Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

THE SEPARATION AND EXPLICIT DECLARATION OF MODEL CONTROL STRUCTURES IN SUPPORT OF OBJECT-ORIENTED SIMULATION

Michael K. Ogle

Department of Mechanical and Industrial Engineering
Louisiana Tech University
Ruston, Louisiana 71272

Terrence G. Beaumariage
Chell A. Roberts

Systems Simulation Laboratory
Industrial and Management Systems Engineering
Arizona State University
Tempe, Arizona 85287

ABSTRACT

This paper introduces concepts for the separation of model logic control structures from simulation models in support of the object-oriented simulation of manufacturing systems. A three part separation of simulation model structures is proposed to support and enhance object-oriented simulation of manufacturing systems. The three-part structural separation is made to enhance flexibility, extensibility, modularity, reusability, and understandability. The process of partitioning a simulation model into these three parts utilizes systems modeling and object-oriented modeling principles. These modeling principles enable definition of the individual modules which represent the natural partitioned structure and dynamics which should be present in an object-oriented simulation model.

1 INTRODUCTION

Traditional simulation modeling implementations package all aspects of the simulation model (entities, processes, resources, queues, decisions, etc.) within a network of nodes, or within a procedural definition of model entities and flow, or within a totally declarative description of components and flow. These past and current approaches stifle flexibility and limit the range of experimental options available to the simulation modeler.

Existing commercial simulation systems developed in non-object-oriented languages suffer from a lack of flexibility and modularity which is inherently available in an object-oriented language such as Objective-C or Smalltalk. Simulation model code developed in a non-object-oriented language is difficult to reuse because modeling constructs contained in these languages contain limited provisions for the alteration and extension of system models.

Many of the commercial systems available today face limitations due to the fact that they were originally created from five to twenty years ago. These older systems, although constantly updated and upgraded, were created in an age of limited computing resources. Compromises were made to provide working systems which would operate successfully in an age of limited resources. Although the computing resources available to programmers and model builders have increased, most languages and development environments must maintain some level of compatibility with past implementations. The implementation decisions which limited their capabilities and flexibility must influence future generations of that product. Object-oriented simulation systems which will be available in the near future will have few of the limitations currently contained within the existing commercial systems. Existing systems continue to use a "separate procedures act on separate data" orientation. Object-oriented systems use a modular "objects act on their own data" orientation. This encapsulation of structure and function enhances modularity and provides the ability to move to a more powerful generation of simulation tools.

2 PARTITIONING OBJECT-ORIENTED SYSTEMS

Object-oriented systems have been touted as a great step forward in the building of simulation models. Object-oriented systems encapsulate structure (data) and function (procedures) within one package called an object. Although object oriented systems have capabilities which provide for a great deal of flexibility, modularity, and reusability, they may still be used to create poor designs. The encapsulation process is often carried too far, placing all functionality and control for individual model elements within a single object definition. This type of packaging has the drawback of destroying modularity and adding overhead and redundancy of definition. Structuring programs in this manner is no better than structuring them in a non-object-oriented language in the same manner. Monolithic encapsulated structures do not exhibit good object-oriented programming style. Large monolithic structures exhibit a lack of modularity and flexibility for object definition and reuse.

Partitioning of object-oriented simulation structures is a necessity to allow greater understanding of individual object representation and behavior. Partitioning is also necessary to maintain consistency with structured programming techniques. Building blocks are as much a part of object-oriented simulation models as they are to the building of a physical model. Building blocks in current object-oriented simulation environments tend to be a complete physical encapsulation of code for each representation of a complete physical entity. These encapsulated packages bundle element definitions and the external control definitions used to specify dynamic behavior. Rather than an actual physical packaging of functional modules, there should be a conceptual perception of packaging the functionality. The conceptual packaging approach allows construction of individual objects from basic building blocks which may be added, removed or substituted at will, resulting in enhanced modularity and support for flexible modeling and experimentation.

