Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

JOINT MODELING AND ANALYSIS USING XMSF WEB SERVICES

Arnold Buss

MOVES Institute
700 Dyer Road
Naval Postgraduate School
Monterey, CA 93943, U.S.A.

John Ruck

Rolands & Associates Corporation
500 Sloat Avenue
Monterey, CA 93940, U.S.A.

ABSTRACT

This paper describes the creation of a new analytical modeling capability by bringing together the Naval Simulation System (NSS) for sea strike and COMBAT^{XXI} for littoral and land warfare modeling. The models are linked by Web services using principles from the Extensible Modeling and Simulation Framework (XMSF). This implementation is an exemplar for a transformational framework for design, development, and integration of simulation models.

1 INTRODUCTION

This paper describes ongoing research and development using a transformational analytical modeling framework. Web services are used in an innovative way to connect multiple model components in a flexible, scalable, extensible architecture. Following the strategic trajectory of the XMSF effort, this work starts first with functioning exemplars, then progresses to supporting tools, and then steps up to world-class modeling challenges, analysis and results. Key sources of functionality for these efforts include the Simkit discrete event simulation application program interface (API) developed by the Naval Postgraduate School (NPS), Naval Simulation System (NSS) developed by SPAWAR Systems Center, San Diego, and COMBAT^{XXI} under development by the Army and Marine Corps at the Army TRADOC Analysis Center, White Sands Missile Range (TRAC-WSMR), which already incorporates Simkit,

The research objectives are to produce new analytic capabilities by connecting diverse tools using Web services. This effort involves software analysis, design, and development to review and upgrade existing code bases (Simkit and NSS) leading to integration of functional capabilities with the Simkit-based COMBAT^{XXI} simulation. The effort also demonstrates the analysis capability of the hybrid tools through design and conduct of an examination of specific operational problems. The work thus spans the areas of Discrete Event Simulation methodology, Operations Research, and distributed programming. In addition to the specific tools of Simkit, NSS, and COMBAT^{XXI}, emerging technologies are critical components, specifically ubiquitous use of XML for data and of web services. Finally, the Open Source model has proved its worthiness as a means by which technically solid and robust standards and practices can emerge. These form the basis for the Extensible Modeling and Simulation Framework (XMSF).Extensible Modeling & Simulation Framework (XMSF)

2 EXTENSIBLE MODELING AND SIMULATION FRAMEWORK (XMSF)

The Extensible Modeling and Simulation Framework (XMSF) provides the technical basis for transformational interoperability via XML interchange, profiles, and recommended practices for web-based modeling and simulation. Broad technical interoperability is provided by open standards, XML-based markup languages, Internet technologies, and cross-platform Web services. XMSF supports diverse distributed modeling and simulation applications. It also enables simulations to interact directly and scale appropriately over a distributed network through composable and reusable model components. In addition to employing mainstream practices of enterprise-wide software development, XMSF provides support for all types and domains of modeling and simulation (constructive, live, virtual, and analytical).

XMSF is a natural foundation because of Open Source philosophy underlying this research. XMSF offers comprehensive support by Open Standards in Web, Internet, and XML technologies. As predicted by an authoritative research workshop, Web services allow self-validating syntax + semantics to achieve cross-cutting interoperability in modeling and simulation. XMSF maintains active working-group efforts in Simulation Interoperability Standards Organization (SISO) (<http://www.sisostds.org>) and Web3D Consortium (<http://www.Web3D.org>) and provides a growing foundation in Web-based open standards.

This research does not utilize the full spectrum of support embodied in XMSF. Specifically, our efforts are focused on the use of simulation for analysis purposes. The simulation methodology most useful for analysis purposes

is Discrete Event Simulation (DES), and the models we utilize all adopt a pure DES world view. Although some interaction with real-time scenarios is envisioned for eventual implementation, the scope of the current effort remains with DES (Blais 2002).

