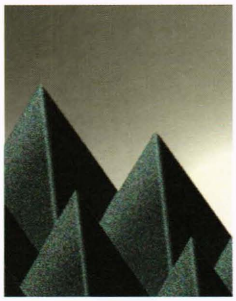


The Submarine Combat Information Laboratory and the Object-oriented Rule-Based Interactive System

Michael D. Dykton and Robert D. Sanders

Recent advances in a variety of technologies are fueling a revolution in the areas of simulation, rapid prototyping, and system integration. The APL Submarine Technology Department is concentrating these technologies into a single facility to focus them on a wide range of problems. This facility, the Submarine Combat Information Laboratory, is a flexible, land-based testbed and simulation facility developed to support Navy and Department of Defense activities. A key component of this facility is its powerful simulation development environment and stimulation software, called the Object-oriented Rule-Based Interactive System (ORBIS).

INTRODUCTION

The ship's commanding officer scans the fire control system displays and orders firing point procedures. Flawlessly, the well-trained crew execute their tasks. The skipper then gives the commands that launch a torpedo toward its target.

In the past, this scenario was likely to be a drill aboard a submarine at sea. Today, it could also be the simulated action taking place in the APL's Submarine Combat Information Laboratory (SCIL).

Significant advances have recently been made in the areas of high-performance computing and networking, distributed simulation, expert systems, and computer-visualization technology. These advances offer an opportunity to revise the paradigms that government and industry use in system requirement definition, acquisition, test and evaluation, doctrine development, and training. The benefits of a revised approach include reduced costs, more effective products, and faster development cycles.

The APL Submarine Technology Department (STD) is incorporating the recent technological advances in a single departmental facility, the SCIL. The foundation of the SCIL is a high-fidelity synthetic environment generated by the Object-oriented Rule-Based Interactive System (ORBIS). This multifaceted expert-system simulation tool supplies the development framework that enables swift prototyping of complex scenarios, assessment of emerging technologies, and development of control strategies. ORBIS enables these tasks to be performed by providing high-powered features such as sophisticated reactive rule control structures, detailed object representations, and dynamic editing mechanisms. Through the richness of these features, ORBIS can function in several arenas, including

Monte Carlo, interactive, and real-time man-in-the-loop simulations.

This article describes the conceptual development and the contents of the SCIL, including its physical facilities, supporting software infrastructure, uses, and users. Particular attention is given to ORBIS because it is the underlying simulation engine that stimulates the SCIL and creates the virtual world in which the SCIL operates.

THE SCIL CONCEPT

The SCIL is the brainchild of James R. Austin, former supervisor of APL's STD (personal communication, J. R. Austin, 28 Feb 1994). The SCIL concept emerged in the mid-1980s, when the STD was engaged in more than a dozen projects aimed at addressing new requirements or correcting perceived deficiencies on U.S. submarines by adding prototype sensors, processors, and decision aids. The STD faced three general problems in developing and fielding these systems: (1) physical space limitations, (2) information flow bottlenecks, and (3) expensive and time-consuming prototype sea-testing. A fact apparent to even a casual visitor to a submarine is that little space is available for new systems. An even more difficult issue is handling the information generated by new systems. Modern control and sensor systems manipulate massive amounts of complex data that can quickly overwhelm operators unless the data are carefully filtered and prioritized. The solution to these dilemmas is proper system integration.

The third problem listed in the preceding paragraph stems from the limited availability of resources for developing and testing new tactics and systems. Even under the best circumstances, the resources required for testing at sea are significant. In an era of declining assets and naval force levels, new methods had to be developed in response to resource limitations. One solution to this problem was to rely more on land-based system development and testing provided, of course, that it could be a realistic and effective substitute for time at sea.

Recognizing the advances made in key enabling technologies, Austin conceived a flexible, land-based development and test facility that could be used to integrate tactics, operational procedures, and software and hardware systems. The facility that emerged was the SCIL and its associated work spaces, collectively known as the Combat Information Test and Evaluation Facility (CITEF).

PHYSICAL FACILITIES

The CITEF consists of an interconnected collection of laboratories, equipment rooms, computer rooms, a briefing area, and other work spaces (Fig. 1). At the heart of the facility is the SCIL, which is a physical mockup of a submarine control room (Fig. 2). The SCIL is sized and laid out to duplicate the control room of the largest type of U.S. submarine, a Trident-class strategic ballistic missile submarine (SSBN). Appropriately scaled and located control console mockups, an arched panel ceiling, and lighting options create the illusion of being aboard ship. To add to the realism, some components were salvaged from decommissioned U.S. submarines. These components include a periscope stand and tubes, steering and diving station controls, and even flashlights and their mounting brackets.

Although we tried to give operators the sense of being aboard a submarine, our goal was not to create an exact replica of a control room. In keeping with its role as a research and development facility, the SCIL is equipped with a network of modern commercial off-the-shelf workstations instead of the Mil-Spec computers currently used in U.S. submarines. To increase the SCIL's flexibility, provision was made for unbolting sections of the flexible ceiling so that it could be reconfigured to duplicate the curvature of a submarine with a smaller-diameter hull, such as a nuclear attack submarine. Likewise, a raised computer room floor was built so that modular cabinetry and commercial off-the-shelf equipment could be reconfigured to address future needs.

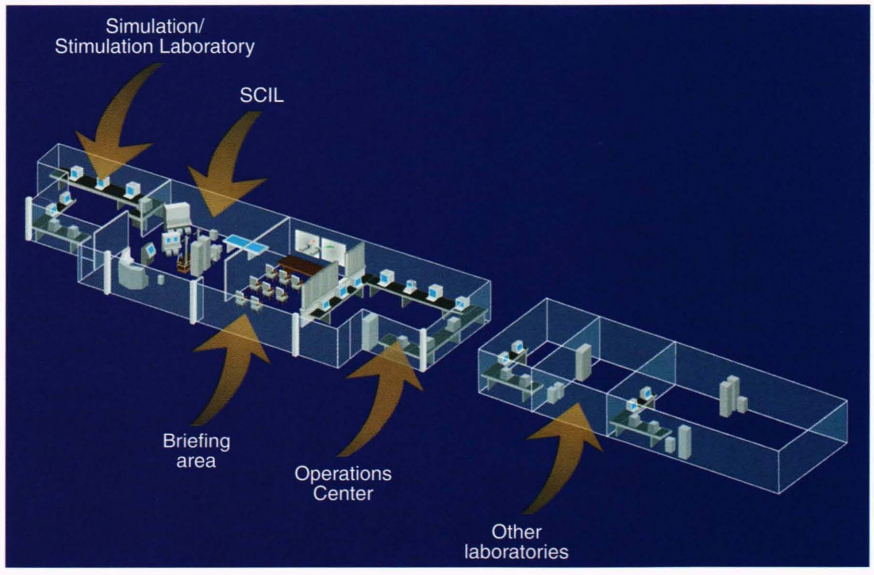


Figure 1. APL's Combat Information Test and Evaluation Facility (CITEF). This facility creates interactive virtual worlds for testing new submarine technologies and tactics.