3 CONTROL STRUCTURE LIMITATIONS

There are two types of control that exist in an object-oriented simulation system. The first type of control element is the control exercised by the system simulation objects. Every simulation has methods for controlling event lists, coordinating the movement of data from one file to another in support of entity and resource processing, and for manipulation of the data structures which define the basic simulation creations, operations, and terminations. The system simulation objects provide the monitoring and control which is not present in the real-world operation of the system, but are necessary to provide for computer simulation and experimentation with the system. In this paper, this control type will simply be referred to as Simulation System Control (SSC). The SSC objects should be transparent to the user. Any construct which does not have a corresponding relationship with an actual real-world entity should be transparent to the user to provide a conceptual match with the real-world system.

The second type consists of control exercised by other object structures as part of the user's conceptual model. These are decisions which emulate the decision making and physical control processes present in the modeler's view of the system. The user may determine that an entity visiting a station A will then go to station B... unless B is full... in which case it will go to C. The user may also determine that C cannot process the entity unless it has resources such as tools, a worker, a fixture, a process plan, and authorization to proceed. These are all control elements which any user must define. This type of control will be referred to as User Logic Control (ULC).

In traditional simulation systems the ULC is implicit in procedural code and the linking of system elements such as queues, stations, and material handling elements. All control of the model objects is imbedded in the model along with definition of the model elements. This arrangement stifles flexibility by creating a structure which is difficult to modify and conceptually hard to understand. There are concepts such as branching, matching, selecting, etc. which enable the modeling of control, but these elements are also placed within model code as generic procedures normally modeled as nodes. These apparent modules are essentially hardwired in the code and treated equally to other blocks in the model. An object passing through the system passes through these blocks as if they were part of the real physical system. Modeling systems using this form of representation is conceptually incorrect and forces the user to deal with elements that do not have a one to one physical analog with real world elements. Control and model elements are hardwired together into the model, restricting the user's ability to easily change and evaluate control methodologies.

It is proposed that ULC be removed from the model element definition to enhance flexibility, modularity, and to provide greater conceptual understanding of the model. This control will exist as a separate object definition external to the individual model elements which correspond to the actual physical elements found within a system undergoing simulation.

4 CONTROL STRUCTURE SEPARATION

Four general areas of research which constitute the scope of this paper are shown in Figure 1. The central issue of this paper is identified in the core of the figure; the development of an explicit representation of control structures applied to the object-oriented simulation of manufacturing systems.

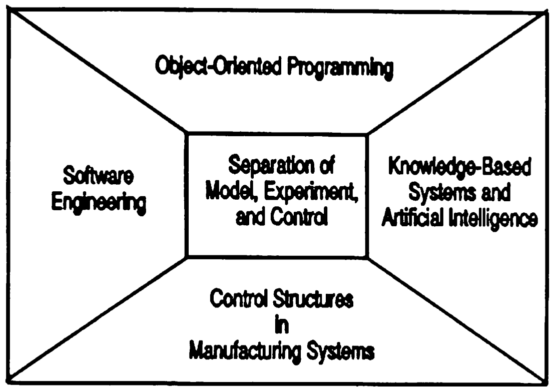
Object-Oriented Programming
Software Engineering
Separation of Model, Experiment, and Control
Knowledge-Based Systems and Artificial Intelligence
Control Structures In Manufacturing Systems

Figure 1: Four Areas Forming Scope Of Paper

Software engineering issues applied to simulation have become increasingly important in recent years. Simulation system developers and users both realize that most simulation development tools are too inflexible to handle the complexities presented by automated manufacturingsystems. A great percentage of manufacturing simulation models are only pursued as tools to evaluate the up-front design of single systems. Manufacturing simulation models must increasingly address the ongoing needs of an enterprise. Models that are constructed in the design phase should be able to service and evaluate the system once it is up and running. Ongoing simulation efforts have been hampered by the lack of flexibility and capability present in current simulation packages.

Flexible manufacturing systems (FMS) are one example where a model of the system must deal with dynamic changes throughout its operational lifetime. Haddock and O'Keefe (1990) note that the flexible nature of these systems is handled poorly by the current generation of commercial simulation systems. The authors state that when dealing with complex manufacturing systems such as an FMS...

