Proceedings of the 1999 Winter Simulation Conference
P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

MODSIM III AND CACI'S APPLICATIONS

Brian Wood
Kerim Tumay

CACI Products Company
3333 North Torrey Pines Court
La Jolla, CA 92037, U.S.A.

ABSTRACT

This tutorial introduces CACI's MODSIM III language, showing how its simulation "world view" together with its object-oriented architecture and built-in graphics contribute to successful simulation model building. The tutorial also provides an overview of CACI's domain-specific simulation tools, namely SIMPROCESS and COMNET III, that are developed using MODSIM III.

1 WHAT IS MODSIM III?

MODSIM III is an object-oriented, general-purpose simulation tool specifically designed for modeling large, complex systems. It offers you a robust environment specialized for the successful development of advanced models which are: 1) Visual, 2) Interactive, and 3) Hierarchical.

MODSIM III fully supports object-oriented programming including multiple inheritance, encapsulation, and polymorphism. It is intended for engineers, analysts, and consultants who are interested in saving time, money, and risk associated with large-scale, complex applications.

MODSIM III is used by over 5,000 users for defense modeling, network simulation, transportation modeling, supply chain simulation, business process modeling, and other applications.

Unlike general purpose languages such as C++ and Java, MODSIM III:

1) captures both concurrent and interacting behaviors of system components in simulation time - not an easy task.
2) provides built-in statistical modeling and statistics gathering functions.
3) includes invaluable development aids such as run-time checking of object accessing, array bounds and memory.
4) comes with simulation-specific graphics libraries and animation,
5) provides interfaces to the HLA (High Level Architecture), the DOD standard for distributed simulation.

MODSIM III combines CACI's experience with simulation programming over three decades with advances in software engineering to offer the most productive environment for the development of large, complex, evolutionary, custom models.

Examples of MODSIM III's simulation features, and the benefits of object-oriented architecture, are demonstrated below using code fragments from a hypothesized airport/airspace planning model. Such a model might be concerned with the representation of aircraft, flight duration, air traffic controllers, runway allocation procedures, and so on. To be of any interest, such a model must represent multiple aircraft in flight concurrently, and delays due to contending requests for resources, such as runways.

2 DEFINITION BLOCK

In support of modular program construction, objects in MODSIM III are described in two separate blocks of code. The Definition block describes the object type by declaring its variables and methods. This is the object description as it will be referred to by other objects in the simulation, and it provides the formal interface specification. An example of a Definition block for an aircraft object is shown below.

```
Aircraft = OBJECT;
  BestCruise : INTEGER;
  InFlight       : Boolean;
  ASK  METHOD SetCruise   (IN speed:INTEGER;
  TELL METHOD FlyDistance (IN
                                   distance:INTEGER);
 END OBJECT;
```

The Definition block for an aircraft object declares the variables and methods that aircraft objects use in the simulation model. The information the aircraft knows is contained in its variables. In this simple case, the aircraft is responsible for the management of two variables, which represent its state:

* BestCruise-the optimal speed to cruise at for given conditions
* InFlight-whether or not the aircraft is currently in flight.

3 IMPLEMENTATION BLOCK

The aircraft behaviors are described in its methods. These methods are named in the object description provided by the Definition block. The logic of what they do and how they affect the state variables of the object are described in the Implementation block, shown below.

```
OBJECT AircraftObj;

ASK METHOD SetCruise (IN speed:INTEGER);
   BEGIN
   BestCruise := speed;
  END METHOD;

TELL METHOD FlyDistance (
                              IN distance; INTEGER);
   BEGIN
   InFlight := TRUE
   WAIT DURATION distance/BestCruise;
   END WAIT;
   InFlight:=FALSE;
   OUTPUT ("Arrived Safely at",SimTime);
END METHOD;

END OBJECT;
```

The behaviors that objects can perform are the methods described in the Implementation block.

In this case the aircraft is capable of the behaviors described in the following two methods:

* ASK METHOD SetCruise-When the aircraft is requested to perform this behavior, it registers the new value for its optimal cruising speed instantaneously.
* TELL METHOD FlyDistance-When requested to perform this behavior, the aircraft calculates the required flight time to cover this distance at its cruising speed. This particular activity then pauses in execution until this period of time has elapsed within the simulation model, before completing the remainder of the behavior-in this case printing a notification that it has arrived safely. Unlike ASK methods, TELL methods are used to describe behaviors that elapse simulation time. While this method is paused, waiting for time to pass, other methods of other objects may be executing.

