event notices as shown in figure 3-8. The schema definition for the event notice **event1** is shown in figure 3-9.

The simulation continues by removing the first event notice, **event1**, found in **calendar** and interpreting it. The event states that **order1** "arrives" at **queue1** at time 0. To interpret the "arrives" event, KBS evaluates the contents of the ARRIVES slot in **queue1** schema. In this case it is the rule: **arrival-rule1** (figure 3-6). As can be seen from the definition of **arrival-rule1**, if the machine is free, the THEN slot causes an event notice to be generated and put on the calendar (EVENT-TIME: 0.0; EVENT-NAME: LOAD; EVENT-FOCUS: machine1; EVENT-PARAMS: order1). This event is executed by

```
{{ two-stage-queueing-model
   {INSTANCE system
      PRIME-EVENT: prime-rule
      TOTAL-TIME :
      TOTAL-EVENTS :    } }}
```

Figure 3-5: Schema Definition for the Current Model

```
{{ arrival-rule1
   { INSTANCE rule
      IF: (status of machine = free)
      THEN: (schedule a load for the machine) } }}
```

Figure 3-6: **arrival-rule1** Schema

```
{{ prime-rule
   { INSTANCE rule
      IF: (t)
      THEN: (Function read-orders order-file) } }}
```

Figure 3-7: **prime-rule** Schema

```
{{ calendar
   { INSTANCE priority-queue
      EVENT-LIST: (event1, event2, event3, ...) } }}
```

Figure 3-8: **Calendar** Schema After Execution of Prime Event

evaluating the "rule" associated with the LOAD slot of **machine1**. Since **machine1** is defined to be of the type

```
{{ event1
   { INSTANCE event-notice
      EVENT-TIME: 0.0
      EVENT-NAME: arrives
      EVENT-FOCUS: queue1
      EVENT-PARAMS: order1 }
```

Figure 3-9: **event1** Schema

```
{{ event3
   { INSTANCE event-notice
      EVENT-TIME: (current-time + service-time)
      EVENT-NAME: unload
      EVENT-FOCUS: machine1
      EVENT-PARAMS: order1 } }}
```

Figure 3-10: **event3** Schema

discrete-machine, it will inherit the "load rule" associated with machines of that type. This results in "scheduling" of a future event to unload **machine1**. This is shown in figure 3-10.

When **event3** (see figure 3-10) is executed by evaluating the "unload rule" this will cause two more events : **event4** (to load machine1) and **event5** (to cause order1 to arrive at queue2). Each of these events will inturn cause further events (**event4** will cause **event6** to unload order2 and **event5** will cause **event7** to load **machine2** with order1). This chain of events will continue until the simulation is halted for lack of "outstanding notices", or because of meeting prespecified conditions. The next section describes selective instrumentation of models built using KBS.

4. Conclusion

In this paper we took the view that a simulation model need not be explicitly constructed, but rather be derived from a **knowledge base**. We also show that a suitable knowledge base can be constructed using the SRL knowledge representation facility. It also demonstrates that model acquisition can be accomplished by the instantiation of generic

schemata found in a library for a specific domain. Events are represented as rules associated with schemata which provide a convenient method for specifying various actions in the model. A companion paper(Reddy and Fox) in this proceedings describes the model acquisition process and a system for checking consistency and completeness.

5. References

Birtwistle G.,(1980), "The DEMOS Discrete Event Package", *Proceedings of the Summer Computer Simulation Conference*, 1980, pp 179-183.

Bobrow D., and T. Winograd, (1977), "KRL: Knowledge Representation Language," *Cognitive Science*, Vol 1, No. 1, 1977.

Brachman R.J., (1977), "A Structural Paradigm for Representing Knowledge," (Ph.D. Thesis), Harvard University, May 1977.

Dahl, (1967), "SIMULA: A Language for Programming and Description of Discrete Event System", User's Manual, Norwegian Computer Center.

Fahlman S.E., (1977), "A System for Representing and Using Real-World Knowledge," (Ph.D. Thesis), Artificial Intelligence Laboratory, MIT, AI-TR-450.

Foderaro J.K., (1980), "The FRANZ LISP Manual", Department of Computer Science, University of California at Berekely.

Fox M.S., (1981), "The Intelligent Management System: An Overview", Technical Report CMU-RI-TR-81-4, Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, July 1981.

Fox M.S., (1982), "SRL: Schema Representation Language", Technical Report, Robotics Institute, Carnegie-Mellon University, Pittsburgh PA, in preparation.

IBM, (1973), "GPSS/360 Introductory User's Manual", GH20-0304-4.

Kiviat P.J., Villanueva, R., Markowitz, H.M., (1969), *The SIMSCRIPT II Programming Language*, Prentice- Hall.

Klahr P. and W.S. Fought, (1980), "Knowledge-Based Simulation", *Proceedings of the First Annual Conference of the American Association for Artificial Intelligence*, Stanford CA, pp. 181-183,

Pritsker A.A.B., (1974), *The GASP IV Simulation Language*, New York: John Wiley and Sons.

Pritsker A.A.B., (1977), *Modeling and Analysis using Q-GERT networks*, New York: John Wiley and Sons.

Reddy Y.V. and Bryan, R.H., (1973), "DESPL/1: A PL/1 Based Simulation Language", *Proceedings of the Summer Computer Simulation Conference*, Seattle WA, pp. 100-106.

Roberts D.S. and Scheir, J.S., (1980), "INS: A Simulation Language which facilitates modeling and Analysis" *Proceedings of the Summer Computer Simulation Conference*, Seattle WA, pp. 36-41.

Sauer C.H., (1978), "Characterization and Simulation of Generalized Queueing Networks", RC6057, IBM Research Yorktown Heights, NY.

Wendt S., (1980), "BORIS: A new General Purpose Interactive Simulator for Hierarchical Models of Discrete Systems" *Proceedings of the Summer Computer Simulation Conference*, Seattle WA, pp. 50-55.

Zisman M.D., (1978), "Use of Production Systems for modeling Concurrent Processes", *Pattern Directed Inference Systems*, Waterman, Hayes-Roth, & Lenat (Eds.), Academic Press, pp. 53-68.

[1]This research was supported in part by the Westinghouse Corporation, and by the CMU Robotics Institute.

[2]Y.V. Reddy is with the Computer Science Dept., West Virginia University, Morgantown, West Virginia.

[3]In this paper we restrict ourselves to discrete event simulation systems.

[4]The refinement and alteration of schemata is described in (Fox, 1982).