Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

SIMOBJECT: FROM RAPID PROTOTYPE TO FINISHED MODEL — A BREAKTHROUGH IN GRAPHICAL MODEL BUILDING

John G. Goble

CACI Products Company
3333 North Torrey Pines Court
La Jolla, California

ABSTRACT

SIMOBJECT is a portable, model-building framework of objects embedded in a graphical editing environment. This framework facilitates the prototyping of systems as diverse as communications or transportation networks, manufacturing processes, and business operations. OBJECT.MGR, the graphical object editor, is used to extend these prototypes and evolve them into detailed applications as more is learned about the system that is being simulated.

1 BACKGROUND

In a wide variety of industries, analysts use simulation to predict the performance and study the behavior of complex systems, perhaps most successfully in communications, transportation, logistics, manufacturing, and military planning. However, barriers to its wider use are the time, cost, and technical expertise required to create and maintain a simulation model for each particular system. The availability of off-the-shelf models has partially eliminated this barrier, but there remains the need for more timely development of prototypes, new models, and customized versions of off-the-shelf models.

Our approach for overcoming the difficulties of developing simulation models is to provide the model developer with a collection of reusable software components that can be extended and configured into a model. By also providing the model developer with a graphical interface to manage the modification and assembly of these components we have greatly reduced the amount of time required to build, enhance, or tailor a model.

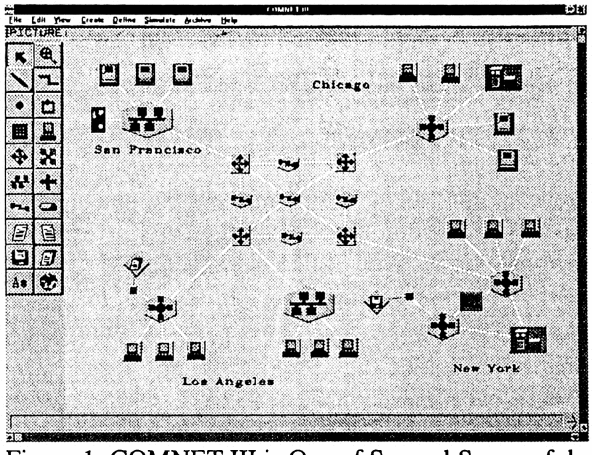

Figure 1: COMNET III is One of Several Successful Applications Built on SIMOBJECT

The need for reusable components that can be extended requires an object-oriented approach. The inheritance mechanism of an object-oriented model-building technology provides the capability to meet that need. Through inheritance from other objects, an object can be given additional or modified functionality—it can be given new data fields and new method procedures to manipulate them. But, at the same time, the inheritance mechanism constrains the changes so that they conform to the original interface specification, and thus preserves interchangeability.

Objects with the same functional interface, although differing in their internal implementations, can be substituted for each other without disturbing the rest of the model. For instance, a traffic router node in a network simulation model can be replaced with a node that has an entirely different routing strategy without having to make changes to other parts of the model.

The choice of programming language for simulation modeling is important, especially if the application is to be developed by someone who is an expert in the system being studied instead of a professional programmer. It is possible to develop a simulation model using a general

purpose language such as C++ supplemented by subroutines to support the concepts of events sequencing. However, there are major benefits to starting with a language that understands the semantics of object interaction over time and the concurrency of object behaviors. For example, using such a simulation language, the developer will be warned at compile time about the misuse of WAIT or INTERRUPT statements. On the other hand, when using C++ subroutines, the misuse warning does not appear until late in the testing phase when the code is executed at run-time.

At CACI, we have 30 years of experience in providing simulation software such as the SIMSCRIPT II.5 language and off-the-shelf simulation applications such as COMNET II.5. In 1989, we introduced MODSIM II, a high-level, object-oriented simulation language. MODSIM II allows the description of the behaviors of objects over time, and also can address concurrent object behaviors such as waiting, activating, and interrupting.

