Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

An Investigation of a Standard Simulation-Knowledge Interface

Carol S. Russell
Adel S. Elmaghraby
James H. Graham

Computer Science and Engineering Program
University of Louisville
Louisville, Kentucky 40292, U.S.A.

Abstract

The benefits of separating the simulation of a physical system from the decision-making process are discussed. A formal model for the interface between such a simulation and a knowledge-based manager is presented. The model is then applied to a case study involving a scheduling and planning problem from manufacturing.

1 Introduction

Simulations are conducted for the purpose of illustrating, defining and/or analyzing the behavior of complex systems. The system itself is often a collection of subsystems. While the subsystems may be well-defined, it is possible that the interaction among the subsystems is not well-defined. When this is the case, the usual procedure is to provide some *a priori* rule to regulate the interactions and then run the simulation. After successive runs, the data from the simulation are analyzed and necessary modifications are made in the simulation.

While this run-analyze-modify procedure is appropriate for some simulations, it is not necessarily the most efficient or the most productive method. This is especially true if the modifications are applied to the rules of interaction among modules within a system which is sensitive to rapidly changing conditions. In systems of this type, it may be desirable to provide a mechanism by which decisions concerning the interaction of the subsystems can respond to current simulation conditions by modifying the rules of interaction while the simulation is still in progress.

This work will show that by separating the physical characteristics of the system from the rules of interaction, a more efficient and effective simulation will result. This will allow:

1. dynamic allocation of resources and processes;
2. selection of best interactive rule based on current and changing conditions; and
3. decoupling into a modular simulation which can accommodate changes in either the physical description and/or the management objectives.

Section 2 provides a brief overview of the incorporation of intelligent decision-making into simulations and a rationale for the proposed approach. Section 3 presents a formal model for an interface between a simulation and an intelligent decision-maker. Section 4 discusses an application of the formalism to a problem from manufacturing.

2 Overview of AI and Simulation

2.1 Artificial Intelligence in Simulation

Since the mid-70's [Ören 1977], many attempts to use artificial intelligence to benefit simulation have been made. Much of this work has been devoted to helping investigate the problem, develop the simulation and analyze the results. These attempts are well documented as in Floss and Talavage [1988], Fishwick [1988], Rozenblit and Zeigler [1985], and Widman [1988], among others.

"Cognizant simulation" [Ören and Zeigler, 1987], which implies the existence of a knowledge base about the system being simulated, also provides the ability to explore various levels of abstraction within a single simulation or to explore various viewpoints or goals [Fishwick 1989].

Some implementations of these concepts have taken the form of the simulations, the knowledge base and the inference engine for decision-making all written entirely in procedural languages such as FORTRAN. Others have written all the requisite parts of the intelligent simulation in a traditionally AI language such as PROLOG [O'Keefe 1989]. Still another approach has been the development of hybrid languages that provide simulation constructs and statistic-gathering properties with pattern-matching abilities and reasoning capabilities [Ruiz-Meir and Talavage 1989].

Unfortunately, there seem to be almost as many solutions to the problem of integrating simulation and artificial intelligence as there are investigators into the problem. However, there is one point of agreement among all the previously cited works - the need for some intelligent agent to control the behavior of the simulation. This paper presents an alternative solution to the problem which provides a means to utilize the strengths of existing simulation and AI languages and provide greater flexibility than either one alone.

2.2 AI and Simulation Languages

The use of a single language to provide intelligent control for a simulation has some desirable aspects. This approach also has two distinct drawbacks:

1. It presupposes that the language can handle all the requirements of both the intelligent decision-maker and the simulation; and
2. It presupposes that the programmer has expertise in both simulation and AI.

Simulation languages provide elements that are specific to the construction of simulations: means to describe physical entities, attributes and relations to other entities; mechanisms for specifying the passage of time; functions and procedures to gather and compute statistical measures; and means to provide for random events. Queuing structures and event scheduling are well-defined and the typical simulation language is capable of complex numeric computations.

In contrast, the strengths of the AI languages lie in the ability to extract patterns from a complex set of facts, to reason on incomplete information, and to resolve conflicting goals. AI languages have the power of symbolic computing but are not suited for complex numeric computation.