A key benefit of using MODSIM III in building complex simulations is the easy modeling of these behaviors. In a large model, many objects will have behaviors that must take account of the passage of time. Often, these behaviors will be concurrent, or overlapping in time. For example, our model will want to represent multiple "instances" of the aircraft object type. These instances can be created as needed; each can be given its own identifier and has its own state variables and can execute its methods as requested.

For a simple example of concurrent behaviors, let's look at how an aircraft dispatcher in our model might order two aircraft to fly to different destinations:

```
...
ASK JumboJet TO SetCruise(600);
TELL JumboJet TO FlyDistance(3000);

ASK Biplane TO SetCruise(100);
TELL Biplane TO FlyDistance(200)IN 1.0;
```

Using TELL methods, the flight times of both the JumboJet and Biplane aircraft can be modeled concurrently.

In this example, the aircraft object named JumboJet will elapse 5 hours flying a distance of 3000 miles at 600 mph. One hour after the JumboJet takes off (... IN 1.0), the Biplane aircraft will take off and fly 200 miles at 100 mph. It will complete its flight two hours before the JumboJet arrives at its destination. MODSIM III is responsible for sequencing the execution of the methods of both object instances, including the pauses to represent the flight times, so that the events of taking off and landing are played out in the correct order in the model. ASK methods do not elapse any simulation time.

4 TIMING AND INTERACTION

Besides executing concurrently, time elapsing behaviors may interact. To make the model more realistic, we want to consider the effect of changing the cruising speed of an aircraft while it is in flight-perhaps in response to a change in weather conditions. Such a change invalidates the original computation of flight time, and a new arrival time must be determined based on the new cruising speed and the distance remaining. Let's look at how the logic, or implementation, of the methods of our aircraft objects can be refined to incorporate this modified behavior. The method which is responsible for registering a change in cruising speed may INTERRUPT the time-elapsing method, FlyDistance, if appropriate. On recognition of this INTERRUPT, the remaining time to WAIT is reevaluated. To see the changes that we've made, compare this code with the original Implementation block for the aircraft object, presented earlier.

```
OBJECT AircraftObj;

 ASK METHOD SetCruise(IN speed:INTEGER);
  BEGIN
  BestCruise := speed;
  IF InFlight
    INTERRUPT SELF FlyDistance;
  END IF;
 END METHOD;

  TELL METHOD FlyDistance (IN
                              distance;INTEGER);
  BEGIN
  InFlight := TRUE
  WHILE distance > 0.0
    speed := BestCruise;
    start := SimTime;
    WAIT DURATION distance/BestCruise;
       distance := 0.0;
    ON INTERRUPT
      elapsed := SimTime-start;
      distance := distance-(elapsed*speed);
    END WAIT;
  END WHILE;
  InFlight:=FALSE;
  OUTPUT ("Arrived Safely at",SimTime);
End METHOD;
```

The aircraft's CruiseSpeed can now be changed while in flight-the arrival time will be recomputed each time this occurs.

Look at how the FlyDistance method describes the entire flight from take off to landing, allowing multiple speed change events in a logical activity description. Contrast this with multiple, disconnected, event subroutines in a conventional programming language which does not support the concept of time-elapsing behaviors.

MODSIM III understands the meaning of these simulation features. Thus it can diagnose inadvertent misuse early-for example, WAIT statements are not allowed in ASK methods that are always instantaneous. Not only does such checking save time in building and running a model, but it can help avoid debugging subtle logic errors in simulations with complex interactions.

These specialized features for modeling concurrent and interacting behaviors distinguish MODSIM III as a simulation model development tool. In addition, MODSIM III includes a rich collection of simulation building block objects. These library objects are designed to fulfill many common simulation modeling requirements. MODSIM III uses the power of object -oriented software architecture to allow these pre-built library objects to be readily adapted to special needs.

Because MODSIM III provides you with a rich set of features to manage the complex scheduling, interaction and synchronizing of time-elapsing behaviors, it saves significant development time.

Consider contention for resources, an issue which is at the heart of many discrete system simulations. Specific allocation policies are a basis for common behavior. Objects incur delays in competing for resources; they queue for resources on some priority basis; they may choose to abandon requests after a time-out interval, and the simulation model will want to report to some degree on measurements of resource utilization, waiting time statistics, and so on.