3 DISCRETE EVENT SIMULATION COMPONENTS

This paper focuses exclusively on Discrete Event Simulation (DES) models, which have proven to the be the most useful for performing analysis. The web services that have been implemented in the work are all based on a DES world view. Specifically, both NSS and Simkit adopt the DES world view, and COMBAT^{XXI}, the model that will ultimately form the land asset portion of the joint model, is likewise DES oriented. In fact, COMBAT^{XXI} uses Simkit's discrete event engine and utilizes Simkit's component design for its implementation.

We will now briefly describe, for completeness, the DES world view. Further details can be found in most introductory simulation textbooks, such as Law and Kelton (2000). Following that, we will again briefly discuss Event Graph methodology and the LEGO component framework (Schruben, 1983; Buss 2000; Buss, 2001; Buss and Sanchez, 2002).

3.1 Discrete Event Simulation

Discrete Event Simulation (DES) methodology is based on the concept of state together with a constraint on how state variables change values. This in turn implies a means for advancing the simulation clock in an efficient manner.

The ubiquitous concept of state incorporates a description of the model at a point in time, like a snapshot of the system. Viewed temporally, state variables are quantities that change (or at least have the potential to change) value in the course of a single simulation run. The constraint applied to state trajectories in DES is that they only change values instantaneously. A rule by which states change value is termed an Event, and occurs in zero simulated time. Equivalently, the collection of Events completely describes the possible state transitions, and these transitions occur instantaneously.

A DES is executed with the help of a Future Event List (FEL) or simply "Event List." The Event List is a set of pending future events, sorted in increasing order of scheduled occurrence. Because of the constraint on how states can change value, at any point in time of the simulation there is absolute certainty that no state will change value prior to the next scheduled event. So the fundamental DES algorithm for time advancement works by advancing time to that of the next scheduled event, removing that event from the Event List, and then processing its corresponding state transition.

Another consequence of an Event occurring is the scheduling of additional events or possibly the canceling of some events that had previously been scheduled. When an Event has completed its processing (state transition, scheduling, and canceling), control returns to the Event List, which continues by advancing time to the next scheduled event in the same manner.

The most straightforward way to describe a DES model is using Event Graph methodology, introduced by Schruben (1983), and briefly described next.

3.2 Event Graph Methodology

Event Graph methodology provides a small, stylized, yet extremely powerful way to describe the structure of DES models. An Event Graph model consists of three elements (Schruben 1983; Buss, 2001): Parameters, a collection of variables each of which stays fixed throughout a given simulation run; State Variables, a collection of variables that do change (or have the possibility of changing) in a single simulation run; and an Event Graph, consisting of nodes and edges, with the nodes representing the (instantaneous) state transitions and the edges representing scheduling and canceling relationships between events. The constructs in the Event Graph are shown in Figure 1.

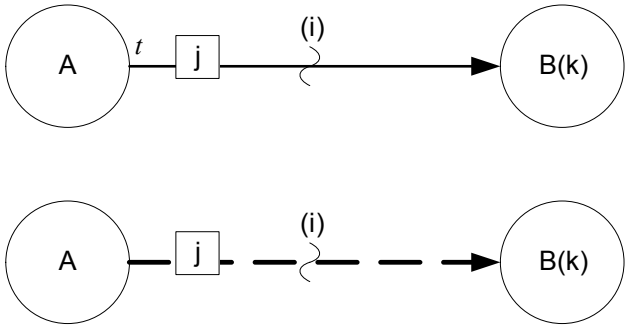

Figure 1: Basic Event Graph Constructs

The top structure in Figure 1 is a scheduling edge and is interpreted as follows. When event A occurs, then if boolean condition (i) is true, then event B is placed on the event list with a delay of time t. When event B occurs, its formal arguments, denoted by k, are passed the values in the expression denoted by j on the scheduling edge. The bottom structure is a canceling edge and is interpreted as follows. When event A occurs, then the first scheduled event named B whose parameters exactly match the expression j is removed from the Event List. If no such event is scheduled when A occurs, then nothing happens and there is no error.

For more information on Event Graph models, Schruben, 1983; Buss, 2001; Buss 2004.