Therefore, an ideal solution to intelligent simulation would take advantage of these language traits and strengths by keeping the physical simulation separate from the AI engine that makes the decisions. This separation also addresses the second point above.

The use of separate languages allows for separate development, testing and debugging of the individual components by programmers with expertise in the specified field. It also allows for modular development of a complex system and for interchangeability among compatible components.

These issues lead us to consider the alternative of writing the physical simulation in a simulation language and the decision-maker in an AI language and then providing an interface between them. Therefore we propose a "Standard Simulation-Knowledge Interface" (SSKI) as a formal model. Elements in the development of SSKI include:

1. How will an expert system/rule-based manager for a simulation be invoked? Will it be invoked as needed, by monitoring a variable or group of variables in the simulation? Or will it be invoked on a regular basis using the simulation timing mechanism and/or event scheduler?
2. What information is needed in order for the decision-maker (the Manager) to reach a decision? Should specific information pertaining to a particular goal or all information about the system be available to the Manager?
3. What information from the Manager needs to be returned? Will the Manager alter attributes of the simulation directly, or will it be necessary for the Manager to pass the desired values and have the simulation alter the attributes?
4. What device or system should be used for the passing of information back and forth between the simulation and the Manager? A system of shared files, a common knowledge base, or a blackboard environment?

These issues, as well as many others, must be addressed in order to design a successful interface between the simulation and its decision-maker.

3 The SSKI Formalism

3.1 General Considerations

A simulation **S** of a real system can be defined as **S** = {A, R, O, T} consisting of a set of observable attributes A and a relation R defined on A which results in a set of outputs O at some simulation time t, where $t \in T$. This definition holds for all simulations of real systems and is independent of the formal model or definition.

Similarly, a knowledge-based manager **M** can be defined as **M** = {K, C, T} consisting of predicates K along with an inference engine which acts on K at time t and yields a result c as a conclusion, where $c \in C$.

If a manager **M** is to make decisions concerning simulation **S**, then the following must hold:

a. There exists a subset $V \subseteq A \cup O$ in **S** which is necessary and sufficient for such decisions.
b. **M** can assert predicates based on V consistent with K and can derive conclusion c.
c. Conclusion c can be used as input to **S** in order to affect some changes in **S**.

Given a simulation **S** and manager **M** as outlined above, an interface **I** can be described thus:

$I = \{V, K, c, T, f, g\}$

where

$V \subseteq A \cup O$ in **S** as defined above;

$K = \{\text{predicates}\}$ in **M** as defined above;

$c \in C = \{\text{conclusions}\}$ in **M**;

$T \subset \{\text{reals}\} \ni t \in T \Rightarrow t \geq 0$ denoting time in **S**;

Define P(V) and P(K) as the power sets of V and K, respectively. Then:

f = function mapping a set of variables of S at time t_s into the predicate set of **M**:

$(f{:}P(V) \times t_s \rightarrow P(K))$;

g = function mapping a set of predicates in M at time t_m into the set of conclusions C:

$(g{:}P(K) \times t_m \rightarrow C \ni g(P(K),t_m) = c)$.

In order for the interface to be generic, t_s and t_m may assume any values in T such that $t_s \leq t_m$. However, for the purposes of this paper, the following assumption will be made:

1. $t_s = t_m$: that is, the simulation will assume the responsibility for calling the manager at time t_s and will suspend all activity until a response is made by the manager. Since t_s represents simulation time, not computational time, no simulation time will be recorded during the call-and-wait procedure.

Although the general model could support a manager which could interrupt the simulation at any time with the results of the original call, the above restriction provides for the following to hold:

2. The function $g{:}P(K) \times t_m \rightarrow C$, P(K) is defined by $P(K) = f(P(A), t_s)$: the subset of the predicates mapped back as a conclusion is precisely that result of the function f. The changes made in simulation **S** as a result of conclusion c at time t_m are a direct result of the invocation of the manager by the simulation at time t_s.

The above assumptions define the relationship between the simulation and the manager as a call-and-wait system with the manager in the role of a knowledge server to the calling simulation.

Thus the simulation itself must be defined so as to provide a means to call the manager and respond from the output of the manager even as the manager must be defined to respond to appropriate input from the simulation.