> ...simulation models should be less disposable, being saved to answer future questions about reconfiguration. ...with many simulation tools, the incorporation of complex scheduling and routing algorithms and heuristics is difficult. Where it is possible to model scheduling, the scheduling rules are too often 'hard wired' into the simulation code, and are difficult to alter. There is a considerable need to make scheduling a separate component in the simulation.

Software engineering issues applied to discrete event simulation are presented by McKay et al. (1986). A set of guidelines and coding standards that are generic for any software system, as well as a set of guidelines that are specific for simulation models are presented. The authors point out that some work has been performed to identify the software engineering requirements of simulation languages, but that the research results had not taken shape in current simulation languages. The authors also point out that simulation code can have a long, multiuse life if the developer uses "black-box" concepts, maintains clean interfaces, and separates the problem into specific and generic characteristics.

Nance and Arthur (1988) present a set of modeling methodology objectives that should be characteristic of any complex modeling endeavor. Five primary objectives are set forth by the authors, the three which pertain to the issues presented in this paper are summarized below.

- Adaptability: changes in successive model specifications should be accomplished with relative ease so that extensibility is achieved without extensive cost.

- Reusability: model components should be extracted and made accessible for subsequent modeling tasks.

- Maintainability: model specifications should enable their modification to meet needs originally unstated.

The above "-ilities" are needs that are not addressed by the current generation of commercial tools. Models are developed for one time usage due to two factors; model developers who do not understand the relationship of a single model to subsequent models and commercial simulation package designers who design toward ease of use while neglecting the ability to reuse. Recent developments in software tools and modeling methodologies will enable both of these shortcomings to be addressed.

5 CONTROL REPRESENTATION AND OBJECT-ORIENTED SIMULATION

Fuller (1989) comments on the shortcomings of current manufacturing systems evaluation tools. Fuller believes that control systems for flexible factories are moving toward a distributed model where processing, scheduling, and routing decisions are made by a network of computers. Fuller states that...

> Simulation analogies for... intelligent task planners must be adopted. These must be available to the user in the form of environmental queries for information that reflects the way actual cell controllers and adaptive scheduling algorithms perform on the factory floor. Based on information gathered on the spot, the process or routing logic can make intelligent decisions, and find minimum-cost paths. Cost functions may be based on distance, queue depths, and machine utilization considerations. For example, a high precision mill is a scarce resource, and thus a low precision operation might be routed to an available low precision mill even if it takes longer.

The explicit representation of control in a simulation model provides a closer conceptual equivalent to the control which will be applied in the real-world system.

Burns and Morgeson (1988) also note that the current generation of simulation tools poorly represent any form of intelligent decision making and control within systems. The authors propose a separation between physical and cognitive activities. The cognitive activities would be performed by actors which have a range of responsibility and a decision set to apply to their action space. An object-oriented world view of simulation is also proposed for the modeling of the actors and their decision making capability.

Decision makers also exist on the factory floor. The decision makers on the factory floor may be workers or shop floor managers. The decision makers may also be a hierarchy of intelligent controllers starting at the workstation level, advancing through cell controllers, and finally reaching up to the point of a plant wide information system. Representation of any of these types of decision making entities is handled poorly within current process oriented, network based simulation tools.

Simulation tools which utilize an object-oriented structure have been proposed to address the control issues introduced above (Beaumariage et al. 1990) (Glassey and Adiga 1990) (Ulgen et al. 1989). These papers provide the best review of three concentrated efforts toward the development of object-oriented simulation tools which enhance reusability. Each of these efforts are beginning to develop capabilities for the separation of decision making capabilities from the physical objects represented in the system.

Other object-oriented simulation environments have been constructed at a rapidly increasing rate due to the increased interest in the capabilities of object-oriented approaches. Derrick, Balci, and Nance (1989) identify object-orientation as one of thirteen conceptual frameworks associated with simulation. Rothenberg (1986) reviews the state of object-oriented simulation and presents the features needed for enhancement of the paradigm in the future. Although it appeared only five years ago, this paper was published at nearly the infancy of object-oriented simulation.

