Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

OBJECT ORIENTED TOOLKITS FOR SIMULATION PROGRAM GENERATORS

Onur M. Ulgen
Timothy Thomasma
Youyi Mao

Production Modeling Corporation
One Parklane Blvd., Suite 1604 West
Dearborn, MI 48126

Industrial and Systems Engineering Dept.
University of Michigan-Dearborn
4901 Evergreen, Dearborn, MI 48128

ABSTRACT

Simulation Program Generators (SPGs) developed using object-oriented environments may be designed to have a number of capabilities not available in traditional SPGs. One such capability is the description of an object's behavior from scratch without any programming when such behavior does not exist in the knowledgebase of the SPG. SmarterSim is an SPG that is currently being developed to provide the user a number of object-oriented tools including libraries of elemental operations, actions/requests, states, and rules and a number of function block templates to create new behavior using the existing primitive elements. The behavior of objects in a robotic manufacturing cell is described using SmarterSim. Like its predecessor SmartSim, SmarterSim supports a hierarchical, modular modeling for building of subsystems (coupled models) from model objects (atomic models) using subsystem management features. The paper also discusses the use of object oriented environments for traditional SPGs and simulation programming languages.

1. INTRODUCTION

There are several problems with the currently available programmer-free icon-based simulation environments (Conway and Maxwell 1987, Gilman and Watremez 1986, Tumay 1987, Kilgore and Healy 1987, Harshell and Dahl 1988). They are not extensible. If an object must be modeled that is not provided for by the environment, then the environment cannot be used. A programmer has to be involved to add the object to the simulation program and this generally requires a large amount of programming. The same is also true if a new behavior is to be modeled for a currently available object. Programmer-free simulation environments based on object-oriented languages such as Smalltalk (Knapp 1986, 1987, Ulgen and Thomasma 1986, 1987a, 1987b, Thomasma and Ulgen 1987, 1988) are much easier to extend due to their class/object/message and inheritance structures but still require a programmer to incorporate the new behavior or object into the environment.

This paper will describe a programmer-free, icon-based intelligent simulation environment called SmarterSim that is extensible. Like its predecessor SmartSim (Ulgen and Thomasma, 1986, 1987a, 1987b, Thomasma and Ulgen 1988), it is based on the object-oriented programming language Smalltalk, but it also provides an environment for description of new objects and new behaviors without any programming. The environment we describe in this paper has a structure analogous to the structure of the REGENT (Encarnacao and Schlechtendahl 1983) integrated system for computer aided design in mechanical engineering (Figure 1). Figure 2 shows how this would be adapted to the situation of simulation modeling in CAD for manufacturing systems engineering. Simulation models at level 1 of Figure 2 are the end results of simulation programming. At present, these are built, on the whole, from scratch. The program generators that exist are at level 2 of Figure 2, except that they do not allow logic governing behavior of objects to be modified. The toolkits and environments we discuss in this paper are intended to be at level 3 of Figure 2.

A fundamental feature of SmarterSim is the capability for user to review and modify the material handling control logic. Methods for specifying control logic include ladder diagrams (Lukas 1986, pp. 55-57), problem oriented languages, function block diagrams (Lukas 1986, pp. 49-60), operation networks (Muller 1985), time-position diagrams (Muller 1985), rule sets with position diagrams (PROMOD 1988), activity cycle diagrams (Hutchinson 1981), Petri nets (Peterson 1981), and state transition diagrams (Edelberg 1988). Petri nets are the most satisfactory of these, but in order to express decision rules, the states in the Petri nets must be understood as states of the entire system. No mechanism is provided for combining states of individual components to form full-system states. Also, state transitions are most naturally communicated in terms of animation: changes in visual characteristics over time. The state transition types of

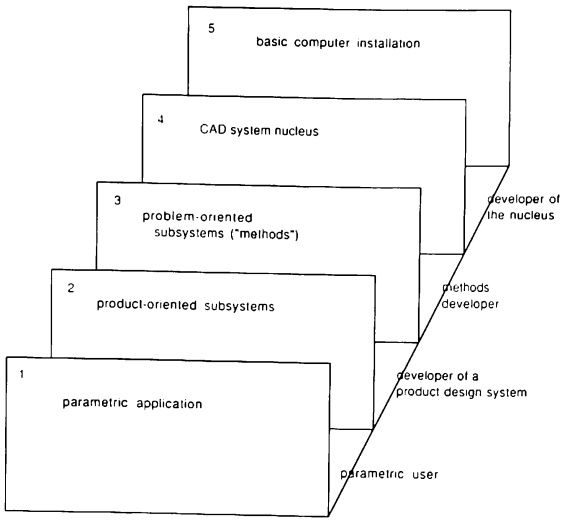

Figure 1: Integrated system for computer-aided design in mechanical engineering (Encarnacao and Schlechtendahl 1983)