It is possible to use Hoare's [1985] description for subordinate processes to describe the relation between **S** and **M**. However this does not address the issues of what information must be communicated between these processes and what effects **M** may have upon **S**.

Within the general definition of interface **I**, some further observations and restrictions are made:

3. V (the set of variables in **S** communicated to **M**) can be partitioned into V_s and V_d where V_s represents the static, unchangeable attributes of **S** whereas V_d represents the dynamic, changeable ones. Clearly, if V_s is known to **M** at $t = 0$, then it is also known to **M** for all $t > 0$ as well. Thus there is no need to communicate this subset of V to **M** for t_{i+1} provided it has already been communicated at t_i. Conversely, since V_d at t_{i+1} is likely to differ from V_d at t_i, then V_d must be communicated to **M** on each call.
4. In order to simplify an implementation, it will be assumed that V_d can only be affected within the simulation itself. Thus the actual mechanism for altering attributes will be provided within the simulation program. This eliminates the need to provide the manager direct access to those attributes and maintains the knowledge-server position of the manager. In other words, the manager will make the recommendations for changes in the simulation variables, but it is up to the simulation to implement these changes. This recommendation is embodied in the predicate value c as described above.

4. Application Problem

4.1 Overview of Recent Work

A flexible manufacturing system (FMS) was chosen as a case study for several reasons. Foremost is the inherent complexity of planning and scheduling in an FMS. Even well documented scheduling rules, as summarized in Gupta, Gupta, and Bector [1989] and Montazeri and Wassenhove [1990], are inadequate for solving every scheduling problem.

Many recent attempts to solve scheduling problems have been made ranging from an algorithmic approach [Chan and Bedworth 1990] and knowledge-based interactive systems [Sarin and Salgame 1990] to simulation approaches [Wu and Wysk 1989]. Despite these varied approaches, an ideal solution has yet to be found.

4.2 Problem Description

In a traditional FMS scheduling problem certain assumptions are made:

1. The jobs to be processed are independent;
2. Jobs enter the system simultaneously; and
3. Jobs leave the system after completion.

With these assumptions, the goals of the scheduler are generally related to minimum makespan (minimum time to complete all jobs). These assumptions and goals generally lead to the use of a few static rules such as SPT (shortest processing time) to select the next job for processing when several jobs await service at the same machine.

In the problem chosen for this work, these three assumptions concerning jobs are not valid. Instead, the following relationships must be satisfied:

1. Parts may enter the system at any time and must follow the processing schedule as given in Table 1 below;
2. On entry, parts become work-in-progress (WIP) and remain WIP until exiting the system;
3. A part may not exit the system until it is assembled with other required parts;
4. WIP must be limited to a total of 500 parts;
5. The three types of machines (A, B and C) have unlimited queuing capacity and require a set-up time of 60 minutes whenever switched from one operation mode to another; and
6. Transportation time between machines and assembly time of completed parts is negligible.

The goal of this FMS is not to minimize the makespan of the individual parts but to devise a schedule which will maximize the number of assembled items able to exit the system in a fixed amount of time. Therefore, the application of traditional scheduling rules may not be appropriate since there must be a balance among the parts of various types in order to provide an assembled item at the exit point.

Table 1: Order of operations for each part

Pt.#	Op.#	Mch.ID	Time
1	10	A	5
	20	B	3
	30	A	3
	40	C	3
2	10	C	2
	20	A	3
	30	A	8
	40	B	15
3	10	C	3
	20	A	5
	30	B	2
4	10	C	20
	20	A	1
	30	B	3
	40	B	11
	50	C	5

The aspects of this problem making it a suitable case study for the proposed interface are threefold:

1. The problem contains various levels of decision-making, from simple deterministic conditional branches to multi-criteria, multi-conditional branches.
2. The problem is small enough to be able to investigate several possible alternative designs yet complex enough to provide interesting insights into the proposed formalism.
3. The problem has a dynamic structure in that entities can be introduced into the system, manipulated and removed from the system according to different conditions allowing for the investigation of various strategies.

Therefore the use of this factory problem will provide a means for discussing and clarifying the interface model.