The developments described here are the outcome of efforts to combine the productivity benefits of off-the-shelf tools with the complete flexibility of a sophisticated modeling language. It seemed that this could be achieved by (1) developing a library of reusable objects, and (2) developing the means for graphically modifying and extending objects and configuring them into a model. Based on MODSIM II, we have developed SIMOBJECT and OBJECT.MGR.

SIMOBJECT can be thought of as providing a means for graphical modeling—the building and modification of simulation models by configuring objects selected from a database of basic, prebuilt simulation objects. OBJECT.MGR, on the other hand, can be regarded as a means for graphical programming—giving objects new attributes and behaviors in order to build new types of objects to add to the database.

2 GRAPHICAL MODELING WITH SIMOBJECT

SIMOBJECT is a model-building framework of objects embedded in a graphical editing environment. The collection of objects in the SIMOBJECT framework support both the building of models of systems that involve flows through networks, and the building of graphical interfaces to edit and configure these models. Put together, these capabilities support the complete prototyping and evolution of many kinds of simulation models.

SIMOBJECT's powerful graphical modeling environment, coupled with its library of off-the-shelf objects, sharply reduces the time required for prototyping a system. A working model is assembled by selecting objects from the palette and dragging them to the work area on the screen. These objects are connected together to represent logical sequences and routes. The parameters of each object are selected through its associated dialog box. At any time, the model can be checked for consistency and completeness.

SIMOBJECT's collection of prebuilt, simulation-based objects includes Nodes, which model processes, resources, and hierarchy, Tokens, which model the elements that move through the system, and Arcs, which form the node-to-node connections along which Tokens flow.

Each node object is associated with a routing object that understands the protocol by which tokens can be either pushed out of or pulled through the node. When a router is presented with a token at a node, the router can use any arbitrary logic to determine when and how to move the token. This separation of the routing object from the node object allows routers to be replaced by routers with different behaviors without perturbing the model layout. Different instances of routing objects can be used to characterize different requirements. In a communications network, for example, one instance of a router can implement a flooding protocol while another implements point-to-point messages. In a model of workflow through a production area, the router might implement a line-balancing method.

A special feature of SIMOBJECT allows the developer of a prototype model to use SIMOBJECT's powerful graphical environment to create a customized graphical interface for subsequent users of the prototype. This feature is unique to SIMOBJECT. Both the prototype model and the graphical interface for the user are built at the same time. The customized interface enables users of the model to graphically assemble the scenario for a run of the simulation without having to write any programming code.

When run, the simulation model presents an animated display of the system being studied. The user may interrupt the simulation, make changes, and immediately see the effect of the changes by continuing the simulation. Most objects have the capability, which can be selectively enabled, to automatically gather statistics from a simulation run.

SIMOBJECT provides an advanced starting point for developing models by providing a model framework. It can be used to model systems as diverse as communications, logistics, and transportation networks, manufacturing processes, and business operations. Any system that can be depicted as flows through a network can be graphically modeled using SIMOBJECT.

Another capability that SIMOBJECT offers the model user is the option to display less detail in order to simplify the screen. With a complex model, the screen

may be so cluttered with icons that it becomes difficult to follow the animation. Because SIMOBJECT's graphics are hierarchical, an entire subsystem can be shown as a single icon even though the subsystem is being simulated in detail.

In the early stages of model development, it is often convenient to define an object or subsystem in a general way, with the details of its behavior to be filled in when more is known about it. Even after the detailed model is completed, it may be desirable to select the general definition for a particular simulation run, either to reduce running time or because it is adequate for the purpose of the study. With SIMOBJECT, the desired version of an object or subsystem can be selected at run time. This capability to conveniently select at run time from different levels of detail, or to select from a menu of alternative subsystem configurations when performing trade-off studies, provides a feature long sought after by model users.

SIMOBJECT graphical modeling is general, and supports rapid model development across a broad range of different applications. This capability applies to any modeling problem that can be characterized by SIMOBJECT's node-link approach.