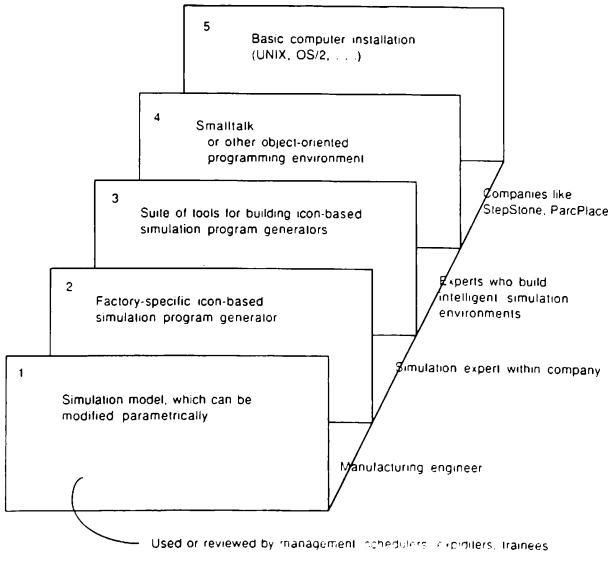

Figure 2: Computer-aided design for simulation modeling in manufacturing systems engineering

diagrams communicate relationships between states by means of spatial locations. Other specification techniques are stronger than Petri nets in some respects. Function block diagrams are better for indicating decision logic. Pure position diagrams (without rule sets) describe systems with the least amount of abstraction, in terms that are closest to the real system, but they are very incomplete. Time-position diagrams give the clearest static view of timing relationships.

SmarterSim uses function block diagrams, combined with state and elemental operation tables, to specify the behavior of objects in the model. Once the object is defined fully with its behavioral characteristics, an icon is assigned for it and is archived. Copies of it can be made and it can be coupled with other icons to form subsytems (Thomasma and Ulgen, 1988).

In the next section of the paper, we give a classification of object oriented simulation environments and briefly describe the tools required to build them. In the following section, we describe SmarterSim. A robotic manufacturing cell is used to describe some of the features of SmarterSim. The last section of the paper gives the advantages and disadvantages of the intelligent simulation environments suggested in the paper.

OBJECT ORIENTED TOOLKITS FOR SPGS

Object oriented simulation environments can be useful at three different levels of application. At the highest level of application, an object oriented environment can be built around a traditional SPG ,e.g., MAST (Lenz 1983), GENTLE (Ulgen 1983), PROMOD (PROMOD 1989), SIMFACTORY (Tumay 1987), that is built using a procedure oriented language such as FORTRAN or C. Figure 3 shows a structure for an object based interface for a traditional SPG. The simulator of the traditional SPG in Figure 3 is assumed to be based on the SIMAN simulation language and the pre-, co- and post-processors may be written in FORTRAN or C. The object oriented environment around the SPG is composed of a number of graphics interfaces based on a common graphics interface. The main advantages of such an environment include reduction in time in training the user, model development, and model analysis. The features required from the environment include copying and archiving of icons (icons may represent operations, storages, etc.); grouping of icons into subsytems and copying and archiving of subsystems (subsystems may represent similar operation and storage sequences, repair loops, etc.); icon-objects having

