Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

EXPERIENCES IN DEVELOPING AN OBJECT-ORIENTED MODELING ENVIRONMENT FOR MANUFACTURING SYSTEMS

Chuda B. Basnet
Phillip A. Farrington
David B. Pratt
Manjunath Kamath

School of Industrial Engineering
and Management
Oklahoma State University
Stillwater, Oklahoma 74078

S. Cem Karacal

Industrial and
Manufacturing Engineering
Rochester Institute of Technology
Rochester, New York 14623

Terrence G. Beaumariage

Department of Industrial and
Management Systems Engineering
Arizona State University
Tempe, Arizona 85287

ABSTRACT

This paper describes the research efforts in the Center for Computer Integrated Manufacturing at the Oklahoma State University to develop an object-oriented modeling environment. After providing the underlying motivation for the research effort, some of the major research tasks are described. Highlights of this development effort are: physical and information/decision components are modeled separately; a set theoretic formalism is provided; a model specification language is critical; a powerful graphical user interface is essential. Current status and future plans are described.

1. INTRODUCTION

1.1 Background

A simulation model is an abstract, logical, mathematical model of a system which can be dynamically exercised through the use of a computer. The main goal of simulation modeling is the development of a model that represents the real system such that experiments performed on the model yield results which can be extrapolated to the real system.

The history of simulation modeling software can be broken into five periods: the era of custom programs, the emergence of simulation programming languages, the second generation of simulation programming languages, the era of extended features, and the current period [Nance 1984]. Early simulation modeling was performed using custom programs written in general purpose computer languages, such as FORTRAN. Although this approach proved the viability of simulation modeling, the models were typically expensive and time consuming to design and maintain. Usually, the work done on a specific modeling project could not be easily utilized during subsequent modeling efforts. This resulted in simulation being used primarily on large, expensive projects.

In the early 1960's, as the field of simulation developed further, discrete event simulation languages such as GPSS, GASP, and SIMULA were introduced [Nance 1984]. These languages were primarily written in general purpose languages but provided generic functions and subroutines to perform many of the tasks routinely required in simulation, such as calendar functions (e.g., scheduling of events) and statistics collection. Unfortunately, the bulk of the simulation model development effort was still spent in developing problem specific code that had little reusability in future problems. In the late 1960's a second generation of simulation languages emerged. In most cases (i.e., GPSS V, SIMULA 67, and GASP IIA), these languages were more powerful replacements of their predecessors.

In the 1970's, as the use of simulation modeling grew, developments in simulation languages were driven toward the extension of simulation specific languages to facilitate easier and more efficient methods of model translation and representation. Many of the languages which evolved from these developments, GPSS, SLAM, and SIMAN, are still widely and actively used today [Law and Haider 1989].

In addition to the developments occurring in simulation languages, changes were also occurring in the way simulation models were used within organizations. More and more frequently simulation was used to study smaller, short term problems and projects. This effectively increased the pressure for development of faster and more efficient modeling methodologies with higher levels of reusability and user friendly interfaces.

1.2 Current Status of Simulation Modeling

In the early 1980's, many changes were occurring in the computer hardware arena; personal computers were becoming a mainstay, high resolution graphics and animation were efficiently realizable, and artificial intelligence (AI) and expert systems were seeing a resurgence with practical implementations. These changes had, and continue to have, a direct impact on simulation methodologies. Simulation modeling is now open to a much broader base of potential users through advances such as: 1) menu and icon driven model builders, 2) expert systems to aid in the building and debugging of models, 3) graphs and charts to display model results both during and after execution, and 4) model animation to view the operation of the system as a whole or zoom in on a specific area of interest.

In the area of graphics and animation, packages such as SLAMSYSTEM, Cinema/SIMAN, and SIMFACTORY [Law and Haider 1989] are among the leading edge competitors. The animation and graphics are typically developed and presented as an integral part of the simulation language. By contrast, AI and expert system concepts impact simulation modeling through the use of a simulation 'front-end' or application generator. These tools interact with the user and ultimately result in a set of code which can be passed directly to the simulation language. Among the leading edge competitors in this area are EZSIM [Endesfelder and Tempelmeier 1987], SMP [Khoshnevis and Chen 1987], and MAGEST [Ören and Aytac 1985].

