Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

OBJECT-ORIENTED SIMULATION USING MODEL BUILDER

Olugbenga O. Mejabi

Industrial and Manufacturing Engineering
Wayne State University
Detroit, Michigan 48202, U.S.A.

ABSTRACT

Model Builder™ is a modular, hierarchical, and object oriented modeling and simulation environment for complex systems. Model Builder combines visual interactive simulation with multi-abstraction modeling, extensible semantics, and direct object manipulation, for providing intuitive modeling capabilities.

1 INTRODUCTION

Model Builder takes full advantage of object orientation to provide a simulation environment that is easy to use without sacrificing any modeling power in the process. Based on the Model Builder simulation architecture, Model Builder uses a hierarchical structure coupled with a "black-box" approach to manage the complexity in large or intricate systems. Modules can be created out of any set of objects and then be stored for later use. Such modules are easily stored on the palette. The palette operates on a drag-and-drop principle. Modules from the palette can be incorporated back into models simply by dragging the desired module from the palette and then dropping it into a model.

Inheritance provides the means for adding new semantics to Model Builder. Inheritance is a central principle of object orientation and permits new data structures and semantics (or model component behavior) to be extended in a graceful and incremental fashion. In this way, Model Builder is open-ended and is suitable for use in a wide variety of modeling domains.

Ease-of-use is facilitated by direct manipulation of objects. All objects in a model can be modified directly through the user interface by using the mouse. Models can be built completely, from the ground up, without any need for programming.

Object **activities** provide a means for even more flexibility. Through the user interface, without generating any code and, therefore, without any need for compiling, the behavior of objects in Model Builder can be enhanced or modified by defining activities which they carry out at specified points in the simulation.

Model Builder exists within the innovative NeXTStep™ operating system and it therefore takes full advantage of all of the graphics and sound capabilities available on the platform. Addition of graphics and sound to models is an integral part of building models and is not relegated to a secondary step performed after the fact.

2 MODEL BUILDER ARCHITECTURE

Object-oriented, as well as, modular, hierarchical modeling architectures have become an important direction for research in the search for improved methods of building simulation models. Some research, as well as commercial systems, have been developed. These include DEVS-Scheme [Zeigler 1987], [Ulgen and Thomasma 1986], and RESQME [Gordon et al., 1986]. These systems have shown considerable promise in terms of the ability to help deal with complexity and for provision of ease-of-use.

The Model Builder architecture defines three basic types of objects for use in building models. **SimObjects** and **ModObjects** share a common origin and are used to represent real-world objects within a model. A model of a factory, for example, would represent all the cells, machines, and workers within the factory as SimObjects or ModObjects. The main difference between SimObjects and ModObjects is that

ModObjects can contain, within them, other SimObjects and ModObjects while, in that respect, SimObjects are atomic -- they cannot contain lower level model objects. ModObjects therefore serve as the basis of the hierarchical structure available in Model Builder.

Each SimObject or ModObject can have any number of **Ports** and **Attributes**. Ports capture the dynamics associated with a SimObject or modObject. A SimObject or ModObject typically has several Ports of different types. Each Port defines a particular facet of the overall behavior of the object being modeled. Ports can be placed at the input to a SimObject or ModObject -- these are called **InPorts**; they can be placed at the outlet of a SimObject or ModObject -- these are **OutPorts**; or they can be placed within a SimObject or ModObject -- these are **Actions**.

Attributes, on the other hand, represent the state information associated with a particular SimObject or ModObject. A SimObject or ModObject typically has several Attributes of different types. Each Attribute defines a particular aspect of the overall state of the object being modeled. Figure 1 below shows a hierarchical model complete with SimObjects, ModObjects, Ports, and Attributes.