4.3 SSKI for the Case Study Simulation

While many models are possible for the illustrative problem, let us consider the following structure.

Variables -

1. Parts and Machines are described as entities having the following attributes:
   A. Parts:
      a part number (PT.NUMBER)
      a batch size (PT.BATCH.SIZE)
      an imminent operation number (PT.OP)
      a per/item processing time (PT.OP.TIME)
   B. Machines:
      an identification/name (MCH.ID)
      a current operation number (MCH.OP)
      a busy/idle status (MCH.BUSY)
2. Machines have queues of batches of parts and the length of a machine's queue is stored in the variable N.MCH.QUE(MACHINE) indicating the number of batches waiting service at MACHINE.
3. The accumulations of various part-types in the system are stored in the variables P.COMPLETE(TYPE) indicating the number of completed parts of each TYPE waiting for assembly and P.WIP(TYPE) indicating the number of each TYPE currently being processed in the system.

The above variables constitute the set V_d as discussed in a previous section. There are other dynamic variables tracked by the system but these are used primarily for the reporting of statistics on the plant operation and are not necessary for the purpose of decision-making.

V_s consists of such information as the number of machines (N.MACHINES), the number of part types (N.PARTS) and the number of parts allowed in the system at any one time (MAX.PARTS).

Processes -

1. OBSERVE.MACHINES: This is the heart of this decision-oriented simulation model. At every minute of simulation time an observation is made of each of the machines. It has been assumed, for this implementation, that if a machine has become idle and has only one batch of parts in its queue, it will immediately begin working on that batch.

   This could be replaced by a global, look-ahead scheme that would survey the system for batches of parts that would be joining the machine queue and, perhaps, override the one-batch default mechanism. While this option could be readily implemented, it will not be considered in this discussion.

   If a machine is busy, nothing happens until the next observation period with respect to the observed machine.

   If the machine is idle and either has no batches in its queue or has more than one batch, then a decision must be made as to what the machine should do next. There are three basic options:
   1. Wait and do nothing.
   2. Process a new batch of parts.
   3. Process a batch from the queue.
2. MACHINE.WORK: When a decision is made to process a batch of parts on a given machine, the status of the machine is set to "busy" and the machine operation number is compared to the part operation number. If these operation numbers are different, the machine operation number is changed to match the part operation number and the total processing time for the batch is incremented by one hour. Once the batch has been processed, the PT.OP is increased by 10, the batch is filed in a temporary set and the MOVE.PART routine is called which routes the batch to the next machine and establishes the per/item operation time. The machine status is returned to "idle."
3. ASSEMBLE: When the part-operation number indicates that all machining has been accomplished, the batch of parts is available to be assembled. Although, for this particular example, the assembly process consumes no simulation time, it is advantageous to schedule ASSEMBLE as an event so that it can be given priority over starting new batches into the system.

Figure 1 shows the logical relations among these events and processes. The initialization routines and reporting processes have been omitted for simplicity. A nonstandard flow-chart symbol, the circle, has been used to denote a multiple value, complex decision block.

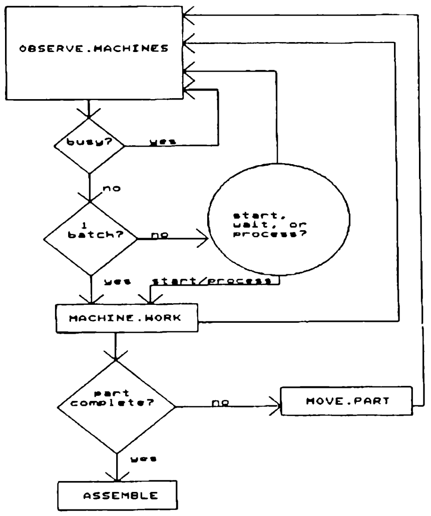


Figure 1: Process flowchart for application problem

While Figure 1 shows the logical connections among the major event and processes in the simulation model of the factory, it does not show the complexity of the logic involved. Figure 2 shows the flow of batches of parts through the three machining centers as a network model. The conditional branches are based on the fixed routings of the various part types. In this figure, the multiple branches into the machines indicate the complexity in the decision as to what should happen next at a given machine.