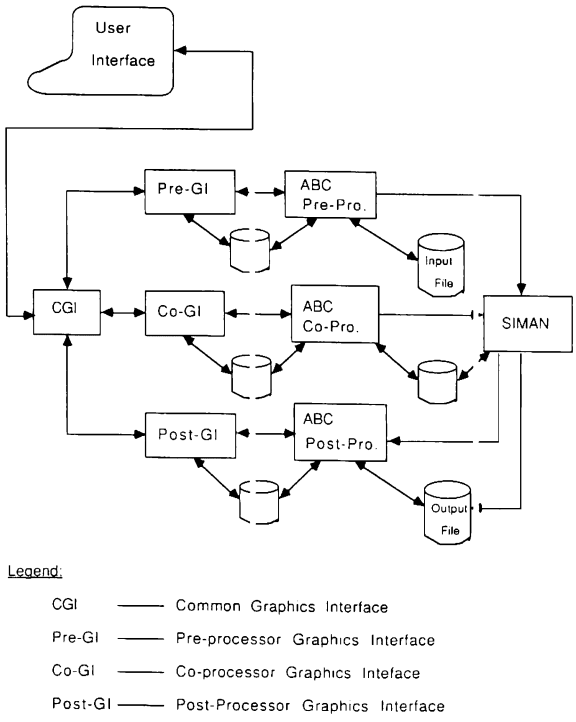

Figure 3: Architecture of an intelligent simulation environment for a traditional SPG

close resemblance to real objects in rough detail (e.g., machine icon, robot icon, conveyor icon, etc.).

The object oriented environment for a traditional SPG may be custom-built using an object oriented language such as Smalltalk or it may be built using generic object oriented tools designed for this purpose. The generic tools required to build such an environment are shown in Figure 4. These tool kits would allow the user to assemble new simulation objects, rather than program them, out of objects and tools that are designed for this purpose. In other words, these tool-kits are programmer-free interfaces to build user-friendly graphical interfaces for traditional SPGs. The following is a list of objects that may be provided with the toolkit of Figure 4 with a brief description.

part browser: allows different kinds of parts to be archived and reused.

object/subsystem browser: allows simulation objects and subsystems to be archived and reused.

icon: provides all the visual display for the simulation objects.

menu: provides all user interfacing capabilities for the simulation objects.

behavior: includes all the control logic specifying the object's states, state transitions, time advance, etc.

tally: collects statistics about the execution of the simulation and knows how to carry out statistical analysis and present the information graphically.

subsystem: allows simulation objects to be grouped and then viewed and interacted with as a single entity in the model.

table: provides a capability for looking at particular sets of statistics or parameter settings in tabular form.

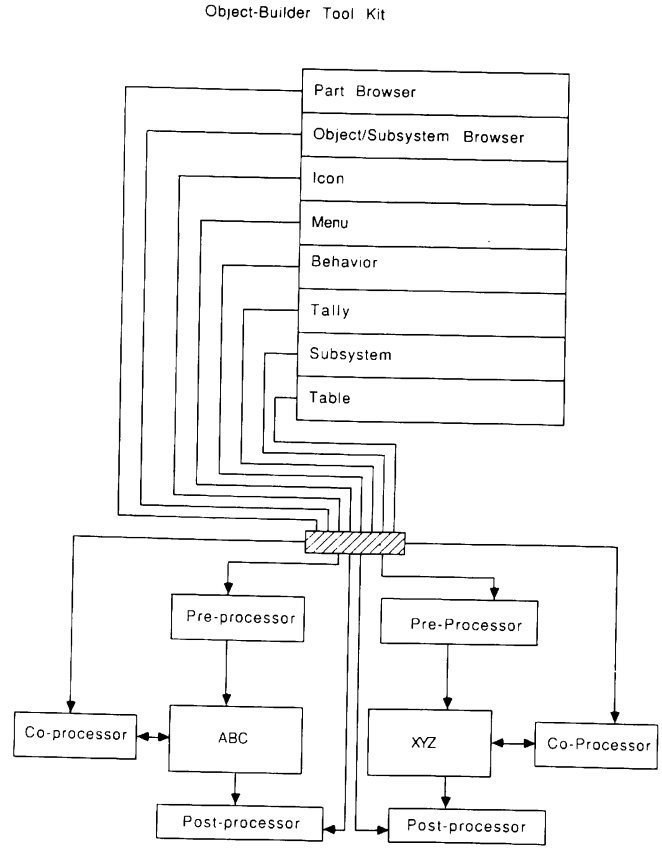

Figure 4: A generalized object-builder tool kit for building intelligent simulation environments for simulation programs and simulation program generators

At the second level of application of an object oriented simulation environment, one can build an object oriented SPG around a traditional simulation language such as SIMAN, SLAM, SIMSCRIPT or GPSS. Figure 5 shows an object oriented SPG around a SIMAN based simulator. The pre-, co-, and post-processor graphics interfaces are all based on object oriented paradigm. Similar to the previous graphics interfaces, they can be custom-built or may be built using generic object oriented tools described above. The same advantages as above are also expected of this object oriented environment.