MODSIM III provides a prebuilt Resource object as one of many objects in its simulation support libraries. In our airport model, for example, runways are clearly a resource. We could use an instance of ResourceObj taken directly from MODSIM III's library to model runway allocation, enqueueing and dequeueing the aircraft on a first-come-first-served basis, and recording statistics.

We need to make one important change, however. To avoid the danger of wake turbulence effects, it is important that a light aircraft not use a runway immediately following a large aircraft; it should delay a short time to allow wake vortices in the air to dissipate. This is where inheritance comes in. It allows us to describe a Runway object in terms of the existing ResourceObj provided by MODSIM III. We only need to specify the differences between the new RunwayObj and ResourceObj.

Inheritance is one of the chief benefits of object oriented software construction, and the basis for providing libraries of useful objects which can be readily adapted to specialized needs.

In the example below, we have imported a resource management object from the MODSIM III library, defined an enumerated variable called AircraftCategory and show the Definition block for Runway. By declaring our Runway object to be derived from the library-supplied resource management object, it inherits all the built-in capabilities for enqueueing requests and maintaining utilization statistics. The Give method is declared as overridden, meaning that a different implementation, for just this method, will be substituted in the Implementation block (not shown). The Runway object also has an extra variable to 'remember' the last aircraft type. Our specialized implementation logic can now be designed to impose appropriate delays before giving the runway to aircraft of different categories.

```
FROM ResMod IMPORT ResourceObj;
TYPE
  AircraftCategory = (Light, Heavy);
...
Runway = OBJECT(ResourceObj);
  lastuse : AircraftCategory;
  OVERRIDE
    TELL METHOD Give(IN number : INTEGER);
END OBJECT;
...
```

The Runway object, derived from MODSIM III's resource management object has been customized to meet special modeling requirements.

Inheritance provides a disciplined way to selectively modify and extend object characteristics. As a specification mechanism, it maintains a clear distinction between those properties which continue to be available unchanged, and those enhancements designed to meet special needs-this is very valuable as software evolves through versions and upgrades.

New object types, derived through inheritance from existing objects, continue to conform to common interfaces, but incorporate additional capability. This is an excellent match to the evolutionary nature of successful simulation models; with increasing understanding of the system comes a desire to add details in areas of special focus.

The reuse of libraries of pre-built objects holds out the promise of real productivity gains in software development. Without a means to adapt such objects to special needs, this promise is rarely fulfilled. The extensibility offered by inheritance, coupled with the modular separation of interface definitions from actual implementation code are the mechanisms needed to support practical reuse of object libraries.

Object orientation offers other benefits to model development. The controlled access to object data structures through the object methods is necessary for building robust objects which can be the basis of reuse. Look back at the modified aircraft object implementation: any request to change the aircraft speed can now ensure a reevaluation of the flight time-which is faithful to the way things happen in the real world.

Taken together, support for object modeling concepts, along with concurrent time based behaviors are what make MODSIM III an effective simulation productivity tool.

5 GRAPHICS AND SIMULATION

MODSIM III comes with rich graphics libraries for graphical scenario layout (User Interface), dynamic analysis, and animation. Through inheritance, the objects in your simulation can aquire a rich set of graphical properties and behaviors. You can use this to provide an interactive, graphically managed model that speeds up analyses and produces easy-to-understand results. Adding graphics is easy. You use a graphical editor to configure the appearance of icons, menus, dialog boxes and presentation charts. Minimal code then connects these to the entities and variables in the model.

5.1 Graphical Scenario Layout

Interactive graphical editing lets you define a scenario to simulate by selecting icons from the palette, positioning them on the screen, and configuring parameters through dialog boxes. This function simplifies development of graphical applications such as SIMPROCESS and COMNET III.

5.2 Dynamic Analysis and Animation

With a scenario on the screen, you can begin the simulation and see an animated picture of the system under study. In addition, you can study plots that are drawn while the simulation is running. You can pan and zoom on areas of special interest. These results, shown dynamically, will suggest alternatives that can be tried immediately. Interacting with the model in this way increases understanding of the system under study and speeds your analysis. Often errors that may have otherwise been difficult to find will be obvious. Dynamic analysis contrasts sharply with the old iterative approach to simulation, where the following steps were repeated: prepare data, simulate, examine results, modify data, simulate, and analyze the results.

Through animation, you can dramatize the effect of alternative system configurations, spot unexpected behavior, and back up your recommendations. It's the best way to sell your ideas.