Let us refer to these entry points as Decision Points. A given decision point corresponds to the logical questions of Figure 1 as to whether to start a part, process a part or have that machine wait and do nothing until the next observation period. Although the logical diagram asks the same questions as the decision points in the network diagram, it is not obvious from Figure 1 how complex these decisions might be. For example, the decision point at machine C not only concerns whether to start a batch of parts at that machine but which part type to start.

Whichever diagrammatic model is used, it can be seen that the number of complex decisions to

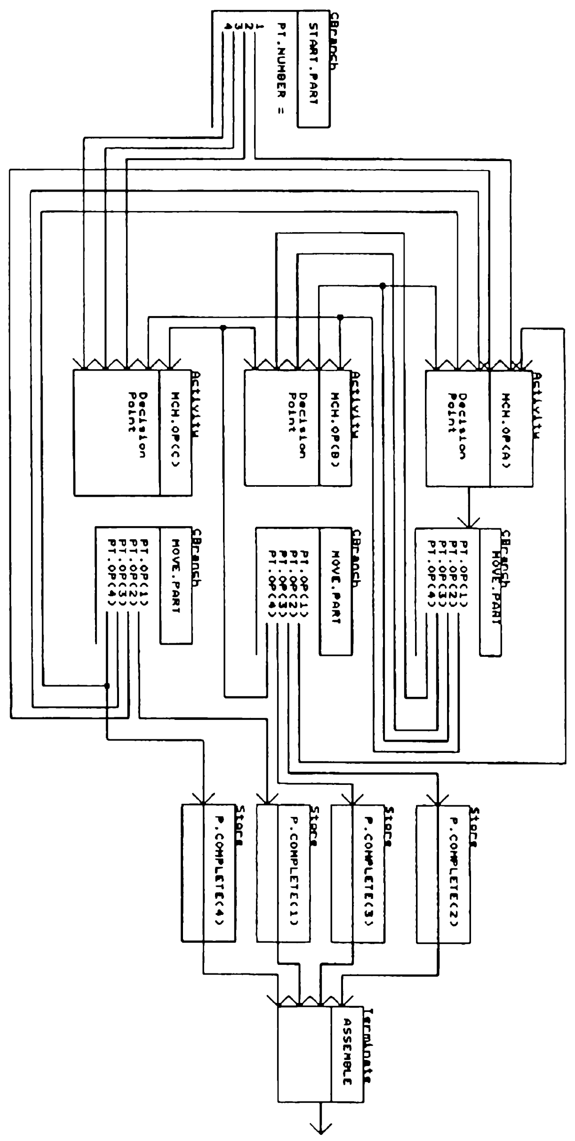


Figure 2: A network model of the application problem

be made is constrained by:

1. the number of processes and events in the simulation, and
2. the number of physical interactions in the system being modeled.

In the given model, only one process (OBSERVE.MACHINES) requires some higher-level decision-making and it can be invoked for each of the three machines. Therefore, there are at most three distinct decision points to be considered in this model.

In general, if there are p processes that require some high level decision-making and each process can be invoked for d_p distinct cases, then there are at most $D = \Sigma\ d_i$, $i = 1, 2, ..., p$, decision points.

Now since each decision point requires the intervention of the intelligent manager, the function f: $P(V) \times T \rightarrow P(K)$ can be restricted to at most D distinct subsets of V, each subset consisting of those variables necessary to make the decision in question.

4.4 Variable Selection

In the design of the interface, one must be cognizant not only of the physical attributes of the system modeled by the simulation but also of the decision-making criteria demanded by the knowledge-server.

In the worst-case scenario, under the circumstances that the interface designer either does not know the decision criteria or wishes to give the knowledge-server as much flexibility as possible, then each of the D subsets mapped by the function f can be equivalent to the entire set V. In other words, on each call to the knowledge-server, all the variables concerning the simulation can be communicated to the manager.

One restriction that could reduce communication costs would be to make V_s (the static variables) available to the knowledge-server at some point during the simulation before a decision point is reached and enable the knowledge-server to retain this information for the duration of the run. This could be accomplished in two ways:

1. encode the static information into the data base of the knowledge-server; or
2. pass the information to the knowledge-server as part of the initialization or set-up routine of the simulation at simulation time $t = 0$.