For example, CACI is using SIMOBJECT to build a next-generation communications model. Another development effort, going on *at the same time*, uses SIMOBJECT as the foundation for a model of business process re-engineering. Two quite different applications — same underlying SIMOBJECT software foundation.

Each of these two applications consists of several hundred object types, of which about 80% are derived from objects provided by SIMOBJECT. Building on off-the-shelf objects through object inheritance reduced development by half. This productivity gain comes about because it requires less work to customize a pre-built object for a particular application than to define and implement an object from scratch.

3 SIMDEMO

SIMDEMO is a simple graphical model built on SIMOBJECT. It demonstrates how the user of a SIMOBJECT-based model builds scenarios of tokens flowing through a network of nodes. This process involves modeling and analysis, *not programming*. Key modeling elements include atomic and hierarchical nodes, input and output controllers, arcs, tokens, resources, and models or scenarios. The following figure presents a typical SIMDEMO model scenario.

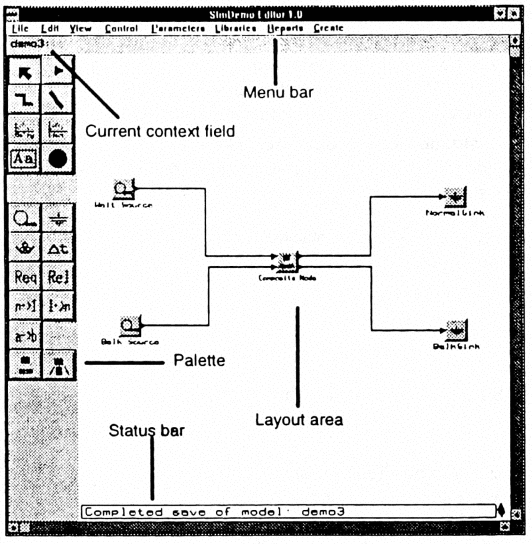

Figure 2: Typical SIMDEMO Configuration

The editor has five main areas—the menu bar, the current context field, the palette bar, the status bar, and the layout area.

The menu bar is located at the top of the editor. To open a menu, click on the menu label. Selecting a menu item causes some action to take place.

The current context field presents the full name of the current node. If you are at the highest level, this is just the name of the model. If you have entered a hierarchical node, then you'll see the name of the model, followed by a colon, followed by the name of the node. Thus, if you double click on node "HierNode," the location field will change to "DemoMod:HierNode."

The third area is the palette bar. The palette bar contains icons representing various kinds of objects that can be added to the layout. For example, nodes and arcs. To add a new node to the model, click on a palette icon representing a node, drag it to the layout area, and release the mouse button. A new instance of the appropriate kind of node will be added to the model.