When a model executes, a series of messages are passed back and forth between the Ports and Attributes in the model. Such messages are only possible between Ports and Attributes that have been connected together. The way in which Ports and Attributes in a model are connected also defines the uniqueness of that particular model. The architecture defines rules which govern which Ports and Attributes can be connected. Every connection has a source and a destination. Four types of relationships are possible between source Ports or Attributes and destination Ports or Attributes. The source and destination can either be on the same SimObject or ModObject, [SAME], or they can both be on peer SimObjects or ModObjects, [PEER], or the source can be on a SimObject or ModObject which is contained within the ModObject that the destination is on, [CONTA'D], or the source can be on a ModObject which contains the SimObject or ModObject that the destination is on, [CONTA'R]. To maintain the modularity of the architecture, certain connections are permitted and others are disallowed. Below is a summary of those connections that are allowed.

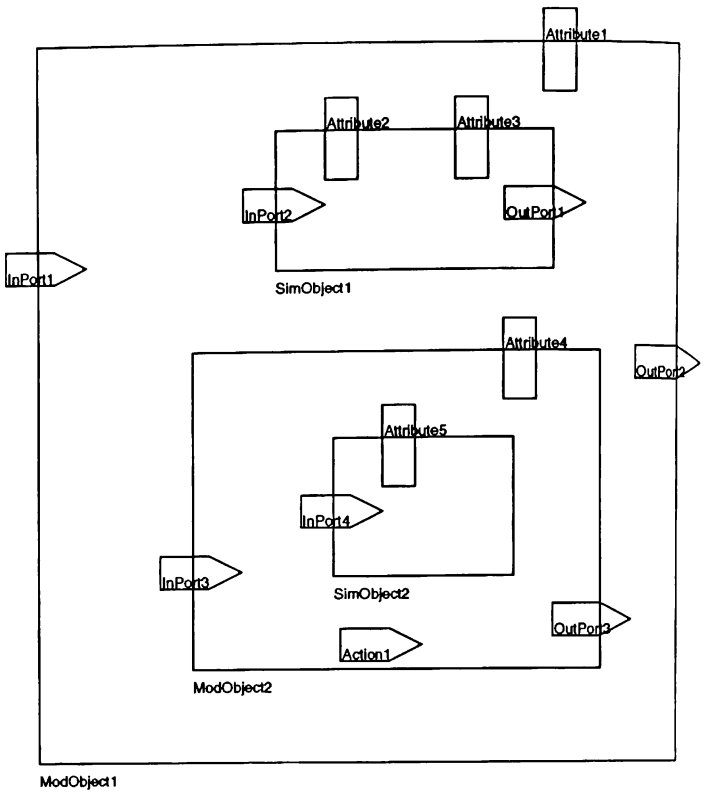

Figure 1: A Model Illustrating Components of the Model Builder Architecture

Source	Destination	Relationship
InPort	InPort	[SAME, CONTA'D]
InPort	Action	[SAME]
InPort	OutPort	[SAME]
InPort	Attribute	[SAME]
Action	InPort	[SAME, CONTA'D]
Action	Action	[SAME]
Action	OutPort	[SAME]
Action	Attribute	[SAME]
OutPort	InPort	[PEER, SAME]
OutPort	Action	[SAME]
OutPort	OutPort	[SAME, CONTA'R]
OutPort	Attribute	[SAME]
Attribute	InPort	[SAME, CONTA'D]
Attribute	Action	[SAME]
Attribute	OutPort	[SAME]
Attribute	Attribute	[SAME, CONTA'D, CONTA'R]

3 MODEL DEVELOPMENT

Building models is a straightforward process. Pre-configured modules can be dragged in from the palette and inserted into a model. This facilitates the incorporation of small or large building blocks into models without the need for much expenditure of effort. Such modules can then be edited, if needed. Editing can include altering values of state variables, changing connections between Ports and Attributes, or wholesale deletion or modification in the module itself. This is the main means by which end users build models in Model Builder.

If the palette does not contain any modules that are appropriate to the particular model being built, then, the model can be built directly from individual SimObjects, ModObjects, Ports, and Attributes. These are available to be dragged in from the Class Browser. SimObjects and ModObjects can be placed inside ModObjects already in the model. In this way a hierarchical structure can be incorporated into the model. Each Port and Attribute available on the Browser corresponds to some unique form of model dynamic or object state. After being configured with Ports and Attributes, connections can be established to facilitate messages to be sent back and forth between them. This method of building models, though easy to use, is not intended for end users. Rather, developers can utilize this to create new modules appropriate for a particular domain. These modules can then be distributed to the end users.