Next to the SCIL are an equipment space and the Simulation/Stimulation Laboratory. These spaces house the Defense Simulation Internet (DSI) networking and encryption electronics. They also house workstations used to develop simulation applications, stimulate the SCIL's consoles, and interface to the DSI.

Also adjacent to the SCIL are a briefing area and the Operations Center. The briefing area allows an audience to monitor in real time the activities of operators in the SCIL and other CITEF work spaces. Such monitoring is possible, in part, because all of the CITEF spaces are linked via Ethernet, color video, dedicated three-channel voice, and RS-232 serial connections. The briefing area is equipped with a pair of color projectors, viewgraph projectors, and a television monitor. By means of a switching panel, the audience can view other laboratories selectively through portable cameras as well as see actual operator or referee "ground truth" console displays. The audience can thus monitor all activities without intruding on an exercise's participants. In addition to this internal monitoring capability, the briefing area is furnished with DSI video teleconferencing (VTC) equipment, which permits monitoring and interaction with remote, VTC-equipped DSI sites.

Figure 2. The Submarine Combat Information Laboratory (SCIL). This facility, a mockup of a submarine control room, creates a sense of being aboard ship. Control console displays enable operators to interact in the SCIL's virtual world.

The Operations Center was originally developed as part of the Critical Sea Test program to address a separate problem. Scientific investigations aboard research vessels can be hampered by limited space for equipment and staff. To overcome this problem, the Operations Center was constructed to provide real-time data links between scientists at APL and research vessels at sea, thus creating a larger pool of researchers and equipment that could contribute to an experiment. Besides the work spaces in the Operations Center, other laboratories and work spaces are available and can be allocated to projects and equipment as the need arises.

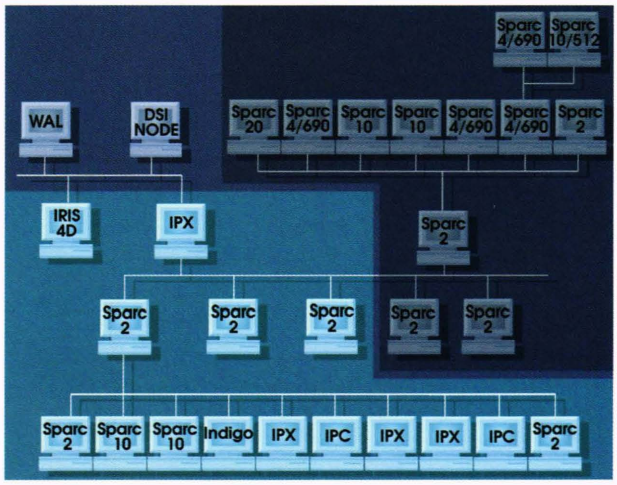


Figure 3. Diagram of the SCIL network. Open architecture permits advanced combat systems to be simulated and provides a flexible integration environment. (WAL = Warfare Analysis Laboratory, DSI = Defense Simulation Internet. Sparc, IRIS, IPX, Indigo, and IPC are names of workstations.)

While not a stated goal when the SCIL concept was conceived, the distributed computer network architecture used in the SCIL has provided a substantial benefit to the STD in the form of a powerful departmental computer facility. The open architecture that permits the SCIL to mirror advanced combat systems and provide a flexible research and development integration environment has much in common with dedicated, advanced, distributed computing facilities. The virtues of such an approach are many, including scalability, redundancy, reliability, graceful performance degradation, incremental equipment replacement and upgrades, and cost-effectiveness. At present, the STD facility contains both Sun and Silicon Graphics workstations, with over 30 central processing units typically on-line (Fig. 3). Because of the nature of the facility, however, the number of workstations fluctuates as subnetworks are dedicated to special tasks and workstations are installed or removed for field tests or off-site uses.

The STD is not unique in reaping the benefits of distributed computing. For example, the IBM Watson Research Center employs a "farm" of more than 50

workstations to supply part of its processing capacity. As reported at a recent conference on high-performance computing and networking, the new implementation has reduced the number of IBM mainframes supporting the Center by 50%.

SOFTWARE INFRASTRUCTURE

The software that supports the SCIL contains four key elements: (1) the SCIL network server (SCILNET), (2) control console display software, (3) the SCILNET/Distributed Interactive Simulation (DIS) protocol translator, and (4) ORBIS. The SCILNET coordinates communication among the distributed simulation components and acts as a central repository for the state of a simulation. The network uses a client-server architecture in which each simulation component—the control consoles, the SCILNET/DIS translator, and ORBIS—is a client program that communicates only with the central server program. The server is solely responsible for distributing messages that need to be broadcast or multicast to client programs.

The control console display system is the man–machine interface that permits operators to interact in the SCIL's virtual world. Its design allows it to be tailored to address a range of problems. Displays vary by vessel type and configuration and include such functions as maneuvering control, navigation, fire control, weapons control, countermeasure control, engineering and ship status, and sensor (e.g., sonar) displays. As designed, an entire vessel can be controlled by one person on one workstation or it can be controlled by many operators each using one workstation to display one or more ship functions. Displays can be dynamically allocated during a scenario to address changing situations or operator preferences. The control consoles do not have to be used in conjunction with the physical control room mockup; additional platforms can be created and controlled from anywhere on the facility's network. For example, an opponent (such as a KILO-class diesel submarine) can be crewed in the Simulation/Stimulation Laboratory and operated in the same simulated battle space as the SCIL submarine.

In 1989, elements of the defense community held a series of workshops to start creating standards in support of Advanced Distributed Simulations, an effort aimed at building large, interactive virtual worlds. This standards support movement is now known as Distributed Interactive Simulation (DIS). The proposed objective of DIS ". . . is to define an infrastructure for linking simulation of various types at multiple locations to create realistic, complex, virtual 'worlds' for the simulation of highly interactive activities."[1] This infrastructure provides interface standards, communication architectures, management structures, measures of fidelity, and other elements needed to bond heterogeneous simulations into unified virtual worlds. These virtual worlds, also known as synthetic environments, are intended to support

1. Design and prototyping
2. Education and training
3. Test and evaluation
4. Emergency preparedness and contingency response
5. Readiness and actual combat situations

The goals of the original SCIL concept show considerable overlap with those of DIS. It was only natural, then, that the SCIL be adapted to function as part of the defense community's large-scale synthetic environment effort to permit it to address a wider range of issues (see Fig. 4). This adaption was completed in 1993 and included installation of DSI networking hardware and development of a SCILNET/DIS translator and interface. The current SCILNET/DIS translator and interface uses a commercial product called VR-Link, developed by Mak Technologies, which adheres to DIS protocol version 2.03.