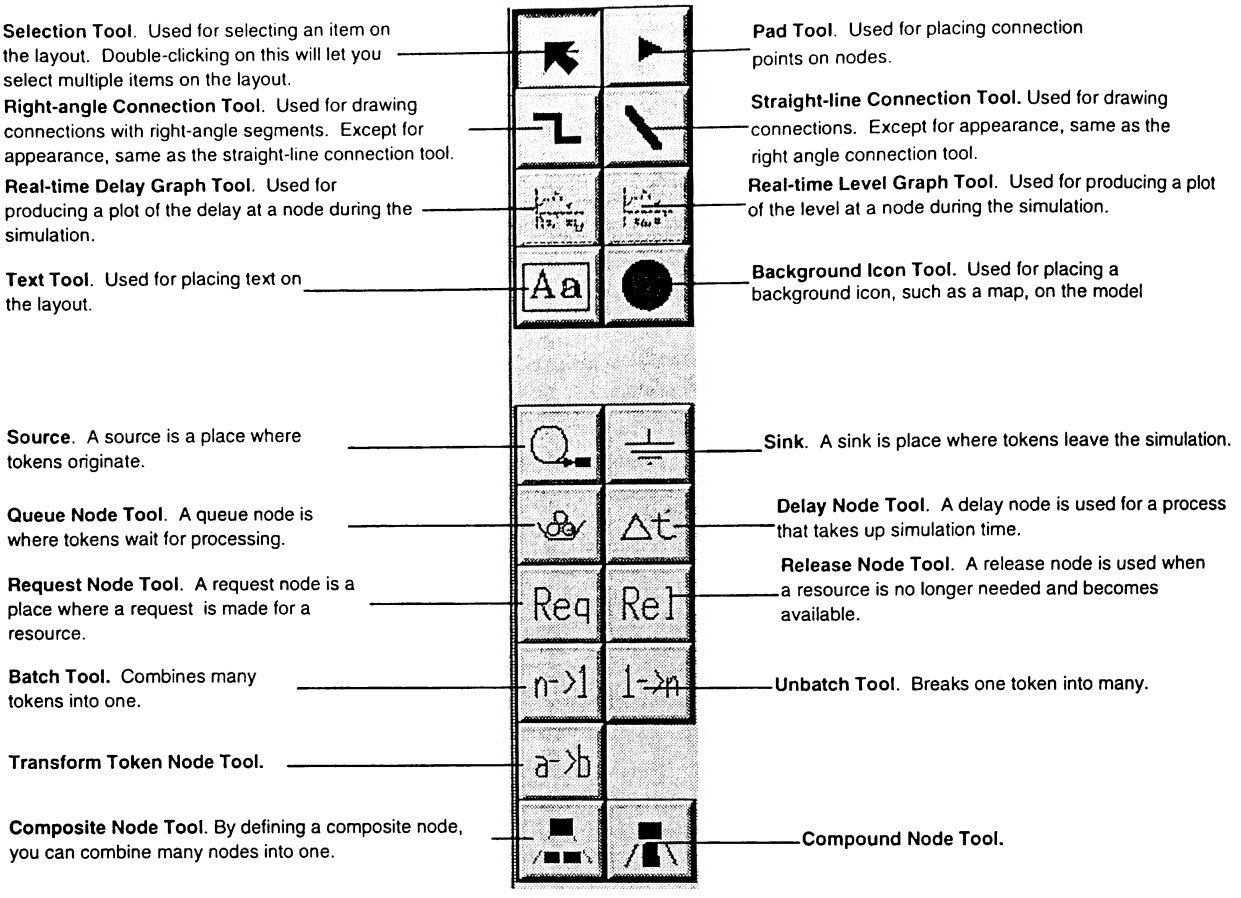

Figure 3: SIMDEMO Palette

The fourth area is the layout area. The layout contains the iconic representation of the topology of the model. It contains graphical model elements representing nodes, pads, and arcs, and visual cues such as background icons and text. The background icons and text have enhance the visual appearance of a model.

The last area is the status bar. SIMOBJECT sends error, warning, and trace messages to the status bar as the editing session or simulation progresses.

A node (NodeObj) represents a process or a hierarchically arranged group of processes. SIMDEMO implements these two types as separate classes—process nodes (ProcessNodeObj) and hierarchical nodes (HierarchicalNodeObj). The NodeObj is the common ancestor of these types.

A process node (ProcessNodeObj) is a non-decomposable model element. Different subtypes model a source, delay, transform, batch and unbatch, resource request and relinquish, and sink. You can, of course, build your own subtypes. To edit the parameters of a node, double-click on the node's icon and fill in the fields of the dialog box.

A hierarchical node (HierarchicalNodeObj) contains other nodes that may themselves be hierarchical nodes. SIMDEMO contains no built-in limits on the depth to which nodes can be nested. Double-clicking on a hierarchical node causes the editor to enter the node. That is, the layout area now contains the contents of the hierarchical node. To edit the parameters of a hierarchical node, highlight the node by clicking on it, and select **Edit/Detail** from the menu bar.

