Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

MODSIM III
A Tutorial with Advances in Database Access and HLA Support

John Goble and Brian Wood

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

ABSTRACT

MODSIM II is an object-oriented discrete event simulation language featuring extensive run-time libraries, graphical user interface and results presentation tools, database access, and hooks to HLA. This tutorial introduces the MODSIM III language. It shows how MODSIM III's simulation "world view" together with its object-oriented architecture, built in graphics, and interoperability contribute to successful simulation application building.

1 WHAT IS MODSIM III?

MODSIM III is a complete development environment. The Simulation Layer, Graphics Editor, Compilation Manager, and the Interactive Debugger provide the environment required for the successful development of advanced models. MODSIM III runs on Windows 95, 98, and NT as well as Unix platforms.

MODSIM III is an object-oriented discrete event simulation language. It offers extensive run-time libraries to help the developer create commercial quality simulation applications quickly. These libraries include modules focused on the unique aspects of simulation as well as modules for developing graphical user interfaces and results presentation mechanisms. Because MODSIM III generates C++ that compiles using the platform's native compiler, it also offers seamless interoperability with other tools.

MODSIM III combines CACI's three decades of experience with advances in software engineering to offer the most productive environment for the development of large and complex models.

2 DEFINITION MODULE

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 known by other objects in the model; it provides the formal interface specification. An example of a Definition block for an aircraft object is shown below.

```
Aircraft = OBJECT
  BestCruise : REAL;
  InFlight   : Boolean;
  ASK  METHOD SetCruise (IN r : REAL);
  TELL METHOD Fly (IN dist : REAL);
END OBJECT;
```

The Definition block for an aircraft object declares the variables (fields) and methods (functions) that aircraft objects use in the simulation model. The information the aircraft knows is contained in its variables. The things it can do are described by its methods. In this simple case, the aircraft is responsible for the management of two variables, which represent its state:

`BestCruise` - the optimal cruise speed
`InFlight` - whether or not the aircraft is in flight.

3 IMPLEMENTATION MODULE

The aircraft behaviors are described in the methods provided in 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.

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

TELL METHOD Fly (IN dist : REAL);
BEGIN
 InFlight := TRUE
 WAIT DURATION dist/BestCruise;
 END WAIT;
 InFlight:=FALSE;
 OUTPUT   ("Arrived   Safely   at   ",
SimTime);
END METHOD;
```

The behaviors that objects can perform are fully 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 an aircraft receives the `SetCruise` message, it immediately registers the new value for its optimal cruising speed; simulation time does not elapse.

`TELL METHOD Fly` - When an aircraft receives the `Fly` message, it calculates the required flight time to cover this distance at its cruising speed. Next, the activity pauses in execution until the indicated period of time has elapsed within the simulation model. It then prints a notification that the plane has arrived safely. Unlike `ASK` methods, `TELL` methods describe behaviors that elapse simulation time. While this method pauses, waiting for time to pass, other methods of other objects may execute.

A key benefit of MODSIM III in building complex simulations is the ability to easily model these stochastic behaviors. In a large model, many objects will have behaviors that need to elapse time. Often, these behaviors will be concurrent, or overlapping in time. For example, our model can contain multiple aircraft created as needed; each can be given its own identifier, 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:

```
VAR
  JumboJet   : AircraftObj;
  Biplane    : AircraftObj;

BEGIN
  ...
  ASK jumboJet TO SetCruise(600.0);
  TELL jumboJet TO Fly(3000.0);

  ASK biplane TO SetCruise(100.0);
  TELL biplane TO Fly(200.0)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 sequences the execution of the methods of both object instances, including the delays representing the flight times, so that the events of taking off and landing are played out in the correct order in the model. ASKing the object does not elapse any simulation time.