At the third or lowest level of application, an object oriented environment is built around an object oriented simulator. At this level of application, one may be able to create objects and behaviors not previously stored in the knowledge base of the environment. The SmarterSim environment described in the next section provides such a medium for a programmer-free, icon-based, and extensible simulation environment.

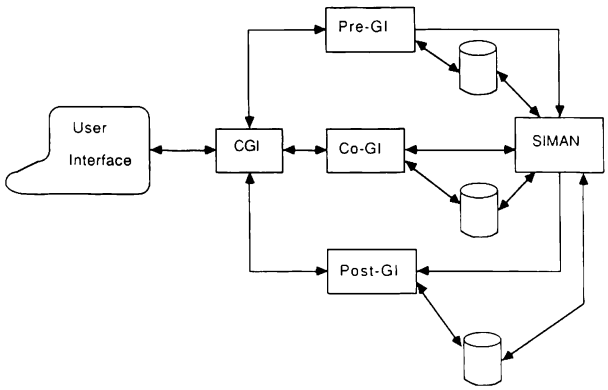

Figure 5: Architecture of an intelligent simulation environment for a simulation language

SMARTERSIM SIMULATION ENVIRONMENT

SmarterSim simulation environment provides a number of tools for defining new objects and behaviors, including tables/libraries of elemental operations, states, rules, and requests. The heart of the SmarterSim environment is the description of behavior which is defined by choosing elemental operations, requests and rules and then relating them using function block diagrams.

An elemental operation is an indivisible activity that requires some amount of time to accomplish. Each elemental operation corresponds to a state which is the state of undergoing that operation. When an operation is completed, a new operation or state is started, based on the rules associated with the completion of the previous elemental operation.

Determination of when an operation begins or ends can be done in one of the following three ways;

(i) according to an event schedule, (A rule can schedule the beginning or ending of an elemental operation at some time in future.)

(ii) according to a rule associated with end of previous operation,

(iii) according to a rule associated with a request.

Behaviors also include requests. A request has a name, by which the simulation object can recognize it. A request triggers a rule at the object that receives it. In general, each simulation object is capable of sending some requests and each is capable of responding to some requests.

Rules defining behavior are similar to rules in an expert system. The lefthand side contains a logical condition in which predicates are descriptions of states of simulation objects and/or requests. The righthand side consists of messages. These messages can be of three kinds, as given below:

(i) requests sent to other simulation objects,

(ii) immediate start of new elemental operations,

(iii) scheduled start or end of elemental operations or scheduled requests.

Rules are fired at the ends of elemental operations.

EXAMPLE: A ROBOTIC MANUFACTURING CELL

A simple robotic manufacturing cell (Medeiros and Sadowski 1983), shown in Figure 6, is used to demonstrate some of the above features of SmarterSim. The cell contains a robot, a machine, an input point and an output point. When a piece is picked up by the robot at the input point, it is taken to the machine where the robot holds the piece for processing. At the end of processing, the piece is taken to the output point and it is put down if the output point is empty. Otherwise, the robot waits for the output point to be emptied. When the piece is placed down at the output point, the robot moves to the input point. If there is a piece at the input point, the robot picks it up and travels to the machine. Otherwise, it waits empty and idle at the input point until a piece arrives to the input point.

SmarterSim simulation environment provides a library of states, elemental operations, and rules to the user. Table 1 gives the states, rules and elemental operation libraries for the robot cell. Some of the elements in these libraries may not be available initially but they will be automatically added to the libraries as they are defined. Figures 7 and 8 show some of the rules for robot and machine, respectively. Rule 1 for robot, "Start Picking Up a Piece," is triggered when a request is made for the robot to start picking up a piece. The rule states that the elemental operation "Picking up a pc" will begin if all the following three conditions are true:

(i) robot is empty and idle at the input point,

(ii) a piece is waiting and ready to be picked up at the input point, and

(iii) a request has been made to start picking up a piece by the robot.

On the other hand, according to the robot rule 2, the elemental operation "Moving to Mach" begins when the robot sends a request to itself to start moving to machine.