ORBIS

If the SCIL is the heart of the CITEF, ORBIS is its soul, for it drives the virtual world in which the simulated submarines, sensors, weapons, countermeasures, opponents, and other objects exist and operate (Fig. 5). The depth and richness of the ORBIS environment endow the SCIL with capabilities for addressing a wide array of problems.

ORBIS is actually a time-stepped expert system that manipulates simulated objects on the basis of the state of the virtual world. It is written in Common Lisp and runs in an X-Windows (UNIX) environment. ORBIS

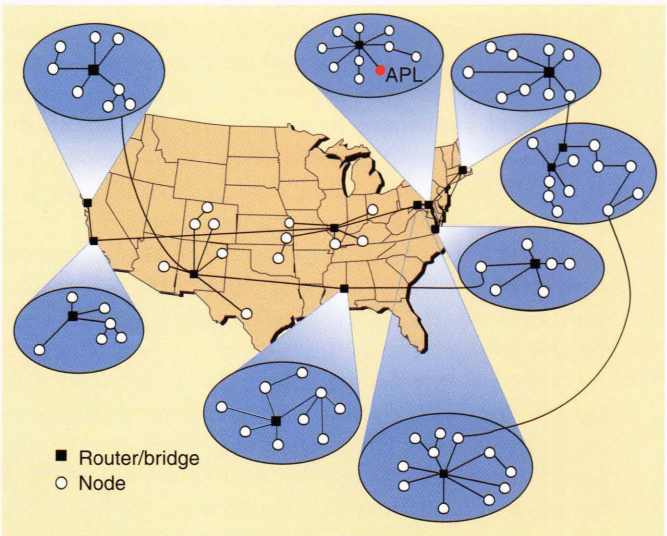


Figure 4. Map of the Defense Simulation Internet (DSI).

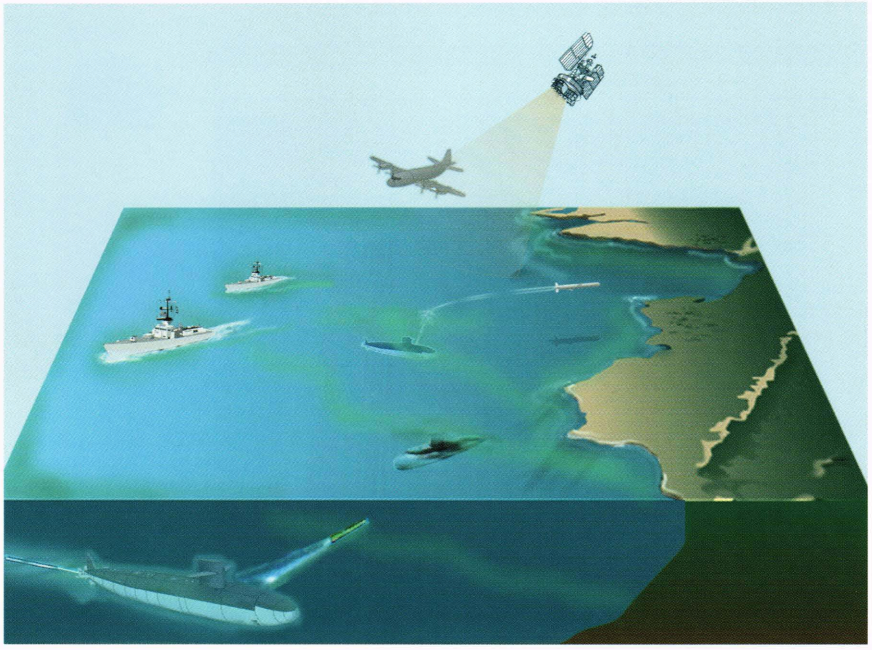


Figure 5. The SCIL's virtual world. This world is generated by ORBIS, which is a time-stepped expert system that manipulates simulated objects.

enables users to create domain-specific synthetic environments or *applications* by providing a set of tools collectively known as the *shell*. Included in the shell are various editors, such as the Rule Editor, Dictionary Editor, Object Tree Editor, and Setup Editor. These tools enable developers to create and control, manually or by rules, the objects that exist in the SCIL's virtual world. After the objects are developed, they can be exercised in real time using a *simulation engine*, which carries out functions such as object state updates and interpretation of rules or manual instructions.[2]

As its name indicates, ORBIS uses an object-oriented approach. This approach has two levels of objects, top-level and dependent, and their states are represented by a collection of *attributes*. *Message variables* can also be associated with an object to allow it to base its state on the state of other objects. The distinction between top-level and dependent objects is that movements of dependent objects generally depend on movements of top-level objects. For example, the position of dependent objects in a submarine or towed by a submarine depends on the motion of the top-level *submarine* object.

ORBIS's name, besides indicating an object-oriented approach, reflects the presence of rules in applications. On the basis of expert experience or *a priori* knowledge, these rules automatically control the actions of objects such as torpedoes through the use of a *reactive planner* approach.[3] This approach takes into account all possible situations prior to simulation time and establishes state patterns that will cause reactions when recognized in the virtual world. The rules are a combination of dictionary variables (i.e., state values or reaction trip-points), rule or activity names, and standard constructs (e.g., IF, THEN, <, >, =). Rules are grouped to form *rule bases*, which are then combined to form *rule base systems*. Reactions specified in the rules take the form of rule base system state adjustments (such as motion goal changes) or object creations.

Once the top-level object types, dependent object types, and rules or manual controls for an application are defined, they are used to form object trees. The trees are assigned an initial state (such as x, y, and z position) and are then grouped to form a *setup*, as conceptualized in Fig. 6. The setup in Fig. 6 represents a possible scenario in the virtual world defined by the objects, the rules, and the SCIL's physical facilities. Included with object trees in a setup are parameters for controlling the setup, such as object removal and end-of-run conditions.

After a setup or scenario is created, it is exercised by manning the SCIL and starting a simulation clock. While time is advancing in the synthetic environment, object states are updated (for example, their positions are changed), and rule-based or manual reactions are stimulated by patterns in the state of the virtual world. During operation in this interactive mode, a graphical representation of the virtual world is provided that includes object positions, motion characteristics, and rules that have been executed or "fired" for automatically controlled objects (see Fig. 7). ORBIS can also run in a background mode that allows collection of Monte

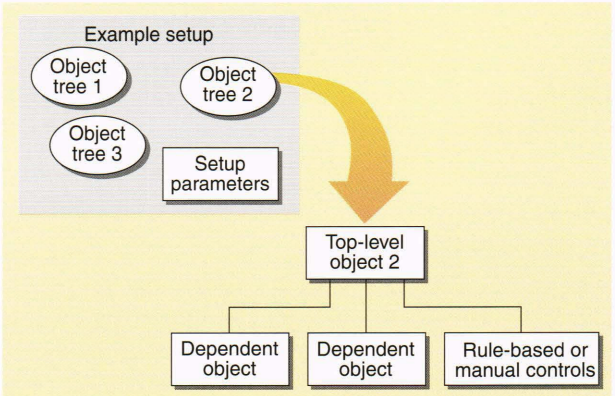


Figure 6. Conceptualized ORBIS setup and object tree structure.