If needed, additional logic can be added to the model by specifying **activities** for one or more of the Ports or Attributes in a model. Activities are composed of one or more **steps**. Activities are analogous to subroutines in regular programming languages and steps are analogous to individual lines of code. Activities, however, are more akin to the output from CASE tools than to program source code in that they are neither interpreted nor compiled, rather, they are created and executed directly at run-time. The activity editor in Model Builder is an easy-to-use tool which permits the creation, testing, and editing of activities. An activity step, essentially corresponds to a message, called an **operation**, to be sent to a **target object** as the step executes. Operations can optionally accept one or more operands as arguments, and the editor can be used to create new variables, for this purpose, in an ad-hoc fashion at any time. All operations return a value which can then be assigned to any variable of the appropriate type. Examples of operations that are provided include those for generating random variates, for performing addition, multiplication, division, branching, and looping, among others. In this way the activities provide full features for incorporating new logic into Ports or Attributes.

4 MODULE STORAGE AND REUSE

Once portions of a model have been completed, modules can be stored on the palette for later use. The process to accomplish this is straightforward. Any ModObject or SimObject can be selected and designated as the module to be saved onto the palette. If a ModObject is selected, and it has other ModObjects or SimObjects contained within it, then, the whole hierarchy becomes part of the module. All that remains is to provide a name and an icon for the module. On completion, the module becomes available for use in future model building.

If the aim is to create modules for distribution to end users, then instead of placing such modules on the palette, they can also be stored into a file which can then be copied and distributed as needed.

5 ADDING NEW SEMANTICS

Model Builder is capable of being extended to an unlimited degree. By using this feature, new semantics can be easily added for different domains. By inheritance, new Ports and Attributes can be designed and created. New Ports can add new types of logic, dynamics, and behavior. New Attributes can add new notions of object state. The extensibility feature makes it possible to model arbitrary types of systems without being stuck with certain, barely appropriate, semantics, as is often the case with closed-ended simulation languages.

A new class is defined simply by specifying a superclass from among the classes already created and then adding instance variables and methods. The methods contain all the code which defines the manner in which the new class behaves. After being compiled and tested, the new class can be added into Model Builder by loading it onto the browser. Once this has been accomplished the new class can be used like any other class available in the system. It can, itself, also serve as the superclass for a new class to be created at a later time. The executable image created exists in a file which can also be copied and distributed to other users for use in building models.