Rules are fired at the ends of elemental operations. Figures 9 and 10 show the elemental operations of robot and machine, respectively. In Figure 9, when the "Picking Up of a Piece" elemental operation ends, it triggers the "Start Moving to Machine" rule whose right-hand side may be executed if all the conditions are true for that rule. One should note that, at the end of an elemental operation, rules of different objects may be triggered. For example, when the robot ends the moving the piece to machine operation, two rules are triggered. One is for the robot to tell it to start holding for processing and

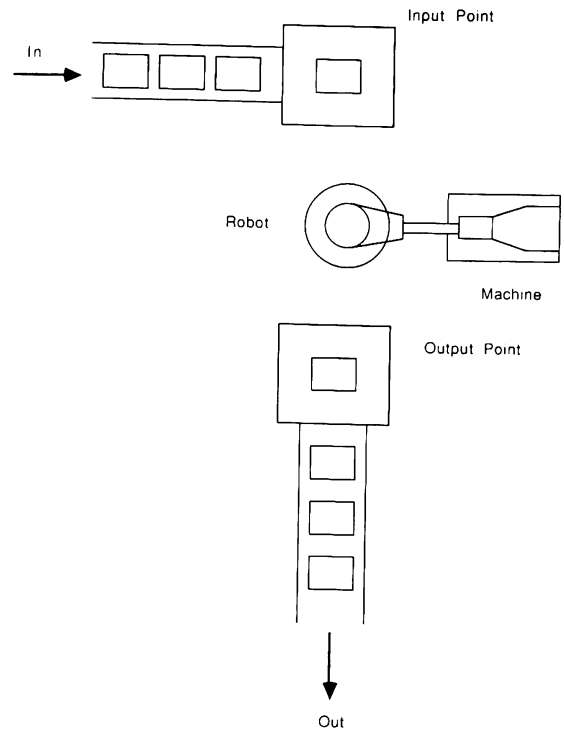

Figure 6: A simple robot manufacturing cell

the other one is for the workstation to start processing the piece.

SmarterSim function block diagrams as given in Figures 7 through 10 can be developed easily by pick and drop keys and using the library information available on the rules and elemental operations.

In a similar way, one may also create new types of objects for the model and archive them.

CONCLUSIONS

In this paper, we have described the different types of object oriented simulation environments. We also briefly described some of the main features of SmarterSim.

Object-oriented programming technology can be used in various ways in simulation modeling. It provides a powerful mechanism for adding an object-oriented world-view (Burns and Morgeson 1988) to traditional SPGs by means of which the user can construct and visualize simulation models. New SPGs

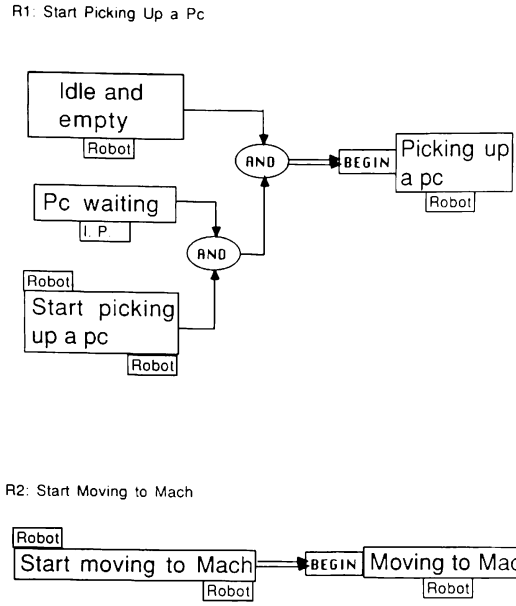

Figure 7: Rules for robot (partial)