4 TIMING AND INTERACTION

Besides executing concurrently, time elapsing behaviors may interact. To make the model more realistic, let's 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. This 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 to refine the implementation, of the methods of our aircraft object to incorporate this modified behavior. The `SetCruise` method can interrupt the `Fly` method 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.

```
ASK METHOD SetCruise(IN speed : REAL);
BEGIN
  BestCruise := speed;
  IF InFlight
    Interrupt(SELF,"Fly");
  END IF;
END METHOD;

TELL METHOD Fly (IN dist : REAL);
BEGIN
  InFlight := TRUE
  WHILE dist > 0.0
    speed := BestCruise;
    start := SimTime;
    WAIT DURATION dist/BestCruise
      dist := 0.0;
    ON INTERRUPT
      elapsed := SimTime-start;
      dist := dist-(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, and the arrival time will be recomputed as needed.

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

Because MODSIM III provides features to manage the complex scheduling, interaction and synchronizing of behaviors that elapse time, you get increased readability and consistency in your models. These factors translate directly to increased productivity and maintainability.

MODSIM III understands the meaning of these simulation features. Thus it detects 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 you avoid subtle logic errors in models with complex interactions.

These specialized features for modeling concurrent and interacting behaviors distinguish MODSIM III as a simulation model development tool.

5 SIMULATION LIBRARIES

MODSIM III includes a rich collection of already built and tested simulation components. These library objects meet many common simulation modeling requirements. MODSIM III's object oriented architecture lets you customize these library objects to your special needs.

Consider contention for resources an issue at the heart of many discrete system simulations. Objects incur delays in competing for resources and queue for resources on some priority basis. The objects may choose to abandon requests after a time-out interval. Otherwise, they use the resource for a while and then release it. Almost every simulation model that includes resource will want to report 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, we can model runways as a resource. We can use an instance of `ResourceObj` taken directly from MODSIM III's library to model runway allocation, servicing 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, light aircraft must not use a runway immediately following a large aircraft; they should delay a short time to allow wake vortices in the air to dissipate. Inheritance 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 import a resource object from the MODSIM III library, define an enumerated variable called `AircraftCategory` and show the Definition block for `Runway`. Since our `Runway` object is derived from the library-supplied resource object, it inherits all the built-in capabilities for enqueueing requests and maintaining utilization statistics. The `Give` method is overridden, meaning that a different implementation will be substituted in the Implementation block (not shown). The Runway object also has an extra field 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;

AircraftCategory = (Light, Heavy);
  ...
Runway = OBJECT(ResourceObj)
  lastUse : AircraftCategory;
  OVERRIDE
    TELL METHOD Give(IN n : INTEGER);
END OBJECT;
...
```

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

New object types, derived through inheritance from existing types, conform to common interfaces but incorporate additional capability. This is an excellent match to the evolutionary nature of successful simulation models; we add details in areas of special focus as our understanding of the system increases.

The reuse of libraries of pre-built objects holds out the promise of real productivity gains in software development. The extensibility offered by inheritance, coupled with the modular separation of interface definitions from actual implementation code 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 allows us to build robust objects which can be the basis of reuse. Look back at the modified aircraft object implementation: any request to change the aircraft speed now ensures 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, make MODSIM III an effective simulation productivity tool.

6 GRAPHICS AND SIMULATION

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. Adding graphics can enhance the appeal of a model in three principal areas.

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.

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 analysis where the following steps were repeated: prepare data, simulate, examine results, modify data, simulate.

Finally, 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.

7 DATABASE ACCESS

Harry Markowitz is a Nobel Prize winner in Economics, one of the founders of CACI Products Company, and the co-inventor of the SIMSCRIPT II.5 programming language. Late last year, Harry noted "The world is a database." We listened, and MODSIM III now provides tools that allow you to access ODBC-compliant databases through your MODSIM code.