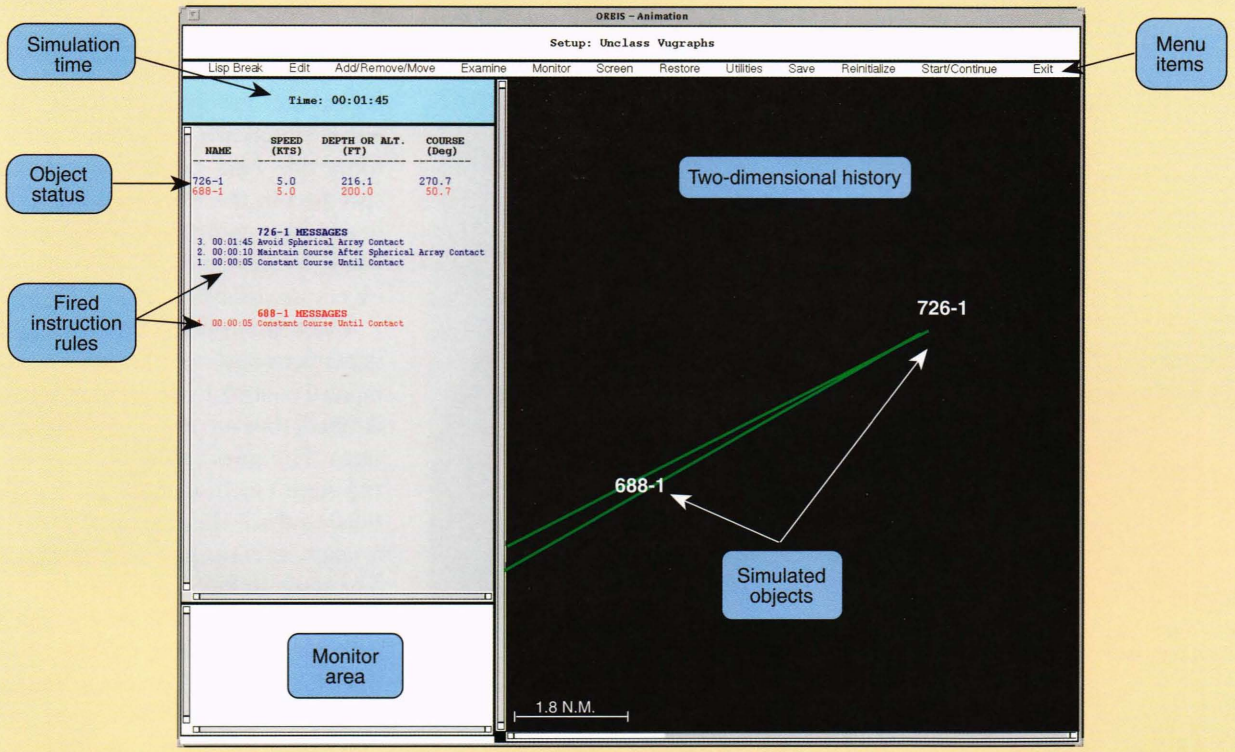


Figure 7. ORBIS graphic display during operation in the interactive mode.

Carlo statistics. This mode can be used for many purposes, such as statistical testing of preliminary tactics developed manually in the SCIL, assessing the utility of emerging technologies, and sensitivity studies.

An important feature of ORBIS is that objects and rules created for one virtual world can be used in another. For example, a 688-class *submarine* object previously developed for a search scenario can also be used in a battle-group support application. The ability to reuse objects and rules permits the ORBIS development team to create an ever-growing library of objects and their associated behaviors. The benefits of this archiving and reuse include faster simulation development, reduced development costs, greater flexibility, and continual enhancement of the ability to address more complex problems.

Objects in ORBIS

ORBIS objects contained in a setup are actually instances (copies) of a particular object type and kind. For example, if there is a *submarine* object type in an application, *726* (i.e., the USS *Ohio*) might be one of the object kinds associated with that type. Figure 6 shows how these copies could be grouped with rule base system instances to form object trees. Figure 8 shows an example of some of the structure that would exist in a simplified *submarine* object type definition. The figure shows that object type definitions include static, dynamic, and graphic attributes as well as message variables. The state of an object instance in a setup consists of a list of specific values for each attribute and message variable in the associated object type definition.

The static attributes in an object type definition are unique for each object kind and represent a database of object characteristics such as Max-Speed. For example, the value for the Max-Speed static attribute associated with the kind *726* in Fig. 8 is 12 kt. The static attribute values associated with each kind can be single values such as Max-Speed, simple pair-lists (such as rudder and advance value pairs in the Advance attribute), or even multidimensional lists.

Dynamic attributes, which are not kind-specific, contain expressions that are evaluated at every time step using the current state of the virtual world. Expressions can be a simple expression, such as the Time-on-Constant-Course expression in Fig. 8, or a function call. In some cases, more than one dynamic attribute is updated using a single expression, as indicated by the call to Update-Dynamic-Variables. In other cases, more than one expression is associated with a single dynamic attribute. This feature can be used to specify a different