In terms of continuing the growth of simulation modeling and expanding the use of simulation in general, construction of new simulation models and modification of existing models still provide formidable challenges to researchers. For example, modeling primitives and animation objects must be expressed in the user's language. Also, the time required to construct and validate simulation models must continue to decrease through the use of concepts such as rapid prototyping and model reusability. Object-oriented programming appears to have the potential to be a major contributor to these areas of research.

1.3 The Object-Oriented Paradigm

The concepts of object-oriented programming (OOP) are having a profound impact on computer software construction. Advantages of OOP over traditional (procedural) programming have been documented in Cox [1986] and Meyer [1987]. According to Meyer [1987]: "*...object-oriented design may be defined as a technique which, unlike classical (functional) design, bases the modular decomposition of a software system on the classes of objects the system manipulates, not on the functions the system performs.*" Meyer goes on to point out that objects remain more or less stable, whereas functions tend to adapt to changing needs or circumstances.

The principal idea associated with OOP is that all items (e.g., variables) in the system are treated as "objects". An object is either a "class" or an instance of a class. A class is that software module which provides a complete definition of the capabilities of members of the class. These capabilities are either provided by the procedures and data storage contained within the immediate class definition or

inherited from other class definitions to which this class is related. An instance of a class is a realization of the class having all of the operating capabilities provided in the class definition. OOP embodies four key concepts which result in making software systems more understandable, modifiable, and reusable. These concepts are: encapsulation, message passing, late binding, and inheritance.

Encapsulation means that an object's data and procedures are enclosed within a tight boundary, one which cannot be penetrated by other objects. Data stored within an object is directly accessible only by the procedures that have been defined as part of the class to which the object belongs. The use of objects therefore improves the reliability and maintainability of system code.

Message passing is a necessary result of encapsulation. In order for one object to affect the internal condition of another object, the first object must request (by sending a message) the second object to execute one of the second object's procedures.

Binding refers to the process in which a procedure and the data on which it is to operate are related. In contrast to early binding (i.e., at the time of code construction) in traditional procedural languages, late binding provided in OOP delays the binding process until the software is actually running. This feature provides for variable data types to change during execution, operator overloading (i.e., same message structure for different code implementation), and storage independent of data type.

Inheritance provides for a low level form of software reuse. OOP classes are defined in a hierarchical tree structure. Each class in the tree structure inherits the methods and data storage structure of all of its superclasses. Inheritance allows the construction of new objects from existing objects by extending, reducing, or otherwise modifying their functionality.

The differences between software development in procedural languages and OOP languages is due to these four characteristics. First, understandability of classes is improved because they represent the data and method implementations of a coherent concept rather than the loose combination of multiple procedural routines. Secondly, the four features of OOP improve the ease with which already developed software systems can be maintained and modified. By encapsulating the data and methods which use the data, internal class implementations can be altered while instances of the class retain the same message passing relationships to other objects in a software system. Finally, base language code is reusable through inheritance (i.e., definition of new subclasses) and through the use of instances of a class as an internal component of new classes.

Many of OOP's characteristics can be traced to the SIMULA 1 language [Meyer 1988]. SIMULA has found a popular academic following in Europe and throughout the world, but has never gained wide spread use in the commercial environment [Kreutzer 1986, page 105]. While SIMULA embodies some of the concepts of OOP, it is not a pure OOP language. Smalltalk, one of the purest OOP languages, was influenced by Simula's model of computation. Smalltalk added the message passing paradigm creating a programming style which we now know as OOP [Kreutzer 1986, page 194; Meyer 1988, page 437].

The concepts underlying OOP can be extended to simulation modeling [Adiga, 1989; King and Fisher 1986; Mize et al. 1989; Thomasma and Ulgen 1988; Ulgen et al. 1989]. In terms of simulation modeling requirements, following the object-oriented paradigm has the important advantage of preserving the bulk of developed code for general use in model building. Each model building exercise then performs the particular functions that are of interest at that time. The object definitions remain independent of the functions of the system being modeled. The characteristics of OOP allow us to rethink our entire approach to systems modeling using computers. The following section describes the current status of our research efforts to develop an object-oriented modeling (OOM) environment for manufacturing systems. In section 3, we conclude with a summary of ongoing development plans.