6 DEVELOPMENT ENVIRONMENT

MODSIM is a complete development environment that including a Compilation Manager and a Debugging Manager.

6.1 Compilation Manager

The MODSIM compilation manager automatically determines which modules have been edited since the last compilation and recompiles only those modules and any other modules that depend on them. No make files are required.

6.2 Debugging Manager

Selective runtime checking of object referencing, invalid parameters, array bounds, and memory use are invaluable aids to modeling large, complex systems. With debugging support enabled, a runtime error automatically drops you into debugging mode, allowing you to see where the error occurred and letting you examine variables. A traceback shows you the calling chain that led to the current method or procedure, so you can browse up and down the execution stack examining the sequence of procedure and method calls that preceded the error. The debugger supports a wide range of capabilities that are essential to interactive symbolic debugging. In addition it has special knowledge of MODSIM III's simulation constructs and can display the pending list, simulation time, and memory usage information.

7 MODSIM III APPLICATIONS

Using MODSIM III, CACI Products Company has developed two domain-specific simulators. COMNET III (Communication Network simulator) and SIMPROCESS (Business Process Simulator) allow you to rapidly build hierarchical, animated simulations without programming (in a matter of hours). These tools are ideal for rapid prototyping.

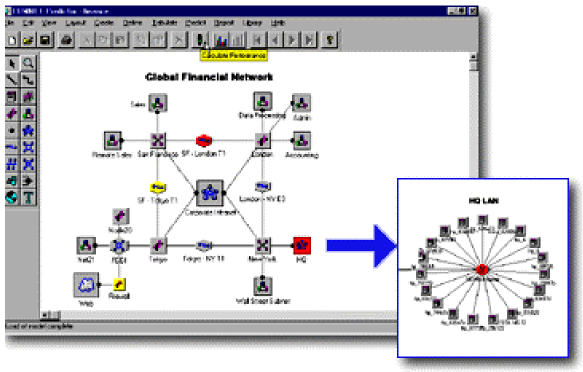

Figure 1: A COMNET III Screen shot

COMNET III is designed for network engineers and managers who need to minimize the response time and maximize bandwidth utilization of their data and telecommunication networks. COMNET III provides seamless interfaces with major network management systems and traffic collection systems. The building blocks of COMNET III, nodes (computers, routers, switches), links (Ethernet, Token Ring, Point-to-point), and traffic sources provide templates for graphically building network performance models.

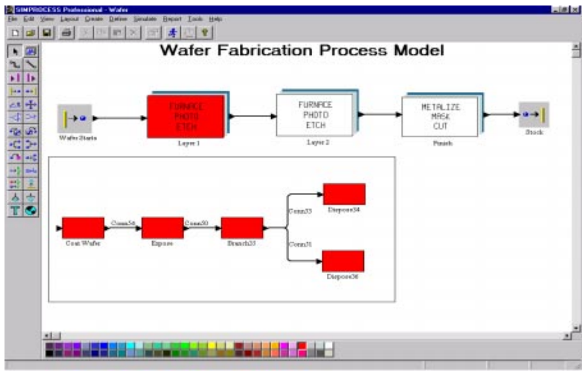

Figure 2: A SIMPROCESS Screen Shot

SIMPROCESS is designed for business process engineers and managers who need to reduce the time and risk it takes to service customers, fulfill demand, and develop new products. SIMPROCESS integrates process mapping, hierarchical event-driven simulation, and activity-based costing into a single tool. The building blocks of SIMPROCESS, namely processes, resources, and entities (flow objects), bridges the gap between ABC (Activity Based Costing) and dynamic process analysis.

These MODSIM III applications will be demonstrated during the presentation of the MODSIM III Tutorial in the '99 Winter Simulation Conference.

AUTHOR BIOGRAPHY

BRIAN WOOD started working for CACI Products Company in 1993 as a project engineer in Europe developing Air Traffic Control simulations for the EC. Since 1996, he has been a project manager for custom MODSIM III applications. During this time, he has completed projects for Teledyne Ryan, Lockheed Martin, Union Pacific Railroad, Raytheon, and the US Navy. Brian has a BS in Industrial Engineering and Operations Research from California Polytechnic State University, San Luis Obispo.

KERIM TUMAY is the Vice President of Modeling and Simulation Solutions for CACI Products Company in La Jolla, CA. He received his MS and a BS degrees in Industrial Engineering from Arizona State University. He is the co-author of two popular simulation books titled "Simulation Made Easy" and "Process Simulation Methods".