Many researchers have associated the development of object-oriented programming with advances in artificial intelligence due to the ability of object-oriented systems to naturally represent complex relationships between data items. Reddy (1987) explains the relationship between the property descriptions offered within object-oriented structures (descriptive, structural, behavioral, and taxonomical) and knowledge based simulation. A more extensive treatment of the KBS system may be found in Reddy et al. (1986).

In an article surveying models and their relationship to artificial intelligence, Shannon (1987) provides a rather extensive analysis of object-oriented simulation model building. The paper is based on the development experiences gained from the Simulation Environment System first presented by Adelsberger et al. (1986). Shannon presents the concept "logic graphics". Logic graphics are similar to flowcharting. Symbols are interactively placed on the screen to represent the systems logic. This is the way GPSS, SLAM, SIMAN, and other graphic simulation tools perform this function of representing the system model logic. Model logic is hardwired into the logic graphics. Complex control structures are difficult to model in this fashion and testing alternative arrangements requires many unnecessary changes to the model.

Ruiz-Mier and Talavage (1987) describe the SIMYON system which is programmed in the hybrid language CAYENE. CAYENE is a hybrid language because of its mix of object-oriented programming, logic programming, and discrete event simulation. The authors report that their goal in creation of this system is to develop a representation of decision making behavior. SIMYON utilizes the network modeling method which in its graphical form Shannon above called "logic graphics." The rule based structure using logic definition capabilities within SIMYON enables the representation of additional behavior that cannot be modeled by a network of nodes. However, the network structure and other relationships between components are still buried within the object definitions.

6 MODEL-EXPERIMENT-CONTROL (MEC)

Research in object oriented simulation at Arizona State is currently focused on the construction of a partitioned conceptual model of a Model-Experiment-Control (MEC) structure. The structure separates User Logic Control from the definition of model elements and the experiments under which the elements are evaluated. The combination of the Model, Experiment, and Control constructs are referred to as MEC (Model-Experiment-Control). The MEC structure may be seen in Figure 2.

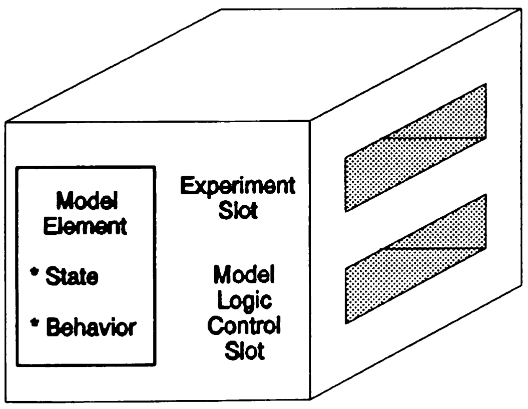

Figure 2: Model Element As An Expansion Box

The cube in Figure 2 represents a model element (such as a tool, a part produced in a factory, a machine, a process plan, etc.). The model state is represented through the local storage of data. The model behavior is represented by procedures or methods through which it is manipulated and interacts with other objects. On the right hand face of the cube is a set of expansion slots. Cox (1984) introduces a concept known as Software-IC's. The Software-IC is an analog to hardware integrated circuits which can be plugged in and associated with other circuits as part of a functional computing system. In this figure, each of the slots represent a placeholder for a software expansion board (a separate object definition) which may be plugged into a simulation model object to extend its functionality. The expansion board analogy is made because unlike the hardware-IC, which is usually placed onto a board permanently, the expansion board may be pulled and replaced with another board (object or set of objects) to easily modify the functionality or appearance of an object. This software expansion slot (placeholder for expansion boards) ties an instance of a model element in a simulation to other functional elements (expansion boards) in a simulation. A library of software expansion board objects may be developed within the object oriented simulation language. The simulation model user may then elect to use a default expansion board object or choose from a library of objects. Modifying the functionality of an individual simulation model element involves removal of the original expansion board object and replacement with another expansion board object.

The experiment slot represents the use of this object in an experiment run by the user. Separation of the model from the experiment is a valuable and flexible partitioning of functionality in a simulation. Zeigler (1976) and Oren and Zeigler (1979) first proposed this distinction in their work on system theoretic concepts. Pegden (1987) implemented this distinction in the SIMAN simulation language. As explained below by Pegden, the distinction between model and experiment is invaluable to the flexibility of model building in support of simulation experimentation.