For the illustrative problem, it was deemed more efficient to include the static information as facts in the manager's data base. The benefits of this approach can be disputed, especially if a purpose of the simulation is to investigate the behavior of a system given different static parameters for separate runs (for example, changing the number of machines).

If this restriction is made, then even in the worst case, the amount of information to be communicated at each of the D decision points is less than the entire set of variables. Indeed, each of the D subsets will now be a subset of V_d, the dynamic variables.

To illustrate the different subset requirements for the different decision points, consider the following:

Example 1.

"If the machine's queue is empty and
if the Machine ID = A and
if the total number of parts in the system is less than the maximum allowed,
then start a batch of Part 1."

The amount of information needed to reach a conclusion for Machine A is quite different from a similar situation regarding Machine B:

"If the machine's queue is empty and
if the Machine ID = B,
then wait."

Performing some preliminary screening of variables before the manager is invoked can help to reduce the number of variables necessary. If no such screening is done, then it may be necessary to pass all the variables in each case.

In the factory example, if the decision to be made concerning a particular machine is based on some local property (information concerning that machine only as opposed to information from all the machines), then the set of variables can be restricted accordingly. In this case, pre-screening by MCH.ID restricted the information being passed to that which concerns only the batches of parts in the machine's queue.

Additional restrictions on the amount of data communicated can be made if more is known about the nature of the decision-making process. For a comparison as to the varying amount of data needed, consider the two following decision criteria for a given machine having three batches in its queue (B1, B2, B3).

Example 2.

1. "If the queue length > 1
   then process the batch having the shortest imminent processing time."
2. "If the queue length > 1
   then process the batch whose part type has the fewest items completed."

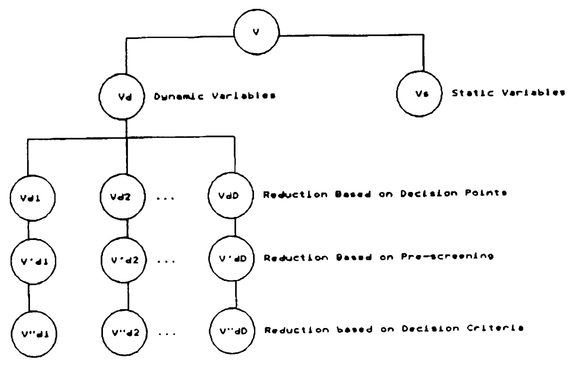
Quite clearly the two different selection strategies require not only different information, but different amounts of information. In case (1), the manager must know the batch size and the per/item processing time for each batch in the queue in order to reach a decision. Therefore the minimal subset would be: {BATCH.SIZE(B1), OP.TIME(B1), BATCH.SIZE(B2), OP.TIME(B2), BATCH.SIZE(B3), OP.TIME(B3)}.

In case (2) however, each part type must be known as well as how many parts have been completed. Therefore the minimal subset would be: {PART.NUMBER(B1), PART.NUMBER(B2), PART.NUMBER(B3), P.COMPLETE(1), P.COMPLETE(2), P.COMPLETE(3), P.COMPLETE(4)}. More complicated decision-making, involving perhaps multi-level comparisons, would require more information. An example of such a multi-level process, requiring the union of the above sets could be:

"If queue > 1,
then process the batch having the shortest imminent processing time.
If there is a tie among batches,
break the tie by processing the batch whose part type has the fewest number of completed parts."

If the goal of the interface designer is to minimize the variable subsets to be communicated from the simulation to the manager, then the interface designer must know both what pre-screening has been done in the simulation (Example 1) and what decision-criteria is necessary in the manager (Example 2).

Figure 3 illustrates the variable-selection process that leads to minimal subsets for each decision point.


V
Vd
Dynamic Variables
Vs
Static Variables
Vd1
Vd2
VdD
Reduction Based on Decision Points
V'd1
V'd2
V'dD
Reduction Based on Pre-screening
V''d1
V''d2
V''dD
Reduction based on Decision Criteria

where $V''_{di} \subset V'_{di} \subset V_{di} \subseteq V_d \subset V$ for $i = 1,2,\ldots,D$ and
where $V_d \cap V_s = \emptyset$ and $V_d \cup V_s = V$.