SUBMARINE OBJECT TYPE DEFINITION*

Kinds	(688 726)				
Static Attributes					
Max-Speed (knots):	(688 15)		(726 12)		
Max-Depth (feet):	(688 200)		(726 250)		
Advance (degrees, yards):	(688	(5 1500)	(10 1000)	(15 500)	)
	(726	(5 2250)	(10 1500)	(15 750)	)
Transfer (degrees, yards):	(688	(5 1200)	(10 800)	(15 400)	)
	(726	(5 1800)	(10 1200)	(15 600)	)
Tactical-Diameter (degrees, yards):	(688	(5 1350)	(10 900)	(15 450)	)
	(726	(5 2000)	(10 1350)	(15 675)	)
Message Variables					
Submarine-Locations (nm, nm):	((Object-Type Submarine) (Attributes X Y))				
Dynamic Attributes					
Last-Course (degrees):	(Get-From *self* 'Course)				
X, Y, Z, Course, Speed:	(Update-Dynamic-Variables				
(nm, nm, feet, degrees, knots)	(Get-From *self* 'X) (Get-From *self* 'Y) (Get-From *self* 'Z)				
	(Get-From *self* 'Max-Speed) (Get-From *self* 'Max-Depth)				
	(Get-From *self* 'Course) (Get-From *self* 'Speed)				
	(Get-From *self* 'Advance) (Get-from *self* 'Transfer)				
	(Get-From *self* 'Tactical-Diameter)				
	(Get-From *environment* 'Environment-Values)				
	(Get-From *motion-goals-rule-base* 'Ordered-Dive-Angle)				
	(Get-From *motion-goals-rule-base* 'Course-Goal)				
	(Get-From *motion-goals-rule-base* 'Speed-Goal)				
	(Get-From *motion-goals-rule-base* 'Depth-Goal)				
	(Get-From *motion-goals-rule-base* 'Ordered-Rudder)				
	(Get-From *motion-goals-rule-base* 'Turn-Side))				
Time-on-Constant-Course (hours):	If (Get-From *self* 'Course) = (Get-From *self* 'Last-Course)				
	Then (Get-From *self* 'Time-on-Constant-Course) + *time-step*				
	Else 0.0				
Submarine-CPAs (nm):	(Update-CPAs				
	(Get-From *self* 'Submarine-CPAs)				
	(Get-From *self* 'Submarine-Locations))				
Graphic Attributes					
Show in Animation:	(688 (Yes No)) (726 (Yes No))				
Color:	(688 (Red Blue Black Yellow Green))				
	(726 (Blue Red Black Yellow Green))				

*Indicated values were created solely for the purposes of this example.

Figure 8. Example of structure in a *submarine* object type definition. The structure includes static, dynamic, and graphic attributes as well as message variables.

expression for different subsets of object kinds in a type definition. Note that in the Time-on-Constant-Course expression, the Course attribute values from before and after update are used to determine whether a course change has occurred. This expression not only points out how state values from the previous update time can be used to determine state values for the current update time, but also how the update order of the object (top to bottom) is taken into account.

The graphic attributes are used to control the graphical representation of an object in the interactive display. As indicated in Fig. 8, these attributes might be used to specify whether or not the object is displayed graphically in Interactive Mode and, if so, what color to use. For example, in Fig. 7, if the 688-1 object were not displayed, the green bearing lines would be the only indication that the object existed. Since these bearings are part of what the 726-1 object is basing its reactions on, the user could tell whether the 726-1 is reacting correctly on the basis of its sensor inputs as opposed to ground truth.

Message variables, as indicated previously, contain values associated with the state of other objects in the setup. For each object in a setup that matches the specified type, a list of the indicated attributes is constructed. These lists are combined into a larger list and assigned to the associated message variable. In Fig. 8, the object type from which to get messages is *submarine*, and the attributes to return are X and Y. A list of these attributes from each *submarine* object in the setup, not including the current object, is assigned to the message variable Submarine-Locations.

A frequently used function in ORBIS is Get-From, which retrieves a specified attribute or variable from an object or rule base system. In Fig. 8, one attribute source used is the variable *self*, which refers to the object being updated. Another source is the variable *motion-goals- rule-base*. This variable is bound to the rule base system that controls either the movement of the object being updated or its associated top-level object. Figure 9 shows an example of such a rule base system that might be used to control the movement of a *submarine* object. When manual controls are in place, this global variable is bound to a *remote object manager* object, which reflects the inputs or goal changes indicated at operator-controlled consoles.

Rule Bases and Dictionary Variables

Rules are used in ORBIS to recognize patterns in the state of the virtual world and react to those patterns automatically. The rules exist in the two-level structure of the rule base system. In this system, an *activity-level* rule recognizes a situation that the associated top-level object is in and selects an *instruction-level* rule base to carry out the appropriate reactions. In Fig. 9, the activity-level rule base is the Tactics rule base, and the instruction-level rule bases are the Constant Course and Random Maneuver rule bases. The simplified rule base system in this figure might be used to cause a *submarine* object to maneuver periodically to a new, randomly selected course after a random period of time. This rule base could be modified to account for avoidance of contacts by adding a higher-priority avoidance rule at the top of the Tactics rule base and an associated instruction-level rule base. The resultant rule base system would continue to maneuver randomly unless avoiding a contact.

Note that rule base systems have different types just as objects do, and that instances of more than one type of rule base system can exist in an object tree. For

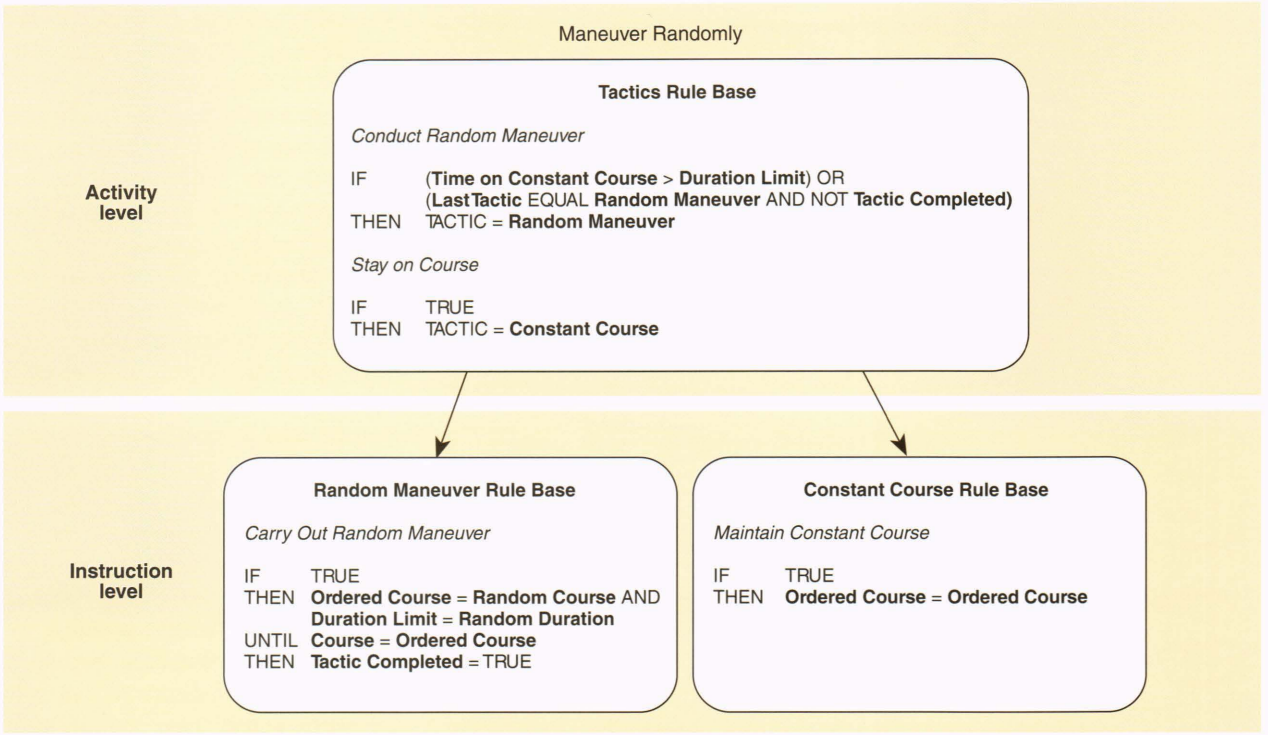


Figure 9. Simplified example of a rule base system, Maneuver Randomly, which could be used to control a *submarine* object. All except the standard constructs are shown in boldface.