Event Graph Models can be used to create DES models of any degree of complexity, at least in theory. In practice, however, models with large number of nodes become in-

creasingly unmanageable and difficult to modify. One remedy that has been proposed is the use of Event Graph components based on the Listener design pattern. These have been dubbed Listener Event Graph Objects (LEGO), (see Buss and Sanchez, 2002), and are briefly described next.

3.3 LEGO Simulation Components

The listener design pattern is used extensively in modern software design, a most noteworthy example being graphical user interface design using Java's Swing components. The pattern consists of three actors: the event source, the event listener, and the event. Note that "event" in this context is not necessarily the same thing as an "event" as used in Event Graph Models above. Event listener objects register interest in an event source's events. When the source object fires the event, all registered listeners are notified and a reference to the event is passed to the listeners. The power of the listener pattern lies in its inherent loose coupling. Both the source and listener classes can be designed and implemented generically (e.g. using interfaces in Java, or an equivalent construct in other object-oriented languages). Thus, neither the source nor the listener need be "aware" of the other in their design. Note that this is in contrast to the Observer pattern, in which a callback is made from the listener to the source, a pattern that couples the two more tightly than the listener pattern.

A special kind of listener pattern, the SimEventListener pattern, turns out to be a key enabler for creating simulation components. In the SimEventListener pattern, the event is in fact a SimEvent that had been previously scheduled by a SimEventSource object. When the scheduled event occurs, the scheduling object is notified by the Event List and passed a reference to the SimEvent in question. A SimEventListener processes a heard SimEvent exactly as if it had scheduled it itself. This is implemented in Simkit using Java's reflection, so that a modeler only has to define the LEGO class and write an executive controller that does the registration.

The SimEventListener pattern is shown in Figure 2. As many or as few SimEventListeners may be registered for a given SimEventSource.

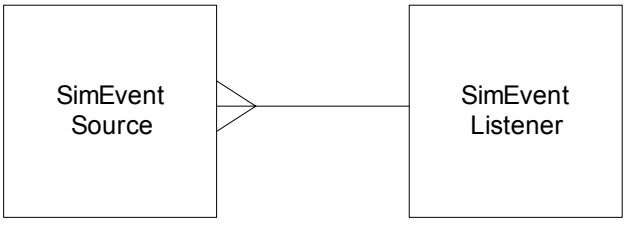

Figure 2: SimEventListener Example

4 JOINT MODELING COMPONENTS

The three components used for this implementation of distributed simulation consist of two existing simulation models and a Discrete Event Simulation (DES) API, Simkit. One existing model, Naval Simulation System (NSS), is particularly focused on modeling naval assets, while the other, COMBAT^{XXI}, specializes in land combat. The DES engine is provided by Simkit. Each component will now be described in more detail.

4.1 Naval Simulation System (NSS)

The Naval Simulation System (NSS) is a closed-loop simulation of naval warfare. It is implemented as an object oriented, stochastic, discrete event simulation (DES). NSS is written in C++ and is a collection of low to medium resolution warfare models. It contains a representation of surface, subsurface, air, ground, and space assets. NSS is C4ISR-centric. Each entity maintains its own perception of the battle space, with commander's orders explicitly represented.

NSS is intended to provide valid warfare models, certified data to populate the models, simulation capability to execute the models over time, and support tools to assist user in scenario setup and analysis of results. It has been used in Naval and Joint operations planning and decision support, C4ISR analyses and assessments, fleet and exercises and experiments, and fleet training.

4.2 Simkit

Simkit is an Open Source engine for creating and executing general Discrete Event Simulation (DES) models. It has been used over the past several years to teach DES to students at the Naval Postgraduate School and has been the basis for more than two dozen masters thesis models. As noted above, Simkit is used as the model engine in COMBAT^{XXI}. Additionally, its listener component implementation is used in COMBAT^{XXI} to implement key functionality, including detection and weapon effects events.

In the Web Service implementation described here, Simkit is primarily used for its discrete event engine for event processing. This is described in more detail in the following section.