2. AN OBJECT-ORIENTED MODELING ENVIRONMENT FOR MANUFACTURING SYSTEMS

Modular representation of physical and information/decision components is a distinctive attribute of the environment described herein. A set theoretic formalism is provided to support this separation. As illustrated in Figure 1, other attributes of the modeling environment are the development of a high level model specification language, the construction of a library of simulation objects, and the provision of a graphical user interface. These concepts are discussed next.

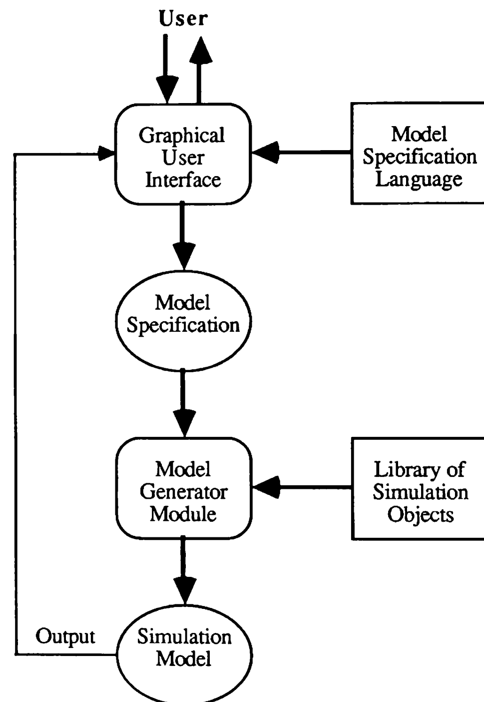

Figure 1. Modeling Environment Architecture

2.1 Separation of Physical and Information Components

The performance of a manufacturing system is highly influenced by control policies used in its operation. In evaluating the system performance, it is desirable not only to consider the physical components and the physical configuration, but also the results of different operating policies. A complex hierarchical decision making structure exists in a manufacturing system. The decisions are based on available information, which is often incomplete, inaccurate, and delayed. A decision maker at each level uses heuristics, personal expertise, company rules, and policies to arrive at a control decision. Traditional system modeling tools do not provide convenient structures for specifying these interactions. For example, in simulation modeling the representation of controlling influence is often embedded into elements of code modeling physical components.

There are two reasons why a modeler would desire to incorporate explicit and separate information processing and decision making structures into his/her models: 1) to obtain a more realistic model of the system, and 2) to determine the effect a certain operating policy will have on the system performance. In performance modeling of manufacturing systems, existing simulation languages fail in this regard. They do not provide realistic constructs for modeling information flows and control decisions. In addition, the constructs they do provide must be hard coded, and dispersed into the model, creat-

ing code that is hard to modify. Thus, evaluation of a different control policy necessitates an almost complete rewrite of the model.

A new modeling paradigm is needed to capture the dynamics of information processing and decision making as well as the physical processes in manufacturing systems. This architecture should separate the decision making entities from the physical components so that changes in one can be done independent of the other. AI and OOP offer tools to create such a modeling environment. For example, the knowledge representation structures in AI provide a means for separating knowledge and data [Nilsson 1980], while OOP offers an environment in which code reusability makes it easy to modify the models [Cox 1986].

In the proposed framework, information flow is modeled in a hierarchical structure [Karacal 1990; Zeigler 1976]. For example, a customer order gets translated into product orders, which, in turn, give rise to component orders, batch orders, and finally, device activity orders. The information processing, and information delays manifest themselves in the creation of these orders. Similarly, the decision making processes are modeled through a hierarchy of control levels: system level control, shop level control, work center level control, and finally, device level control. The control levels can be modeled by embedded expert systems.

2.2 A Formalism for Modeling Manufacturing Systems

Within the context of discrete event simulation, formalism can be defined as a set of conventions for the construction of discrete event simulation models. In broad terms, it gives a definite form to how and what can be expressed about a system to be modeled. Formalism uses set theoretic concepts for the abstraction of real systems that, in the end, generates a uniform convention of communication.