> The system model defines the static and dynamic characteristics of the system. In comparison, the experimental frame defines the experimental conditions under which the model is run to generate specific output data. For a given model, there can be many experimental frames resulting in many sets of output data. By separating the model structure and the experimental frame into two distinct elements, different simulation experiments can be run by changing only the experimental frame. The system model remains the same.

The last slot, Model Logic Control, contains the logic to manipulate the model element as part of a larger system. In traditional simulation systems, the model logic is implicit in procedural code and the linking of system elements such as queues, stations, and material handling components. All control of the model objects is imbedded in the model along with definition of the model elements. This arrangement limits flexibility by creating a structure which is difficult to modify and conceptually hard to understand. There are concepts such as branching, matching, selecting, etc. which enable modeling of control, but these elements are also placed within model code as generic procedures normally modeled as nodes. These apparent modules are essentially hardwired in the code and treated equally to other blocks in the model. An object traversing the system network model passes through these blocks as if they were part of the real physical system. The act of modeling systems using this form of representation is conceptually incorrect and forces the user to deal with elements, namely branching constructs, that do not have a one to one physical analog with real world elements. Control and model elements are hardwired together into the model, restricting the user's ability to easily change and evaluate control methodologies.

The Model-Experiment-Control (MEC) concept provides an extension to the separation of model and experiment specifications. These three object types must be defined for any model element in the simulation.

The first object type specification applies to the declarative description of the physical objects present in the real system. This first type of object definition provides specification of the basic physical structure of an object, object behavior, and relationships with other objects. The relationship definitions do not specify the physical link or routing to another object. The relationship definitions specify a message protocol for interaction with potential clients with which the object may interact.

The second type describes the experiment under which the simulation will be run. The various parameters of the simulation concerning simulation run length, statistics gathering, assignment of distributions, etc. are handled by this type.

The third type of object specification deals with the model logic control elements. The control elements will decide when and how these structures, capabilities, and relationships will be used to support the simulation. This third type specifies the control which is exercised upon the physical objects described as the first type. The control specification defines the routings, the decision points, and all other control decisions which are applied to the model elements physically present in the system. The control structures may also have a one to one correspondence with actual controllers found on the factory floor or envisioned by the simulation developer. A library of objects will exist for each of these types. The simulation model developer may then associate individual library elements to form a fully functional model element object.

7 CONCLUSION

The state of the art in object-oriented discrete-event simulation points in the direction of a separation of model control logic from the model definition and the experiment under which it is evaluated. The separation of structures helps meet the objective of creating simulations that are flexible, modular, reusable, and understandable. The process of separating the simulation model structures is in progress within the Systems Simulation Laboratory at Arizona State University. The structure is being implemented in the object-oriented programming language Smalltalk. Smalltalk is currently running on multiple computing platforms (IBM RS-6000, Macintosh, and IBM-386 under Windows 3.0) within the Systems Simulation Laboratory and provides a highly portable implementation of developed code. The partitioned object definitions developed during this effort are constructed using a programming language independent modeling methodology known as the Object Modeling Technique (OMT) (Rumbaugh et al., 1991).

REFERENCES

Adelsberger, H.H., U.W. Pooch, R.E. Shannon, and G.N. Williams. 1986. Rule Based Object Oriented Simulation Systems. In *Intelligent Simulation Environments, Proceedings of the Conference on Intelligent Simulation Environments*, 107-112. The Society for Computer Simulation, San Diego, CA.

Beaumariage, T., J.H. Mize, and C. Karacal. 1990. Developments In Systems Modeling Offer Opportunities and Challenges to Industrial Engineers. In *Institute of Industrial Engineers 1990 International Industrial Engineering Conference Proceedings* , 28-32.

Burns, James R. and J. Darrell Morgeson. 1988. An Object-Oriented World View for Intelligent, Discrete, Next-Event Simulation. *Management Science*, Vol. 14, No. 12, 1425-1440.

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

Derrick, E.J., O. Balci, and R.E. Nance. 1989. A Comparison of Selected Conceptual Frameworks For Simulation Modeling. In *Proceedings of the 1989 Winter Simulation Conference*, 711-718. Washington, D.C.