A pad (PadObj) models an input or output controller for the node to which it is attached. When a node desires to route a token, it passes the token to one of its pads and asks the pad to route the token. A pad knows the node to which it is connected, and the node keeps a list of the pads that it owns. This separation of the processing behavior (which belongs to the node) and the routing behavior (which belongs to the pad), allows you to mix and match the behaviors without being required to build an unreasonable number of node flavors just to implement alternative routing strategies.

An arc (ArcObj) connects two pads. The arc knows the pads that it connects, and the pad keeps a list of the arcs that it references. The arc does not delay, process,

or transform a token; it merely indicates a connection between the two pads.

The token represents an entity moving through the system. The token can represent packets or frames in a communications network, parts on a factory floor, customers in a bank, etc. SIMDEMO actually implements the concept of a token through two objects, the token category (TokenCategoryObj) and the token (TokenObj). The user edits the parameters of token categories. These parameters include a name, an icon, and statistics collection requests. During the simulation, SIMDEMO creates tokens that flow through the model. Each token has a reference to a token category, and represents a single instance of that type of category. For example, say the user creates a new token category, names it "PS/2 Model 80", and sets the icon and statistics requests. He then selects a source node on the layout and identifies "PS/2 Model 80" as the token for the source to create. During the simulation phase, the source node will create tokens that point back to the correct category. The individual tokens do not carry the name, icon name, and statistics request information; this information is only held in the token category pointed at by the token. Tokens exist only during the simulation phase.

SIMDEMO resources (ResourceObj) model domain elements that constrain the performance of the system. They could represent manpower, communications links, or processing elements. A token requests resources at a request node and releases them at a release node. The token waits at the request node until the desired number of units of the indicated resource are available.

A user configures a scenario by establishing a topology and parameterizing the objects involved. This is represented in SIMDEMO as an instance of a model (ModelObj). The SIMDEMO model knows all elements of the configuration. It has a list of the nodes at the top level, and owns the list of tokens and resources, among other things. The model is able to reach every element involved in the scenario.

4 GRAPHICAL PROGRAMMING WITH OBJECT.MGR

While a library of objects for communications modeling or business process re-engineering may function well enough to satisfy the needs of most users, some modelers will require a well-defined method for extending the modeling environment. Configuring off-the-shelf components by choosing parameters alone may not be enough, for example, where the characterization of a new, or experimental, situation is required.

In the past, successfully making such changes to a large, complex program was dangerous and generally avoided—especially if the original developers had moved on and could not be consulted. The conventional way to do it is by locating the underlying code, changing it, and hoping that unwanted side effects were not introduced. The inheritance mechanism of an object-oriented environment facilitates both locating and isolating the code to be changed. Changes in functionality can be made to objects without changing their interface specifications, which means that the modified object can be used without introducing unexpected disturbances to the model.

At CACI, we felt that it should also be possible to apply graphical interactions to make programming changes. The same set of interactions used to configure a network can be used to graphically configure the properties of an object. We have developed a utility for SIMOBJECT that applies graphical manipulation techniques to the description of objects themselves. We call this utility OBJECT.MGR.

OBJECT.MGR is a utility for graphically browsing, modifying, and constructing the objects that are part of a simulation model. OBJECT.MGR's graphical environment is used to make programming changes to existing objects and to create new objects.

Because OBJECT.MGR maintains complete descriptions of all objects in a database, it can provide the model developer with a variety of ways to view and manipulate these descriptions. A graphical tree representation can display an object's inheritance relationships with all its ancestors. The data structure of an object can be composed or edited by dragging and dropping data fields from the palette. The code of its method procedures can be graphically traversed as a hierarchy of statements.

One of the attractions of using graphical editing techniques is that the editing environment "knows" about the structuring of objects and so can guide the interaction, avoiding meaningless constructs. Rather than retyping identifying names, type and reference information can be associated with an item simply by clicking on icons.

OBJECT.MGR is not restricted to graphical views. Textual presentations of code may be generated at various levels of granularity for inspection and editing with conventional text editing tools. On completion of an edit, the database representation is updated to incorporate the changes. OBJECT.MGR allows editing at any desired level, can translate back and forth between graphical and textual views.