To model the aforementioned separation of physical and information components in a multiple level manufacturing system, a new formalism is proposed [Karacal 1990]. The total system under study is expressed as a structured hierarchy of subsystems, each of which is one of five levels: source system, data system, information system, knowledge system, and intelligent system (see Figure 2). A very brief outline of the formalism is given in the following.

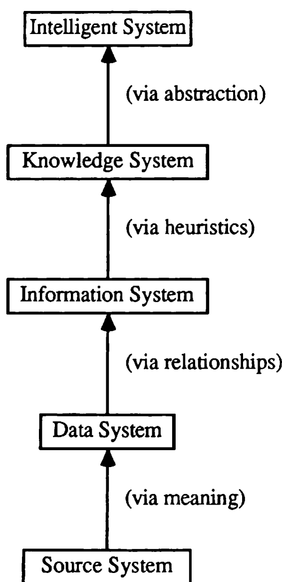

Figure 2. Five-level Model of System Intelligence

The source system, at the lowest (device) level of the hierarchy, provides facts about the system. No meaning is attached to the facts at this level. This subsystem consists of: the physical components, a set of observed facts, the real number set for time, and a mapping from observation sets to the set of observed facts. When meaning is added to the facts coming from the source system, a data system results. Thus, the data system has an additional mapping from the fact set to the data set. This mapping provides the meaning for the facts.

By investing the data with relations between them, an information system is obtained. This, again, is done through a mapping from data and operation sets to the information set. A mapping, named genKnowledge, takes the information set in an information system, and through the defined interdependencies of the information set, generates a knowledge base in predicate calculus format. The knowledge system consists of the information system, the interdependencies, the genKnowledge mapping, and the time for its components. A reasoning mechanism is then used to generate additional knowledge from the knowledge base. Intelligent system, the top level in the hierarchy of this formalism, possesses the components to carry out this reasoning and to make intelligent decisions.

To illustrate the manner in which this hierarchy is expressed in set theoretic formalism, the source system can be abstracted as follows:

$$SS = \langle P, \textbf{read}, F, T \rangle$$

where:

- **P** = representation of physical components
- **F** = set of observed facts
- **read** = mapping from an observation set to **F**
- **T** = bounded real number set for time

and

$$P = \langle X, Y, C, OP \rangle$$

where:

- **X** = set of physical inputs (work parts, tools, etc.)
- **Y** = set of physical outputs (processed parts, scrap, etc.)
- **C** = set of interface channels (communication and material flow channels)
- **OP** = operations set (physical operations)

In a similar manner, set theoretic formal expressions are constructed to represent the remaining four levels of the system intelligence hierarchy (see Figure 2).

The entities in a model of a manufacturing system can be divided into two main categories: decision making entities and data driven physical entities. The decision making entities can be represented with the properties of intelligent system level constructs of the formalism. The physical entities form the bottom level of the formalism. They may be represented at the source system level. In this way, the formalism provides constructs for separately modeling information elements, decision processes, and physical processes in the real system. The formalism described above can be constructed quite naturally in an object-oriented paradigm [Karacal 1990].

2.3 Model Specification Language

The primary purpose of the specification language is to describe the fundamental structure and behavior of the system elements. Modeling a manufacturing system now involves a "translation" of the physical, information, and decision components into this high level language. The syntax and semantics of this language are structured to facilitate the automatic translation from model specification to executable simulation program code.