example, the type of rule base system shown in Fig. 9 is called a motion goals rule base. It is typically dominant in that other types of rule base systems tend to be dormant until stimulated by a motion goals rule base, as when a submarine's motion goals rule base flags the torpedo fire rule base to conduct a torpedo launch. Because of the dominance of motion goals rule bases, the fired instruction rules shown in the interactive display come only from this type of rule base system. Rules that have fired in instances of other types of rule base systems can be viewed via the Examine menu of the interactive display. (Rule base systems also have names, as indicated by the name Maneuver Randomly in Fig. 9.)

Rule bases are processed by evaluating rules sequentially from top to bottom until an antecedent (IF part) is found that evaluates to TRUE. When this rule is found, it fires, causing an activity to be selected or a reaction to be carried out. For example, in Fig. 9, if the antecedent of the Conduct Random Maneuver rule evaluates to TRUE, the Random Maneuver tactic will be selected and the Random Maneuver rule base will be entered. Note that the antecedent of the last rule in each rule base usually has an expression of TRUE, which causes the rule to fire when encountered.

An interesting control construct that can be used in an ORBIS rule is the UNTIL construct. When an instruction-level rule base exits a time step because an UNTIL clause evaluates to FALSE, the same UNTIL clause will be reevaluated if the same rule base is entered during the next time step. If this UNTIL clause evaluates to FALSE, the rule base is exited and no rules are fired. On the other hand, when the UNTIL clause finally evaluates to TRUE, the subsequent THEN clause is carried out. For example, in Fig. 9, when Course finally equals Ordered Course in the Carry Out Random Maneuver rule, Tactic Completed will be set to TRUE. A THEN-UNTIL pair in ORBIS is called an execution plan, so the reaction specified in an instruction rule is considered to be a series of one or more execution plans. Note that an UNTIL TRUE implicitly exists at the end of a rule that does not end with an UNTIL clause, thus completing an execution plan. Therefore, the reaction specified in the Carry Out Random Maneuver rule consists of two execution plans.

Rules are created from standard constructs, dictionary variables, and rule or activity names. In the Rule Editor, a rule is built by clicking on these mouse-sensitive ingredients (see Fig. 10). The rules shown in Fig. 9 illustrate how these ingredients are combined; all except the standard constructs are shown in boldface. Some of the operations that the standard constructs can be used to perform include

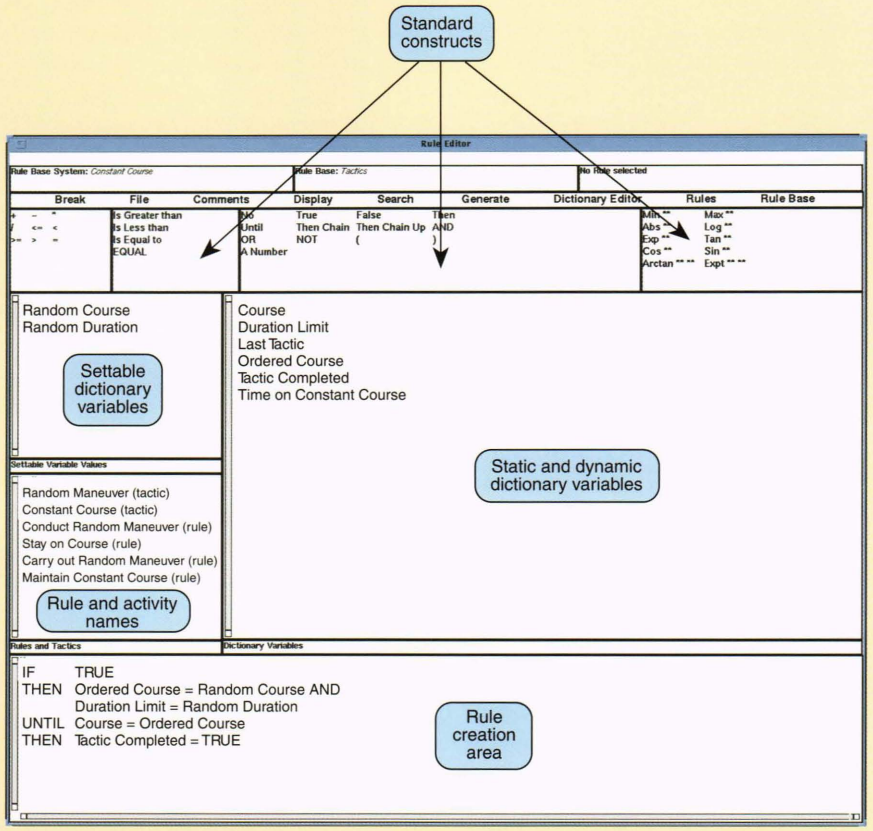


Figure 10. The ORBIS Rule Editor, which permits users to build rules.

1. Numeric comparison
2. Assignment
3. Execution plan definition
4. String comparison
5. Functions (such as max, min, cos)
6. Object generation

The construct applied when generating an object is the Action construct. This construct appears in the rule when the Generate menu command is selected in the Rule Editor. Using this construct, a torpedo launch is represented with the clause Action = Launch Torpedo in a rule. Since generated objects are actually object trees, the torpedo that is generated will consist of a top-level torpedo object, sensor objects, and a motion goals rule base as specified in an initialization structure attached to the Action clause. Default initial values for attributes can be overridden using expressions that reference the state of the virtual world at generation time. For instance, a generated torpedo will have a Course-Goal assigned to it on the basis of the current target solution rather than a default value that is out of context.

As shown in Fig. 9, each rule or activity has a unique name in a rule base system. Their uniqueness allows these names to be referred to in rules, as demonstrated by the Random Maneuver reference in the Conduct Random Maneuver rule. In addition to their software names, all dictionary variables have an English name that is used in the rules to promote readability. An example of this feature is when the Carry Out Random Maneuver rule sets a course goal. The English name used in the rule is Ordered Course, but the actual name of the dictionary variable is Course-Goal.