An important feature of OBJECT.MGR is its support for searching and browsing of object databases. It is in the nature of object-oriented software to reuse objects in

inheritance relationships, building complex objects from simpler ones. A requirement is to be able to search sets of objects for desired capabilities and browse through the inheritance of complex objects.

In OBJECT.MGR the user can search for an object type by name, either for definition or references; the search can be constrained to search only subsets of objects. The browsing feature allows the user to navigate hierarchically through the object collections and structures in the database by clicking on icons. In addition, the inheritance graphs may be used to jump directly to object definitions. Placemarkers to objects of interest can be accumulated during a model-building session for quick access to their definitions.

Within OBJECT.MGR, a mouse click opens the icon representing the SIMOBJECT database. Either browsing hierarchically, or using the search capabilities, the user identifies the object whose functionality is to be modified. Consider how a model developer might derive his own version of one of SIMOBJECT's objects by adding a new field and modifying some of the object's method procedures. For instance, there may be a need for a router object to retain memory of the last chosen route and to base subsequent routing on this.

A new object icon is selected from the palette and dragged into the work area. A dialog entry gives it a new name. Next the router object that is to be modified is dragged from a reference window and dropped on top of the new object. At this point the new object has all of the inheritance relationships of the original router. To add a new field, a field is dragged from the palette, named, and dropped on the new object.

Clicking on the new object opens up a list of icons for its fields and methods. To change the behavior of the object, the developer adds to or overrides the method procedures either graphically or by writing code changes. The logic can be edited graphically by dragging and dropping statements or with a conventional text editor. A menu selection brings up a text editing window and generates a textual display of the code. When the modifications are completed, a menu selection regenerates the SIMOBJECT application. Now SIMOBJECT displays the newly created router object on its palette of available modeling objects. The model developer can drag it from the palette to replace routers in a simulation model.

5 CONCLUSIONS

Graphical user interfaces have been enormously successful in making business software applications accessible to end users who do not want to be involved in programming or using arcane commands. We believe that graphical interfaces developed for SIMOBJECT and OBJECT.MGR can do the same thing for simulation modeling.

SIMOBJECT supports both the building of models and the building of graphical interfaces to edit and configure these models. SIMOBJECT's library of prebuilt objects and model-building framework provide an advanced starting point for the developer of a prototype model. While the prototype is being built, SIMOBJECT's powerful graphical environment can be used to simultaneously create a customized graphical interface for the users of that prototype. This graphical user interface is not something that has been added, but rather is an integral part of the simulation model.

When the objects in SIMOBJECT's library do not meet the needs of the model developer, OBJECT.MGR's graphical environment can be used to make changes to existing objects and to create new objects. This graphical programming capability is a major breakthrough, making it possible to navigate graphically through the database of a complex simulation model to find objects and then to modify them without affecting the rest of the model.

Systems change over time, and models of those systems also must evolve and change. Without the ability to readily update the model to reflect system changes, it may rapidly become obsolete. Many large complex models, developed using traditional technologies, are just too difficult to modify because of the fear of introducing unwanted side effects. Rather than risk the introduction of errors, it is common to "fool" the model through data choices. This last resort usually is unsatisfactory.

The object-oriented architecture underlying models developed with SIMOBJECT ensures that the impact of changes can be accurately predicted and understood. The powerful browsing and editing capabilities of OBJECT.MGR allow modifications to be made readily and with confidence—with exactly the same tools used to develop the model initially.

AUTHOR BIOGRAPHY

JOHN GOBLE has been involved in simulation for more than 12 years. His professional experience began at The Aerospace Corporation in El Segundo, California. He designed and implemented models dealing with wide-area communications networks, embedded computer systems, and launch vehicle processing. John now leads the SIMOBJECT team at CACI Products Company. He has extensive experience teaching and consulting on simulation and languages.