Fuller, Charles. 1989. The Next Generation of Manufacturing Simulation. In *Proceedings of the 1989 Winter Simulation Conference*, 840-842. Washington, D.C.

Glassey, C. Roger and Sadashiv Adiga. 1990. Berkeley Library of Objects for Control and Simulation of Manufacturing (BLOCS/M). In *Applications of Object-Oriented Programming*, 1-27. Addison-Wesley.

Haddock, Jorge and Robert M. O'Keefe. 1990. Using Artificial Intelligence to Facilitate Manufacturing Systems Simulation. *Computers in Industrial Engineering*, Vol. 18 No. 3, 275-283.

McKay, Kenneth N., John A. Buzacott, John B. Moore, and Christopher J. Strang. 1986. Software Engineering Applied to Discrete Event Simulations. In *Proceedings of the 1986 Winter Simulation Conference*, 485-493. Washington, D.C.

Nance, Richard E. and James D. Arthur. 1988. The Methodology Roles in the Realization of a Model Development Environment. In *Proceedings of the 1988 Winter Simulation Conference*, 220-225. San Diego, CA.

Oren, T. I. and B. P. Zeigler. 1979. Concepts for Advanced Simulation Methodologies. *Simulation*, Vol. 32, No. 3, 1979, 69-82.

Pegden, C. Dennis. 1987. *Introduction to SIMAN*, Systems Modeling Corporation, State College, PA.

Reddy, Ramana. 1987. Epistemology of Knowledge Based Simulation. *Simulation*, Vol. 48, No. 4, 162-166.

Reddy, Ramana Y.V., Mark S. Fox, and Nizwer Husain. 1986. The Knowledge-Based Simulation System. *IEEE Software*, 26-37.

Rothenberg, Jeff. 1986. Object-Oriented Simulation: Where Do We Go From Here? In *Proceedings of the 1986 Winter Simulation Conference*, 464-469. Washington D.C.

Ruiz-Mier, Sergio and Joseph Talavage. 1987. A Hybrid Paradigm for Modeling of Complex Systems. *Simulation*, Vol. 48, No. 4, 135-141.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. 1991. *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice Hall.

Shannon, Robert E. 1987. Models and Artificial Intelligence. In *Proceedings of the 1987 Winter Simulation Conference*, 16-23. Atlanta, GA.

Ulgen, Onur M., Timothy Thomasma, and Youyi Mao. 1989. Object Oriented Toolkits for Simulation Program Generators. In *Proceedings of the 1989 Winter Simulation Conference*, 593-600. Washington D.C.

Zeigler, B.P. 1976. *Theory of Modeling and Simulation*. New York: John Wiley.

AUTHOR BIOGRAPHIES

MICHAEL K. OGLE is an Assistant Professor in the Department of Mechanical and Industrial Engineering at Louisiana Tech University. He holds a BS in Mechanical Engineering and an MS in Industrial Engineering, both from the University of Arkansas. He is currently pursuing a Ph.D. from the Arizona State University Department of Industrial and Management Systems Engineering. His interests are in information systems in manufacturing, object-oriented analysis, and simulation. He is a member of IIE, NSPE, ACM, ASEE and SCS.

DR. TERRENCE G. BEAUMARIAGE is an Assistant Professor in the Department of Industrial and Management Systems Engineering at Arizona State University. He holds a BS (Rochester Institute of Technology), MS (Oklahoma State University), and Ph.D. (Oklahoma State University) degrees, all in Industrial Engineering. His interests are in simulation, manufacturing, quality control, and artificial intelligence. He was an NSF Graduate Fellow and is a member of IIE, ASQC, ASEE, and SCS.

DR. CHELL A. ROBERTS is an Assistant Professor in the Department of Industrial and Management Systems Engineering at Arizona State University. He holds a BS (University of Utah, Math), MS (University of Utah, Industrial Engineering), and Ph.D. (Virginia Tech, Industrial Engineering) degrees. His interests are in simulation, manufacturing control, and artificial intelligence. He is a member of IIE, SCS, ASEE, and SME.