Each rule base system in ORBIS uses a set of dictionary variables that act as a blackboard of values that are available to all of its rule bases and represent its state. These dictionary variables can be static, dynamic, or "settable." Their definitions can be viewed while in the Rule Editor by using a mouse and key combination to reveal the Dictionary Variable Examiner window, shown in Fig. 11. Static variables have constant values that generally change only when assigned a new value in a rule. An example of this type of variable is Duration Limit, which is used in the Carry Out Random Maneuver rule (see Fig. 9). Dynamic variables have an associated expression that is evaluated at every time step in the same manner as dynamic attributes. As in the dynamic attribute case, multiple dynamic variables can be set within a single function call or expression (see Active-Bearing-MR in Fig. 11). A sample dynamic variable is Time on Constant Course, which might be set using the simple

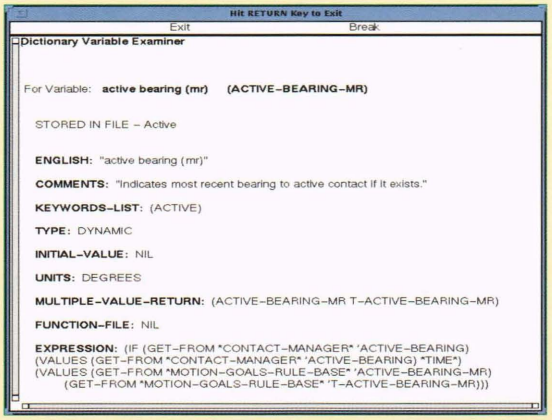


Figure 11. The Dictionary Variable Examiner window, which allows operators to view the dictionary variables used in ORBIS's rule base systems.

expression Get-From *submarine* Time-on-Constant-Course. In this expression, *submarine* is a variable bound to the name of the top-level *submarine* object to which this rule base system is attached. This expression demonstrates how object attribute values can be reflected in a rule base system. However, note that there is a potential delay of one time step associated with this expression (i.e., the rule base system might be updated prior to the *submarine* object). Finally, settable variables are variables that have an associated expression that is evaluated only when the variable is referenced in a rule. This feature reduces computation time when a complex expression is used.

Simulation Parameters and Setups

An ORBIS application usually has various associated setups or scenarios, each of which includes an initial configuration of object trees and a set of simulation parameters. These setups are created using the Setup Editor, which has the same graphical appearance as the interactive display. Adding an object tree in the Setup Editor requires performance of the following:

1. Selection of object tree components
2. Specification of some initial values such as course, speed, and location
3. Selection of expressions to use for multiple-expression dynamic attributes
4. Indication of the desired object tree location

Note that selecting object tree components requires choosing not only the types of objects or rule base systems, but also their kinds or names. When creating the object tree, no restrictions are placed on the object tree structure except that there must be a top-level object, and only one rule base system or object of a specific type can occur in the object tree. This flexibility enables evaluation of unusual combinations, such as placing U.S. sensor types on enemy platforms or vice versa.

Besides adding object trees, the Setup Editor is used to specify preinitialization functions, initialization functions, time increments, object removal conditions, end conditions, and measures of effectiveness. A list of each of these items is defined before entering the Setup Editor, and then selections from these lists are made while in the Setup Editor. Preinitialization and initialization functions are evaluated before the simulation clock is started. The difference between preinitialization and initialization functions is that the former are evaluated before, and the latter after, objects and rule bases are initialized. A preinitialization function might be used to establish a random seed for a particular simulation run before random values are assigned to attributes or dictionary variables. A direct comparison of runs conducted using the same set of random seeds would thus be possible. An initialization function, on the other hand, could be used to place and orient a submarine randomly at the start of a series of simulation runs.

Dynamic-simulation-based time increments are used to advance time in the synthetic environment. When a simulation starts, a default time increment, which is usually large, is applied. However, a different time increment might be used when a significant change in the situation occurs, such as after an initial detection during a hostile submarine encounter. Such a time increment is defined in ORBIS by specifying an expression that will indicate when a detection has occurred. When this expression returns TRUE, the associated time increment is used. If more than one situation-based, time-increment expression returns TRUE, the smallest associated time increment is used. For example, the time increment desired when a torpedo is homing in on a target would be smaller than the desired postdetection time increment.

Removal conditions are used to remove object trees from a simulation. Object trees could be removed for a variety of reasons. For example, an object might be removed when it is hit by a torpedo. The torpedo and anything else in the affected area would also be removed. An object might also be removed when its lifetime expires, as when a torpedo runs out of fuel. End conditions are used to end a simulation run, and they include simulation time constraints (stopping the simulation at a particular simulation time without regard to simulation state), and removal of all significant object trees (as when both submarines in an engagement are sunk by torpedoes).

Measures of effectiveness are used to record observations during simulation runs. These are used during Monte Carlo background runs to collect data for statistical analysis. They can also be used during SCIL simulation runs to collect data for exercise reconstruction. Examples of the types of data collected during background runs include closest points of approach and hits by torpedoes. During SCIL simulation runs, the tracks of each participant and the associated generated objects can be logged.

Once a setup is created, it can be edited, copied, and deleted. Editing a setup involves one of the following:

1. Adding or removing objects
2. Changing initial attribute/variable values
3. Changing the object tree structure
4. Reselecting expressions in multiple-expression dynamic attributes
5. Changing simulation parameter lists

Objects are added in the same manner as when the setup was originally created. Removing objects or changing simulation parameter lists is accomplished by a simple combination of menu selections and mouse clicks. Items 2 through 4 listed in the preceding paragraph are performed using the Object Tree Editor

shown in Fig. 12. This display shows the object tree structure in the upper "tree" window, and information relating to a selected tree node (object or rule base system) in the lower "examiner" window. By clicking on the appropriate category in the examiner window, the user can change the kind or name of any node in the tree as well as the initial attribute/variable values, graphic attribute values, and expression choices of the node. In the tree window, dependent objects or rule base systems can be added to or removed from the object tree.

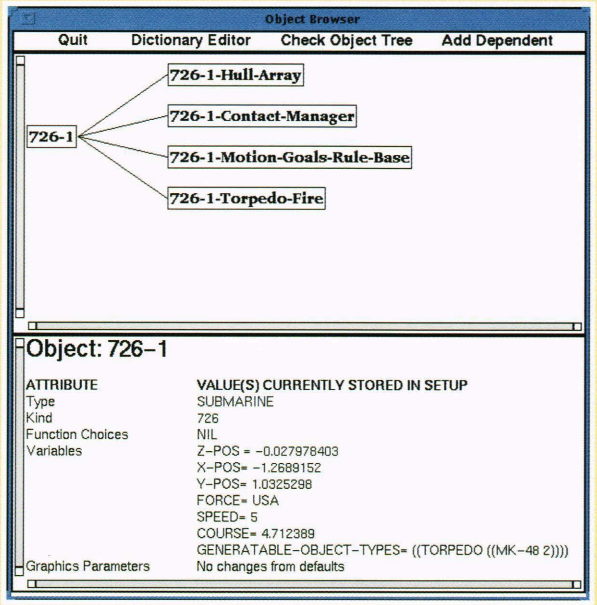


Figure 12. The Object Tree Editor, which is used to edit objects contained in a setup. The object tree structure is shown in the upper "tree" window, and information relating to a selected tree node is shown in the lower "examiner" window.