The initial efforts are directed at the development of a very general form of the model specification language. The vocabulary of this language is tailor-made for manufacturing systems and hence, is easily understood by the modeler (see Figure 3). A model specification will be translated into the (Smalltalk) objects and control structures that model the essential workings of the real system. The object classes and names utilized in this translation are again ones that are readily recognized and understood by individuals involved in discrete parts manufacturing.

```
                    .
                    .
                    .
Arrivals Part-XYZ Exponential 30.0
Arrivals Part-123   Exponential 13.0
Machine MC1
Machine MC2
AssyPart Assembly1 Exponential 10.0
Routing Part-XYZ 1
    MC2 Exponential 5.0 Deterministic 0.0
Routing Part-123 2
    MC2 Exponential 2.0 Deterministic 0.0
    MC1 Exponential 4.0 Deterministic 0.0
                    .
                    .
                    .
```

Figure 3. Model Specification

2.4 Simulation Objects

The Smalltalk simulation objects in the OOM environment can be classified into two broad categories. The first category contains objects providing the software functions which allow the background simulation processing tasks, such as, time advance, event triggering, entity creation, list processing, etc., to be performed. The second category includes objects providing the reusable building blocks for modeling manufacturing systems.

The building blocks for manufacturing modeling consist of general elements found in a manufacturing environment, such as machines, material handling vehicles, conveyors, work orders, routings, etc. As the capabilities of the system are enhanced, the inheritance capabilities of OOP can be utilized to create subclasses of these generic objects that more completely model the behavior of specific items. For example, the modeling of a general conveyor will be extended to such specific conveyor types as a gravity conveyor, or a power and free conveyor, etc. Likewise, the specific types of material handling vehicles will be extended to include manually operated lift trucks, fork lifts, AGV's, etc. These enhancements are readily achievable in the Smalltalk-80 paradigm because the more specific subclasses automatically inherit the capabilities of their more general parents.

The building blocks embody a higher level of abstraction than utilized in currently available simulation languages. For example, the OOM environment utilizes a class of objects called *machines*, which are made up of three basic primitives: an input queue, a processor, and a controller. The queue and processor primitives embody the physical aspects of the machine while the controller models the decision making aspects. As can be seen, this one object embodies the characteristics of at least two SLAM II [Pritsker 1986] node statements (e.g., QUEUE nodes, SELECT nodes, etc.).

2.5 User Interface

While the model specification language is understandable to the average manufacturing systems engineer (i.e., modeler), ideally, he/she should never have to use it directly. It is desirable to provide the system modeler with an environment that allows him/her to build a model graphically, run it, and analyze it without having to directly interact with the specification language. The primary vehicle that facilitates the achievement of this goal is the user interface. The user interface utilizes the model-view-control (MVC) paradigm supported by Smalltalk-80 [Goldberg and Robson 1989]. The prototype user interface consists of two basic modules: a model development module and an execution module.

The purpose of the model development module is to provide a mechanism that allows the user to build a model of the system under study. This part of the user interface presents the modeler with a screen containing a menu for specifying commonly encountered physical system elements (i.e., machines, conveyors, material handling devices, etc.). Another menu facilitates specification of information system elements (for example, part routing, bill of materials).

After entering all of this basic system information, the user can construct and run a simulation model from the application menu. After execution the user is presented with a graphical display summarizing system performance (i.e., throughput, WIP, utilization at each machine, etc.; see figure 4). The user can then make selections regarding presentation of the text version of the output. The three options are to display it on the screen, print a hard copy, or both. Finally, after the user has viewed the output, he/she can, by selecting another menu item, return to the model development mode to make changes to the basic system data and rerun the model.