4.3 COMBAT^{XXI}

COMBAT^{XXI} is a high-resolution combat simulation currently under development at TRAC White Sands Missile Range. COMBAT^{XXI} is intended to support modern modeling needs with a flexible, object-oriented implementation. COMBAT^{XXI} uses Simkit's Event List and many of Simkit's component features in its design. Work is currently underway to implement the capabilities of COMBAT^{XXI} in joint modeling and analysis using the web-based approach described in this paper. As of this writing, the COMBAT^{XXI} web service component is being developed,. Initial indications are that its use of Simkit will enable the process to go smoothly, with the ultimate joint model playing ground forces in COMBAT^{XXI} and NSS playing the Naval forces.

5 DESIGN OF THE WEB SERVICE

In this design, the simulations interact at the event level. This differs from most previous efforts where the simulations interact by publishing and subscribing to entity attributes and therefore interacting at the entity level. The simulations will make use of a shared event queue in order to accomplish this goal. The simulations will schedule events with the single event queue based on their internal event logic [event graph?]. When an event reaches the top of the queue, the owner and any simulations that have registered to be "event listeners" for the type of event being processed, will be notified to process this event. The use of the single shared event queue eliminates simulation time synchronization issues and eliminates any requirement for the models to be able to roll back time. This also supports an additional goal to eliminate or at least minimize modifications required to existing simulations.

The design is implemented as a number of web services. The Simkit Web Service exposes the event scheduling and canceling methods of Simkit as web services. The various Simulation Web Services act as clients to the Simkit Web Service. The Simulation Web Services are used to wrap an existing simulation to expose a method to process events. The Simkit Web Service acts as a client of the Simulation Web Services in order to pass events to be processes to the simulations making up the composite simulation.

There are currently 2 Simulation Web Services implemented. The NSS Web Service wraps NSS, replacing its simulation engine with one that interacts with the Simkit Web Service. A "Native" Simkit Web Services, that allows an existing simulation written using the Simkit API to interact with the Simkit Web Service.

5.1 The Simkit Web Service

As stated earlier, the Simkit Web Service exposes the simulation engine of the Simkit API as a web service. The Simkit Web Service exposes two web service methods. One allows simulations to schedule events for execution; the other allows simulations to cancel (interrupt) previously scheduled events. After initialization, during which the individual simulations schedule initial events, the Simkit web service sends the next event to be processed to the simulations that will process the event. The mechanism for determining which simulations are notified of which events is discussed below. While processing an event, a simulation may schedule additional events with the Simkit Web Service.

The first step is to initialize the Simkit Web Service. The control program calls the Simkit Web Service initialization method, passing in a control file. The control file (in XML) contains elements to set up the participating simulations and the relationships between the simulations. For each simulation element in the control file, the Simkit Web Service first instantiates a Simulation Instance object. This Simulation Instance represents its corresponding simulation in the Simkit event list. Each Simulation Web Service's initialization method is then called, passing it a control file specific to the underlying simulation. During this initialization, the simulations may schedule their initial events. Next the inter-simulation listeners are set up. The control file contains elements that indicate which simulations will listen to (and therefore process) which events from which simulations. Using the Simkit event listener pattern, the listening simulation's Simulation Instance is set to listen to events of the source simulation through a Simulation Event Filter that will only pass the desired event types.

After initialization is complete, the Simkit Web Service enters the run mode. The run method of the Simkit Web Service starts the simulation loop of the underlying Simkit Event List. The event list is a collection of pending events sorted by event time. The next event on the list is taken and passed to the Simulation Instance object that originally scheduled it. The Simulation Instance then calls the process event method on the web service for the underlying simulation. The Simulation Web Service will be responsible for determining which of the actual simulation entities in a simulation need to process the event. Simulation Web Service event processing will be discussed below. After the original scheduler processes the event, any simulations registered to listen are called to process the event. When there are no more events in the event list, or the time of the next event exceeds the optional stop time set by one of the simulations, the Simkit Web Service run method returns control to the main control program.

5.2 The Simulation Web Service

For each simulation participating in the composite simulation there is a Simulation Web Service. The Simulation Web Service exposes a process event method in addition to its initialization method. The initialization method is specific to the type of underlying simulation and is further discussed in the sections below for the two specific simulation web services. The process event method updates the simulation time for its simulation based on the time of the current event. The event is then passed to the Simulation Proxy for this simulation for processing. The function of the Simulation Proxy is discussed in the next section.