The interactive operation of ORBIS setups allows a user to observe and manipulate the ground truth of a synthetic environment as it evolves. The graphic interface and functions used are the same as those used by the Setup Editor. A significant difference, however, is that the functions can be applied any time during a simulation run. This feature allows the user to affect the behavior within a virtual world by changing object states or even adding or removing objects. One function available only during interactive operation is the ability to monitor variables, attributes, or expressions during the execution of a setup. Another such function is the Time Restore feature, which can create an instant replay of elapsed events. (This function is not currently available during man-in-the-loop SCIL operations.) With these functions, the user can stop the simulation, restore to an earlier time, edit the setup (including attribute values, variable values, and rule structure), and then restart the simulation from the restored time. This process, known as dynamic editing, promotes rapid development of rule base systems in ORBIS.

ORBIS Application

ORBIS has proved to be a robust tool for creating interactive synthetic environments that exercise emerging technologies and tactics. The state information it passes to the SCIL consoles creates an authentic atmosphere that prompts operators to respond realistically within the synthetic environment. The high level of fidelity enables advanced technologies and tactics to be extensively developed, integrated, and tested in the laboratory, reducing the resources expended for field or sea tests. ORBIS has also been effective when operated as a stand-alone simulation development environment and analyst's tool. Its flexible interactive mode, used for "what if " investigations, coupled with a background mode that can generate Monte Carlo statistics, has proved useful in assessing new technologies and developing control strategies.

THE USES AND USERS OF THE CITEF

The CITEF was originally developed to support the activities of the STD's traditional principal sponsors and programs, such as the SSBN Security Program, the SSBN Survivability Program, and the SSN Security Program. CITEF's resources have accordingly been used in tactics development, advanced sensor concept assessment, exercise and sea-test planning, cost and operational effectiveness assessment, and prototype system development.

As the CITEF's capabilities have grown, however, the facility has been used by other sponsors and programs, primarily the Department of Defense's Advanced Research Projects Agency (ARPA). The first ARPA-related effort was the ARPA/Maritime Systems Technology Office's Maritime Simulation Demonstration (MSD), which took place in September 1993. The MSD was the first high-fidelity distributed simulation of undersea warfare in a major regional contingency scenario.[4] In this demonstration, the SCIL supplied both U.S. and foreign man-in-the-loop-controlled submarines performing Anti-Submarine Warfare missions. The MSD was also the first test of the SCIL as a functioning node on the DSI.

In late 1993, the SCIL also furnished simulated submarines in a strike warfare role for a DIS exercise sponsored by the ARPA/Advanced Systems Technology Office. During the summer of 1994, the SCIL was the site of a demonstration for the ARPA-sponsored Ship Systems Automation Program.

SUMMARY

The SCIL and its associated facilities have proved capable of supporting a wide range of Navy and Department of Defense activities. The activities include ORBIS-based Monte Carlo, man-in-the-loop, and distributed simulations (both internal and DIS-related); software development; war-gaming; sea-test planning and support; technology demonstrations; and system prototype development, integration, and testing. The success of this facility is due to the effective incorporation of key advances in computing, networking, simulation, expert systems, and visualization technologies.

REFERENCES

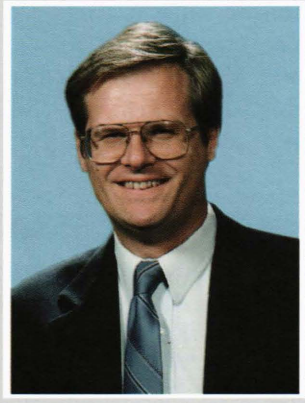
[1] DIS [Distributed Interactive Simulation] Steering Committee, *The DIS Vision: A Map to the Future of Distributed Simulation*, Version 1, IST-SP-94-01, Institute for Simulation and Training, University of Central Florida, Orlando, FL (1994).

[2] Evans, R. B., and Sanders, R. D., "ORBIS—A Tool for Simulation Development," in *Proc. 1994 Summer Computer Simulation Conf.*, San Diego, CA, Society for Computer Simulation (1994).

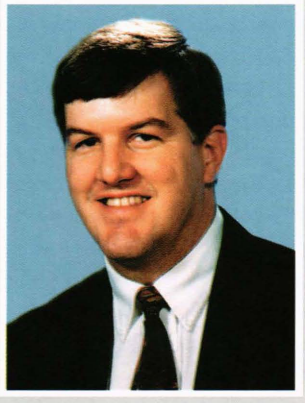
[3] Cohen, P. R., "Architectures and Strategies for Reasoning Under Uncertainty," in *Readings in Uncertain Reasoning*, Morgan Kaufman Publishers, Inc., Amherst, MA (1990).

[4] Root, S. L., and Jackson, J. P., *ARPA Maritime Simulation Overview*, STD-R-2289, The Johns Hopkins University Applied Physics Laboratory, Laurel, MD (1993).

THE AUTHORS



MICHAEL D. DYKTON is an APL Senior Staff Physicist specializing in operations analysis, tactics development, and computer simulations. He received a B.S. degree in astronomy from the University of Maryland in 1979. He worked at the Naval Research Laboratory and EG&G Washington Analytical Services Center before coming to APL. Mr. Dykton joined APL in 1985 and initially worked as an analyst in submarine operations analysis and tactics development. Since 1992, he has been the manager of the Tactical Simulation Project and is responsible for overseeing the development of ORBIS and elements of the SCIL, and for associated technical investigations. In 1994, he was appointed supervisor of the System Modeling and Simulation Technologies section.



ROBERT D. SANDERS is an APL Associate Professional Staff member specializing in computer simulations, tactics analysis, and technology assessments. He received a B.S. in computer science from Tulane University in 1982 and an M.S. in computer science from The Johns Hopkins University G.W.C. Whiting School of Engineering in 1992. From 1982 to 1987, he served in the Navy's nuclear submarine community, leaving as a lieutenant. Mr. Sanders then worked as a subcontractor at the David Taylor Research Center, where he developed and verified thresholds for the Trident submarine monitoring subsystem. In 1991, he joined the Advanced Combat Systems Development Group at APL and is currently involved in a variety of ORBIS-based simulation efforts.