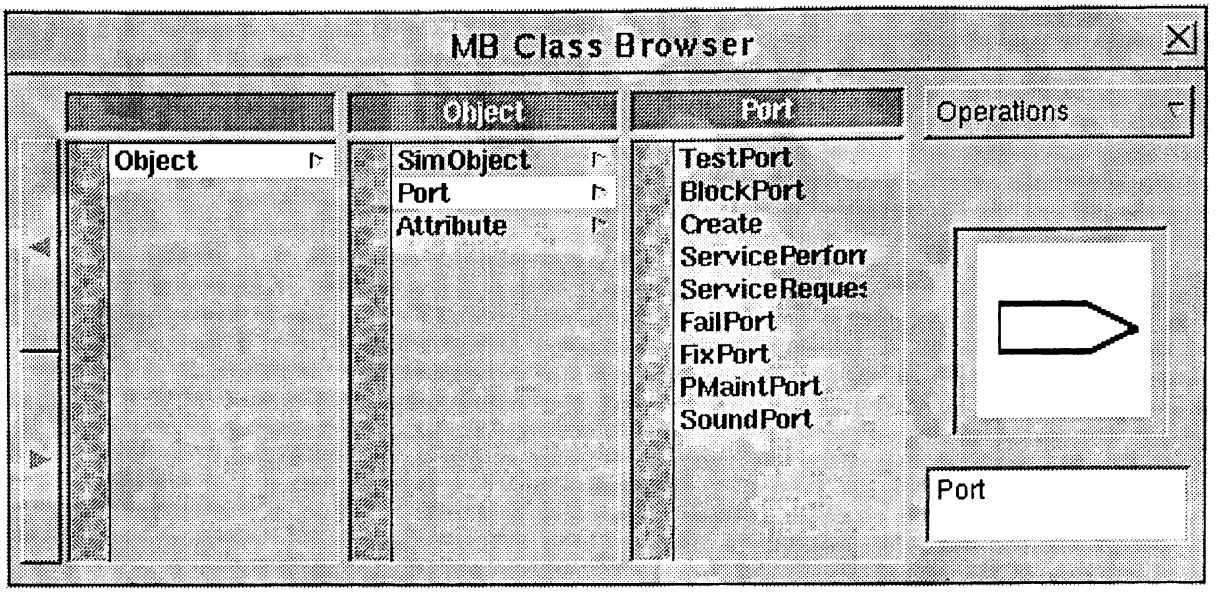

Figure 2: Model Builder's Class Browser

Figure 2 above shows the Class Browser with several subclasses of Port.

6 GRAPHICS AND SOUND

Animation, graphics, and sound, are an important part of Visual Interactive Simulation. Model Builder incorporates features for using all three. Images are objects which can be manipulated directly through the user interface or through logic built into Ports and Attributes. In this way they can be made to change color, move across the screen, or to alter other aspects of the image as the simulation executes. Sound is also stored in objects and can be played by messaging a sound object to play itself. New sounds are created simply by recording any sound and storing it in a file.

7 CONCLUSION

Model Builder introduces several new concepts into the discipline of simulation model development. It provides all the traditional benefits of Visual Interactive Simulation, and thereby promotes easy model development. In addition, by utilizing the hierarchical model structure, very complex models can be built. Using inheritance, Model Builder is open-ended and allows new semantics to be added in a completely natural and graceful manner. To take full advantage of this feature, class and module libraries designed for specific domains can be created for modeling in each domain of interest. Some possible domains include transportation systems, service systems, manufacturing systems, and data communication networks.

REFERENCES

Burns, J. R., and Morgeson, J. D., 1988. "An Object-oriented world-view for intelligent, discrete, next-event simulation." *Management Science* 34, pp. 1425-1440

Goldberg, A. and Robson, D., 1983. *SMALLTALK-80: The language and its implementation*. Addison-Wesley, Reading, Mass.

Gordon, R. F., MacNair, E. A., Welch, P. D., Gordon, K. J., and Kurose, J. F., 1986. "Examples of Using the RESearch Queueing Package Modeling

Environment (RESQME)." *Proceedings of the 1986 Winter Simulation Conference*. The Society for Computer Simulation, San Diego, Calif.

Macintosh, J. B., Hawkins, R. W., and Sheppard, C. J., 1984. "Simulation on microcomputers -- the development of a visual interactive modelling philosophy." *Proceedings of the 1984 Winter Simulation Conference*. The Society for Computer Simulation, San Diego, Calif.

Mejabi, O. O., 1991. "The Model Builder Architecture." Technical Report, Wayne State University, Detroit.

Ulgen, O. M., and Thomasma, T., 1986. "Simulation Modeling in an Object Oriented Environment using Smalltalk-80." *Proceedings of the 1986 Winter Simulation Conference*. The Society for Computer Simulation, San Diego, Calif.

Zeigler, B. P., 1984. *Multifaceted Modelling and Discrete Event Simulation*. Academic Press, London.

Zeigler, B. P., 1987. "Hierarchical, Modular, Discrete Event Modelling in an Object-oriented Environment." *Simulation* 49 No 5 pp. 219-230

AUTHOR BIOGRAPHY

OLUGBENGA O. MEJABI is an Assistant Professor in Industrial and Manufacturing Engineering at Wayne State University. He earned his Ph.D. in Industrial Engineering from Lehigh University and his M.Sc. in Manufacturing Technology from University of Manchester Institute of Science and Technology in England. His research includes development of tools and methodologies for systems simulation, analysis of general system flexibility, and development of methodologies for effective technology transfer.