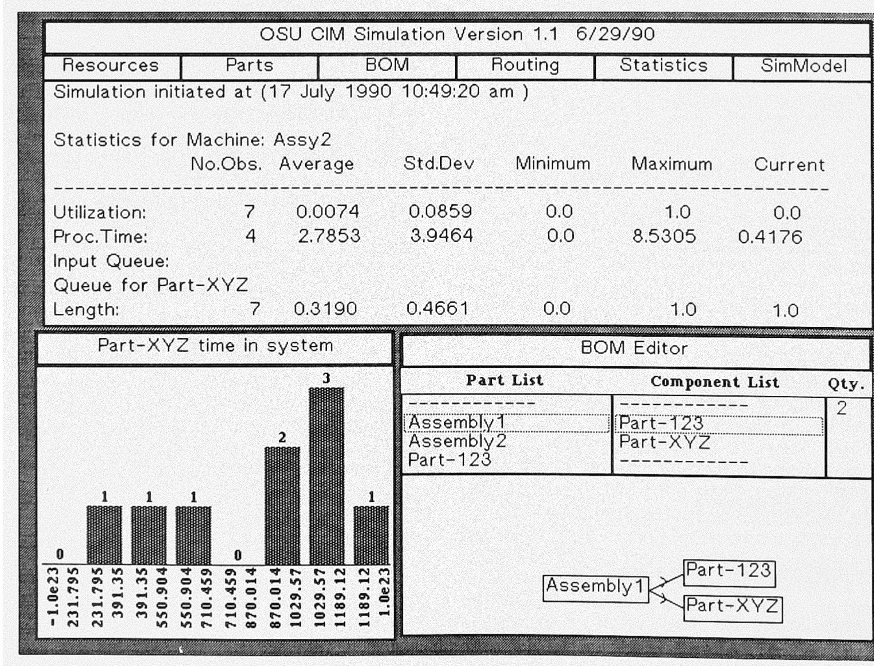

Figure 4. Output Display

2.6 Prototype Implementation

A kitting operation in a large electronics manufacturing plant served as the test bed for the first real-world implementation of an earlier version of the object-oriented modeling environment described in previous sections. Figure 5 presents a diagram of the physical layout of this system. Components enter the system as: "selects" which are directly applied to kits, bulk parts which are preformed prior to inclusion in kits, and reeled parts which are sequenced before being applied to kits. Kits which exit the kitshop are composed of the appropriate grouping of selects, preformed bulk parts, and sequenced reels. The workstations include two sequencing machines, ten kitting stations, and fifteen preform operation stations. There are work-in-process storage locations for selects, preformed bulk parts, partially completed kits, and sequenced reels. Approximately ten different kits are produced. The experience in modeling the kitting operation demonstrated the desirability of pursuing object-oriented modeling [Beaumariage, 1990].

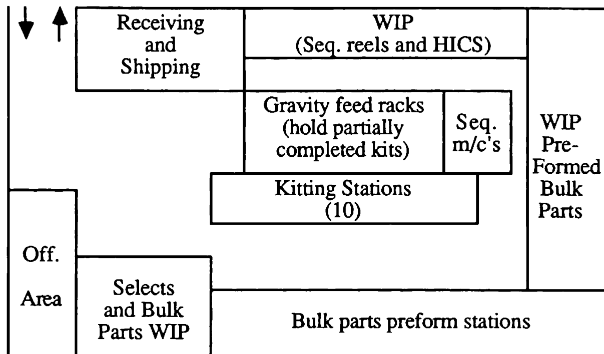

Figure 5. Kitshop in an Electronics Plant

3. ONGOING DEVELOPMENT PLANS

A number of enhancements to the environment described above are under development that will make it a powerful system design and analysis tool for the practicing engineer. These enhancement are in: 1) simulation classes and methods, 2) on-line output analysis and report generation, and 3) user interface.

Examples of enhancements which will be added to existing simulation classes and methods include:

- ➔ The ability to handle a variety of queueing disciplines.
- ➔ Expanding the current general classes of machines, material handlers, and conveyors.
- ➔ The ability for multiple executions of the same model.
- ➔ Addition of a battery of random variate generators.
- ➔ Provision for extensive statistics collection, for example, monitoring idle, busy, and setup time statistics for machines.

Examples of enhancements which allow on-line output analysis and report generation include:

- ➔ Develop the capability for storing the output data in a spreadsheet or database format.
- ➔ Allow the user to develop special report forms on-line.
- ➔ Implement a set of statistical analysis tools for on-line analysis of results, i.e., automatic confidence interval construction, etc.

Examples of enhancements to the user interface

- ➔ Develop an icon-based modeling environment.
- ➔ Provide for specifying system parameters through dialog boxes instantiated by clicking on icons.

ACKNOWLEDGEMENT

The authors acknowledge the support provided by grants from the AT&T Foundation and the Oklahoma Center for the Advancement of Science and Technology (OCAST). In addition, T. Beaumariage was also supported by a National Science Foundation fellowship.

The authors thank Prof. Joe Mize and an anonymous referee for their insightful comments which have considerably improved the presentation.