In the same way that you can use the `ResourceObj` from MODSIM III's `ResMod` module, you can use a number of database access objects from the new `DatabaseMod` module. For example, to connect to a database, you use the following protocol.

```
VAR
  db            : DatabaseObj;
BEGIN
  NEW(db);
  ASK db TO Connect(foo, "", "");
```

To determine the status of the connection , use the `GetStatus`() method of the database object to get a `StatusObj` containing detailed information about the current status of the database. Once you have verified that you have a good connection, you create SQL statements using the database object's `CreateNewStatement` method, format an SQL string, and pass the statement the `Execute` message to actually fire it. Given a handle to the database (`db`), a table name (`table`), a string with comma-separated field names (`fields`), and a string with comma-separated values (`values`) you might use a code fragment like this.

```
statement := db.CreateNewStatement();
OutputError(db.GetStatus());
sqlStr := "Insert Into " + table +
          " (" + fields + ")  " +
          "VALUES (" + values + ");";
ASK statement TO Execute(sqlStr);
OutputError(statement.GetStatus());
result := statement.GetResult();
OutputResults(result);
DISPOSE (statement);
```

If we make the following assignments and call a procedure containing the code fragment show above, we will insert a record into the table named Ages with the value of Kevin for the Name field and 39 for the Age field.

```
table := "Ages";
fields := "Name,Age";
values := "Kevin,39";
```

The DatabaseMod module provides tools to retrieve data from a database as well as commit data to the database. With its roots in ODBC and SQL, and its object-oriented design and implementation, you are able to make your MODSIM III application a player in the database-centric world.

8 HLA

HLA (The High Level Architecture) is a standard for interoperability among simulations within the Department of Defense. HLA is related to earlier DoD standards such as DIS. The DoD vision for modeling and simulation to be able to construct simulation or training exercises from libraries of component simulations. HLA has been proposed as an IEEE standard and its standardization process is proceeding. More information on HLA can be obtained at http://hla.dmso.mil/.

HLA is supported by a Run-Time Interface (RTI) that is callable from many popular languages, including C++, Ada, and Java. The C++ bindings for HLA are usable from MODSIM applications and two of the first HLA federations included JSIMS and NASM/MP, models written in MODSIM. The RTI supports both local intranets and the global internet.

CACI's vision for MODSIM III is to harness the power of networks to make it easy for you to build simulation models that operate through a network and interoperate with useful components written in any language.

In order to provide the best possible support for HLA, CACI is developing MODSIM object library support for the HLA, integrated with other parts of the MODSIM system such as SimGraphics.

MODSIM's new HLA support packages the raw HLA API into MODSIM objects. These objects address all HLA areas, including Federation Management Services,

Declarations, Object Management and Ownership, Time Management, and Data Distribution. MODSIM's HLA support provides automatic handling of message pumping, exceptions, and callbacks.

MODSIM III with HLA support also provides universal data value representation so that data can be seamlessly transported "through the wire" between different computers and operating environments.

Since MODSIM III with SimGraphics is available on Windows 95/98/NT as well as on all popular Unix Systems, it is easy to load-balance a simulation model through a network or to display the user interface and animated graphical output on different or remote computers.

9 MAJOR APPLICATIONS AND USERS

Data Communications - CACI Products Company.
COMNET III, a graphical, off-the-shelf package, lets you quickly and easily analyze and predict the performance of networks ranging from simple LANs to complex enterprise-wide systems. COMNET III supports a building-block approach where nodes representing servers, computer, routers, and switches and links representing ethernet, token ring, FDDI, and satellite can be configured to model your own network. COMNET III interfaces with most network Management Systems and Network Monitoring Systems. Extensive libraries of real-world network devices are supplied for rapid and accurate modeling of networks. Hierarchical modeling objects such WAN clouds simplify modeling of X.25, Frame Relay, and ATM networks.

Business Process Modeling - CACI Products Company.
SIMPROCESS is a process simulation tool based on CACI's object-oriented MODSIM language. SIMPROCESS is a hierarchical and integrated process simulation tool that radically improves your productivity for process modeling and analysis. SIMPROCESS integrates process mapping, hierarchical event-driven simulation, and activity-based costing into a single tool. The object-oriented, hierarchical approach provides development of Reusable Templates. The building blocks of SIMPROCESS, namely processes, resources, and entities (flow objects) and its graphical modeling features minimizes the time it takes to rapidly prototype large MODSIM III applications.