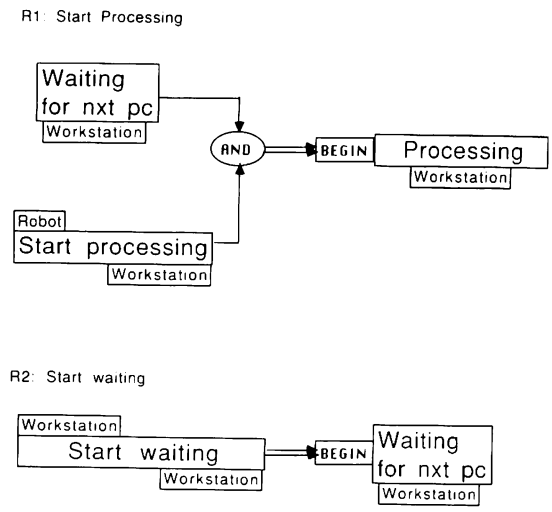

Figure 8: Rules for machine

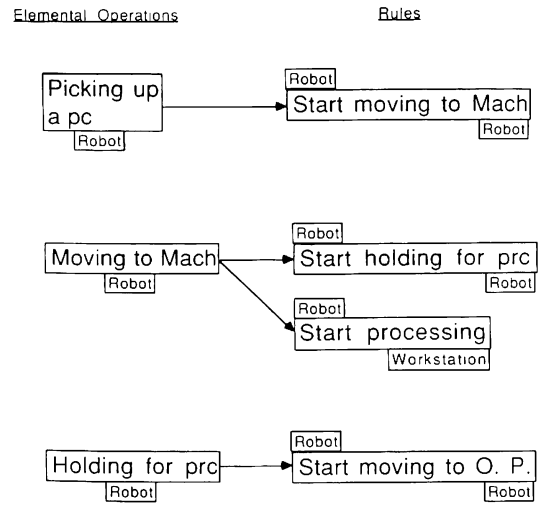

Figure 9: Elemental operations for robot (partial)

can be constructed using object-oriented programming as well. The advantages of doing this are the increased programmer productivity that results from using object-orientefd programming languages and the natural connection that exists between object-oriented programming languages and the object-oriented simulation world-view. Toolkits can be developed to further assist in the construction of SPGs.

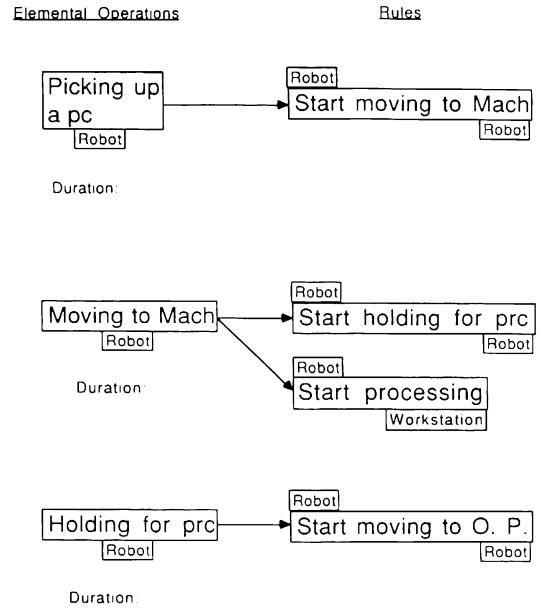

Figure 10: Elemental operations for machine

TABLE 1: SMARTERSIM LIBRARIES FOR STATES, ELEMENTAL OPERATIONS, AND RULES

STATES
Idle and empty
Holding for prc
Moving to Mach
Moving to O. P.
Moving to I. P.
Idle and full
Picking up a pc
Putting down a pc

ELEMENTAL OPERATIONS
Holding a pc for prc
Moving to Mach
Moving to O. P.
Moving to I. P.
Picking up a pc
Putting down a pc

RULES
R1: Start picking up a pc
R2: Start moving to Mach
R3: Start holding for prc
R4: Start moving to O. P.
R5: Start putting down a pc
R6: Start waiting for empty O. P.
R7: Start moving to I. P.
R8: Start waiting for a pc at I. P.

If the SPG generates simulation programs in an object-oriented programming language, a further advantage results (most SPGs generate simulation programs in regular simulation languages like GPSS or SIMAN). The simulation programs and SPGs become extensible, either with no programming, which SmarterSim attempts to support, or with minimal programming in a language like Smalltalk, as with SmartSim.

It has become clear that object-oriented programming has an important role to play in simulation modeling. Much work is still required before we will know how to best realize the potential benefits of object-oriented programming in simulation practice. In this paper we have outlined some approaches that we consider likely to prove fruitful.

REFERENCES

Burns, J. R. and Morgeson, J. D. (1988). An object-oriented world-view for intelligent, discrete, next-event simulation. Management Science, 34, 12, 1425-1440.

Conway, R. and Maxwell, W. (1987). Modeling asynchronous material handling in XCELL+. Proc. of the 1987 Winter Simulation Conference, 202-206.