REFERENCES

Adiga, S. (1989), "Software Modelling of Manufacturing Systems: A Case for an Object-Oriented Programming Approach," In *Analysis, Modelling and Design of Modern Production Systems*, A. Kusiak and W.E. Wilhelm, Eds. J.C. Baltzer AG, Basel, Switzerland.

Beaumariage, T.G. (1990), "Investigation of an Object Oriented Modeling Environment for the Generation of Simulation Models," Ph.D. Dissertation, School of Industrial Engineering and Management, Oklahoma State University, Stillwater, OK.

Cox, B. (1986), *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley, Reading, MA.

Endesfelder, T. and H. Tempelmeier (1987), "The SIMAN Module Processor - A Flexible Software Tool for the Generation of SIMAN Simulation Models," In *Simulation in Computer Integrated Manufacturing and Artificial Intelligence Techniques*, J. Retti, Ed. SCS, San Diego, CA, 38-43.

Goldberg, A. and D. Robson (1989), *Smalltalk-80 The Language*, Addison-Wesley, Reading, MA.

Karacal, S.C. (1990), "The Development of an Integrative Structure for Discrete Event Simulation, Object Oriented Programming, and Imbedded Decision Processes," Ph.D. Dissertation, School of Industrial Engineering and Management, Oklahoma State University, Stillwater, OK (In preparation).

Khoshnevis, B. and A. Chen (1987), "An Automated Simulation Modeling System Based on AI Techniques," In *Simulation and AI*, P.A. Luker and G. Birtwistle, Eds. The Society of Computer Simulation, San Diego, CA, 87-91.

King, C.U. and E.L. Fisher (1986), "Object-Oriented Shop-Floor Design, Simulation, and Evaluation," In *Proceedings of the 1986 Fall Industrial Engineering Conference*, Institute of Industrial Engineers, Norcross, GA, 131-137.

Kreutzer, W. (1986), *System Simulation Programming Styles and Languages*, Addison-Wesley, Reading, MA.

Law, A.M. and S.W. Haider (1989), "Selecting Simulation Software for Manufacturing Applications: Practical Guidelines and Software Survey," *Industrial Engineering*, 31, 5, 33-46.

Meyer, B. (1987), "Reusability: The Case for Object-Oriented Design," *IEEE Software*, 4, 2, 50-64.

Meyer, B. (1988), *Object-Oriented Software Construction*, Prentice Hall International (UK) Ltd., Hertfordshire, Great Britain.

Mize, J.H., T.G. Beaumariage, and S.C. Karacal (1989), "Systems Modelling Using Object-Oriented Programming," In *Proceedings of the 1989 Spring Conference*, Institute of Industrial Engineers, Norcross, GA, 13-18.

Nance, R.E., 1984, "Model Development Revisited," In *Proceedings of the 1984 Winter Simulation Conference*, S. Sheppard, U.W. Pooch, and C.D. Pegden, Eds. IEEE , Piscataway, NJ, 75-80.

Nilsson, N.J. (1980), *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, CA.

Oren, T. and K. Aytac (1985), "Architecture of MAGEST: A Knowledge-based Modeling and Simulation System," In *Simulation in Research and Development*, Elsevier Science Pub., North-Holland, Amsterdam, The Netherlands, 99-109.

Pritsker, A.A.B. (1986), *Introduction to Simulation and SLAM II*, Third Edition, Halsted Press, New York, NY.

Thomasma, T. and O.M. Ulgen (1988), "Hierarchical, Modular Simulation Modeling in Icon-based Simulation Program Generators for Manufacturing," In *Proceedings of the 1988 Winter Simulation Conference*, M. Abrams, P. Haigh, and J. Comfort, Eds. IEEE , Piscataway, NJ, 254-262.

Ulgen, O.M., T. Thomasma, and Y. Mao (1989), "Object Oriented Toolkits for Simulation Program Generators," In *Proceedings of the 1989 Winter Simulation Conference*, E.A. MacNair, K.J. Musselman, and P. Heidelberger, Eds. IEEE, Piscataway, NJ, 593-600.

Zeigler, B.P. (1976), *Theory of Modeling and Simulation*, Robert E. Krieger Pub. Co., Malabar, FL.