Figure 3: Reduction of variable subsets

4.5 Restricting the f-Function

In the general description of the interface, F is defined on the power set of V (simulation variables) together with the set T (simulation time). Once we have isolated D decision points, each with its own specified properties and restrictions, we can now restrict the f-function accordingly. Therefore we can now define f as follows:

$$f{:}P(V)\times T=\begin{cases} f(V_s,0) & \\ f(V_{di},t) & \text{for } i=1,2,\ldots,D;\ t\geq 0 \\ \emptyset & \text{otherwise} \end{cases}$$

The size of set V_d may vary depending on whether entities are created and/or destroyed during the simulation run. It is possible to partition V_d into those variables which are always present, V_p, and those which are temporary, V_t. To illustrate these two concepts, consider the variables of interest in the example problem:

V_p = {MCH.ID(MACHINE)), MCH.OP(MACHINE), MCH.BUSY(MACHINE), N.MCH.QUE(MACHINE), P.COMPLETE(TYPE), P.WIP(TYPE) | MACHINE = 1,2,3 and TYPE = 1,2,3,4} and

V_t = {PT.NUMBER(PART), PT.BATCH.SIZE(PART), PT.BATCH.NUMBER(PART), PT.OP(PART), PT.OP.TIME(PART) | PART = 1, 2, ... }.

In order to relay information to the manager concerning subsets of V_t, it will be necessary to ascertain the number of PARTS in question and be able to identify which PARTS are needed.

In most cases, temporary entities such as PARTS are moved through the system by being removed from one queue and placed in another. Therefore it is possible to isolate these entities by examining the various queues in the system. In this factory problem, all PARTS reside in a MACHINE queue from the moment of creation until they exit from the system via the ASSEMBLE event and are destroyed. PARTS are moved from one MACHINE queue to another by the MOVE.PARTS routine which places them in the system-owned set called TEMP. However, MOVE.PARTS consumes no simulation time. So the PARTS are considered as residing in the TEMP set for no simulation time and can be viewed as belonging to MACHINE queues only.

Since the number of PARTS in a given MACHINE queue can vary widely, the transmission of any subset of V_t relating to these PARTS is dependent on the state of the simulation at the time of the communication.

4.6 Restricting the g-Function

This implementation assumes that only the simulation can change values of the variables. Thus, it is assumed that the set of conclusions C is of a form which can be used as input into the simulation in the same way that information from the simulation is used as input to the manager. In general, there will be less variability in the amount of information needed to be passed from the managed to the simulation. It is possible that this information will be restricted to a simple logical value, but it is likely that the conclusion will contain some specific instructions to the simulation. In the factory problem, if the conclusion is made to start a batch of parts on Machine C, then information concerning which of the possible parts must be communicated to the simulation.

5 Conclusions

In order for simulation to benefit from the advances in artificial intelligence, it is necessary to provide a mechanism for them to interact. Rather than merge the two distinct activities of simulation and control into a single, continuous process, this work proposes to decouple the functions, allowing information to flow between them as needed. This decoupling will enable the development of the separate functions individually and within the framework of the language of best fit. It will also enable incremental, modular changes to be made which will not affect the behavior of the other function.

To this end, this paper describes a formal model for a standard simulation-knowledge interface (SSKI) and shows how it may be applied to a scheduling and planning problem in the manufacturing area. A simulation of the physical plant, the machines and the parts was written in SIMSCRIPT II.5 [1987] and a manager was written in NASA's C-based language CLIPS [1988]. Subset reduction was achieved through partitioning the set V and encoding V_s directly into the rule-based manager. V_d was reduced by using decision-point restrictions and pre-screening of certain variables, namely MCH.ID and N.MCH.QUE. Preliminary work has shown that this interface model can achieve the desired results for modularity and flexibility by providing for interchangeability among managers and/or simulations and allowing for incremental modifications in both.

ACKNOWLEDGMENTS

This material is based upon work supported by a National Science Foundation Graduate Fellowship.

REFERENCES

Chan, Ding-Yu, and David D. Bedworth. 1990. Design of a scheduling system for flexible manufacturing cells. *International Journal of Production Research* 28:11, 2037-2049.