Edelberg, A. (1988). The finite state program: A software design for real-time and event-driven programs with CASE implementation. Programmer's Update, Nov./Dec., 60-70.

Encarnacao, J. and Schlechtendahl, E. G. (1983). Computer Aided Design. New York: Springer-Verlag.

Gilman, A. and Watremez, R. M. (1986). A tutorial on SEE WHY and WITNESS. Proc. of 1986 Winter Simulation Conference, 178-183.

Harshell, J. and Dahl, S. (1988). Simulation model developed to convert production to cellular manufacturing layout. Industrial Engineering, 40:12, 40-45.

Hutchinson, G. K. (1981). The automation of simulation. Proc. of 1981 Simulation Conference, 489-495.

Kilgore, R. A. and Healy, K. J. (1987). Animation design with CINEMA. Proc. of 1987 Winter Simulation Conference, 261-268.

Knapp, V. E. (1986). The smalltalk simulation environment. Proc. of 1986 Winter Simulation Conference. 125-128.

Knapp, V. E. (1987). The smalltalk simulation environment, part II. Proc. of 1987 Winter Simulation Conference, 146-151.

Lenz, J. E. (1983). MAST: A simulation tool for designing computerized metalworking factories. Simulation, 40, 51-58.

Lukas, M. P. (1986). Distributed Control Systems: Their Evaluation and Design. Van Nostrand Reinhold, New York.

Medeiros, D. J. and Sadowski, R. P. (1983). Simulation of robotic manufacturing cells: A modular approach. Simulation, 3-12.

Muller, W. (1985). Integrated Materials Handling in Manufacturing. IFS Springer-Verlag, New York.

Peterson, J. L. (1981). Petri Net Theory and the Modelling of Systems. Prentice-Hall, Englewood Cliffs, New Jersey.

PROMOD, (1989). PROMOD User's Manual. PROMOD Corporation, Orem, Utah.

Tumay, K. (1987). Factory Simulation with animation: the no programming approach. Proc. of 1987 Winter Simulation Conference, 258-260.

Thomasma, T. and Ulgen, O. M. (1987). Modeling of a manufacturing cell using a graphical simulation system based on smalltalk-80. proc. of 1987 Winter Simulation Conference, 683-691.

Thomasma, T. and Ulgen, O. M. (1988). Hierarchical, modular simulation modeling in icon-based simulation program generators for manufacturing. Proc. of 1988 Winter Simulation Conference, 254-262.

Ulgen, O. M. (1983). GENTLE: GENeralized Transfer Line Emulation. Simulation in Inventory and Production Control. 25-30.

Ulgen, O. M. and Thomasma, T. (1986). Simulation modeling in an object-oriented environment using smalltalk-80. Proc. of 1986 Winter Simulation Conference, 474-484.

Ulgen, O. M. and Thomasma, T. (1987a). Graphical simulation using smalltalk-80. Proc. of SAE/ESD International Computer Graphics Conference, 317-326.

Ulgen, O. M. and Thomasma, T. (1987b). A graphical simulation system in smalltalk-80. Simulation in CIM and Artificial Intelligence Techniques, Simulation Councils, Inc., 53-58.

AUTHORS' BIOGRAPHIES

TIMOTHY THOMASMA is an Assistant Professor of Industrial and Systems Engineering at the University of Michigan-Dearborn and a Senior Consultant at the Production Modeling Corporation. He received his Ph.D. in Industrial and Operations Engineering from the University of Michigan in 1983. His present research and consulting interests include computer aided design of manufacturing systems, information systems, and software. Dr. Thomasma is a member of IEEE-CS, IIE, SIAM, CASA-SME.

ONUR M. ULGEN is an Associate Professor of Industrial and Systems Engineering at the University of Michigan-Dearborn and a Senior Consultant at the Production Modeling Corporation. He received his Ph.D. in Industrial Engineering from Texas Tech University in 1979. His present research and consulting interests include design of simulation program generators, scheduling of manufacturing systems, and forecasting. Dr. Ulgen is a member of IIE, SCS, TIMS/ORSA, and IEEE-SMC.

YOUYI MAO is a graduate student at the Industrial and Systems Engineering Department of the University of Michigan-Dearborn. He received his B.Sc. degree in Mechanical Engineering from Hefei University of Technology, China. His research interests include object oriented programming, simulation, and modeling of control systems of material handling equipment.