Supply Chain Management - IBM Research
BPMAT is a supply chain simulation modeling tool that allows detailed modeling of various supply chain management processes. A supply chain is defined in terms of seven major process objects, namely, Customer, Manufacturing, Distribution, Transportation, Inventory Planning, Forecasting, and Supply Planning.

Maintenance - HQ AFOTEC
The Rapid Availability Prototyping for Testing Operational Readiness (RAPTOR) tool is a PC/Windows-based modeling framework, built in MODSIM III, which allows for quick creation of RM&A models for almost any system. Using this tool, time for a completed RM&A model of nearly any system can be reduced from months to minutes. Users model their systems graphically by drawing Reliability Block Diagrams (RBDs) and answering questions about the way components fail and are repaired. The component failure and repair rates can then be simulated over time to determine RM&A characteristics of the overall system.

Transportation - Union Pacific Railroad
The transportation network simulation model is a strategic planning tool for determining if there are adequate resources to achieve the next year's projected business and the train schedules to transport that business. It is a discrete event simulation developed in MODSIM III with preliminary requirements and processes defined using SIMPROCESS. The model is driven by train schedules, crew and locomotive requirements that are downloaded from a database. The traces from the simulation are loaded to a Microsoft Access based output database for decision making.

Aircraft/Air Traffic Management - Logistics Management Institute
The Aircraft/Air Traffic Management Functional Analysis Model (FAM) is a discrete event model designed to analyze alternate concepts of air traffic management and control. FAM is a completely flexible modeling environment where the user can set up an airspace by defining a number of aircraft, airport controllers, TRACON controllers, airline operations centers and sector controllers. The main question to be answered by FAM is the FAA study of alternate, more efficient ways of improving air traffic management. By manipulating the model input files, a user can essentially construct an airspace by defining number of airborne objects (i.e., aircraft) and land objects (e.g., airports, sectors, tracons, etc.) in the model.

Airdrop Risk Assessment Model (ARAM)- Air Force Institute of Technology
The Airdrop Risk Assessment Model (ARAM) is a MODSIM III-based object-oriented simulation designed to provide an advanced risk assessment tool for predicting paratroop/vortex encounters. The model essentially converts a Wright Laboratory vortex model and the Purvis-Doherr paratroop trajectory model into vortex and paratroop "objects", respectively. The model can simulate a range of aircraft formations and wind conditions, and incorporates random perturbations due to aircraft motion, wind shear, individual body weight, and parachute glide. It encompasses two coordinate systems -- one air and one

ground -- and can give ground dispersal information on all paratroop objects.

Additional applications of MODSIM III can be found on http://www.caciasl.com/modsim_major_apps.html.

10 MODSIM III AVAILABILITY

MODSIM III is developed and supported by CACI Products Company. MODSIM III is available to your organization for a free trial in your environment, on your computer. We provide everything you need for a complete evaluation at your site including training, software, documentation, sample models, and immediate support when you need it. In addition, CACI regularly offers time-tested training courses. More detailed information about MODSIM III can be found on http://www.caciasl.com/modsim.html.

AUTHOR BIOGRAPHIES

JOHN GOBLE is Vice President and Chief Engineer in the Professional Services Group at CACI Products Company in La Jolla, CA. He holds a MSc degree in Industrial Engineering from the University of Nebraska. Prior to coming to CACI he worked with Motorola as a simulation developer in the Cellular Infrastructure area. John also worked in the Modeling and Analysis group at The Aerospace Corporation in El Segundo, CA.

BRIAN WOOD is a Project Engineer in the Professional Services Group at CACI Products Company in La Jolla, CA. He holds a BS degree in Industrial Engineering from the California Polytechnic State University at San Luis Obispo. Prior to coming to the Products Company, he worked as a consultant in CACI's projects division on air traffic control simulation models for the European Community.