CLIPS reference manual. 1988. Artificial Intelligence Section. Lyndon B. Johnson Space Center.

Fishwick, Paul A. 1988. Qualitative simulation: fundamental concepts and issues. In *AI and simulation: the diversity of applications*, ed. Troy Henson, 25-31. San Diego: Society for Computer Simulation.

Fishwick, Paul A. 1989. Process abstraction in simulation modeling. In *Artificial intelligence, simulation and modeling*, ed. Lawrence Widman, Kenneth Loparo and Norman Nielsen, 93-131. New York: John Wiley & Sons, Inc.

Floss, Peter, and Joseph J. Talavage. 1988. A knowledge-based design procedure for flexible manufacturing systems. In *AI and simulation: the diversity of applications*, ed Troy Henson, 251-255. San Diego: Society for Computer Simulation.

Gupta, Yash P., Mahesh C. Gupta, and C. R. Bector. 1989. A review of scheduling rules in flexible manufacturing systems. *International Journal of Computer Integrated Manufacturing* 2:6, 356-377.

Hoare, C. A. R. 1985. *Communicating sequential processes*. Englewood Cliffs, New Jersey: Prentice-Hall International.

Montazeri, M., and L. N. Van Wassenhove. 1990. Analysis of scheduling rules for an FMS. *International Journal of Production Research* 28:4, 785-802.

O'Keefe, Robert M. 1989. The role of artificial intelligence in discrete-event simulation. In *Artificial intelligence, simulation and modeling*, ed. Lawrence Widman, Kenneth Loparo and Norman Nielsen, 359-379. New York: John Wiley & Sons.

Ören, Tuncer I. 1977. Simulation - as it has been, and should be. *Simulation* 29:5, 182-183.

Ören, Tuncer I., and Bernard P. Zeigler. 1987. Artificial intelligence in modeling and simulation: directions to explore. *Simulation* 48:4, 131-134.

Rozenblit, Jerzy W., and Bernard P. Zeigler. 1985. Concepts for knowledge-based system design environments. In *Proceedings of the 1985 Winter Simulation Conference*, ed. D. T. Gantz, G. C. Blais, and S. L. Solomon, 223-231. Institute of Electrical and Electronics Engineers, San Francisco, California.

Ruiz-Meir, Sergio, and Joseph Talavage. 1989. A hybrid paradigm for modeling of complex systems. In *Artificial intelligence, simulation and modeling*, ed. Lawrence Widman, Kenneth Loparo and Norman Nielsen, 381-395. New York: John Wiley & Sons, Inc.

Sarin, S. C., and R. R. Salgame. 1990. Development of a knowledge-based system for dynamic scheduling. *International Journal of Production Research* 28:8, 1499-1512.

SIMSCRIPT II.5 programming language. 1987. La Jolla, California: CACI Products Company.

Widman, Lawrence E. 1988. Knowledge-based fault identification and 'what if' simulation in symbolic dynamic systems models. In *AI and simulation: the diversity of applications*, ed. Troy Henson, 89-94. San Diego: Society for Computer Simulation.

Wu, Szu-Yung David, and Richard A. Wysk. 1989. An application of discrete-event simulation to on-line control and scheduling in flexible manufacturing. *International Journal of Production Research* 27:9, 1603-1623.

AUTHOR BIOGRAPHIES

CAROL S. RUSSELL is a doctoral student in the Computer Science and Engineering Department at the University of Louisville. She is currently employed as a lecturer in mathematics at Indiana University Southeast, New Albany, Indiana.

Adel Said Elmaghraby is an Associate Professor of Engineering Mathematics And Computer Science at the University of Louisville. His research areas include Simulation and Artificial Intelligence, Robotics and Automation. He is the founding editor of the ACM/IEEE joint Simulation Newsletter and chairman of the IEEE Computer Society Technical Committee on Simulation (TCSIM).

James H.Graham is currently Henry Vogt Professor of Computer Science and Engineering at the University of Louisville. His research interests include robotics, artificial intelligence and distributed computing. He is the editor of Safety, Reliability and Human Factors in Robotic Systems, published by Van Nostrand Reinhold in 1991.