5.3 The Simulation Proxy

As with the Simulation Web Service, some parts of the Simulation Proxy are specific to the underlying simulation. The section will discuss the functionality common to all Simulation Proxies. The Simulation Proxy acts as the client to the Simkit Web Service for its corresponding simulation. In addition the Simulation Proxy is the owner of all events that are sent to and from the Simkit Web Service.

The Simulation Proxy takes requests to schedule events for its simulation and passes them to the Simkit Web Service. Since it is not possible to pass references to

objects via a SOAP message, the event is wrapped and identified with the name of the simulation so that when it is processed, it will be passed to the correct simulation. The simulation specific proxy may need to do some processing of the event to be scheduled prior to passing it to the Simulation Proxy. At a minimum, the simulation that is the source of the event must be able to recover the original event object so that it can be correctly processed. Since the Simulation Proxy is a Simkit "Simulation Entity Base" any events received for processing cause a method with a name corresponding to the method to be called. Further processing of these events is controller by the implementation of the event methods in the simulation specific proxies.

5.4 The NSS Web Service

The NSS Web Service provides a wrapper around NSS allowing it to interact with the Simkit Web Service. Since NSS is not a Simkit based simulation, changes were required to allow NSS to use Simkit as its model engine. In order to prove the concept of NSS using Simkit prior to introducing web services, a version of NSS was developed using the Java Native Interface (JNI), replacing the event queue used by NSS with the Simkit event queue. This required limited changes to NSS. Originally NSS was a Windows executable. The JNI classes were added and NSS was re-built as a Windows dynamic link library (DLL). The main JNI class is the NSS Proxy, which is discussed next. Once is was demonstrated that NSS could be run using the Simkit event queue, then NSS was enabled as a web service.

5.5 The NSS Proxy

When the Simkit Web Service calls the initialization method on the NSS Web Service, the NSS control file is read. The control file contains information on the location of the NSS scenario file and the location of the NSS playback display. Then an NSS specific Simulation Proxy is constructed, during which the information from the control file is stored for later use. The class implementing the NSS Proxy, being a JNI class, is implemented using both C++ and Java. The NSS Proxy is also a Simkit Simulation Entity. Therefore, when the simulation starts, it will process a "Run" event. In a Simkit Simulation, the Run event is a special event that is automatically scheduled for the beginning of the simulation when the entity is constructed. The NSS Proxy Run event simply schedules an initialization event to execute immediately after all of the other entities Run events have initialized. The initialization event calls an initialization method on the C++ (or "native") version of itself.

The native initialization method takes the information stored about the location of the scenario file and playback location and calls the NSS main method. This causes NSS to enter into an initialization state. During this initialization, NSS will conduct internal entity initialization and schedule its initial events. Normally after completing the initialization, NSS would shift to a running state. One of the modifications to NSS was to cause its main method to return rather than continue to the running state. The NSS Proxy then sets the Simkit Web Service's stopping time to the end time of its scenario and schedules an event to conduct cleanup that, in the unmodified NSS, would have been done by NSS shifting from a running state to a cleanup state.

In order to schedule an event with the Simkit Web Service, the NSS Proxy must first wrap the event and event data in an event that will be understood in the composite simulation. The NSS Proxy also stores the original event locally to prevent having to send data not needed by the composite to the Simkit Web Service. The event wrapping mechanism provides a key to retrieve the original event when it is time to process the event. The section below on Data Translators contains more detail on the method of wrapping events.

When an event that NSS will process occurs, it is passed to the NSS Web Service from the Simkit Web Service. The NSS Proxy is then called to retrieve the original event. If the event being process was not originally scheduled by NSS an Event Adapter object is used to translate the non-NSS event so that it can be properly processed by NSS. The Event Adapter is further discussed below.

There is a special NSS event mentioned previously that is processed at the end of the scenario in order to allow the NSS code to perform any required cleanup.

5.6 The Native Simkit Web Service

The purpose of the Native Simkit Web Service is to allow simulations written using the Simkit API to be run in the web services environment. The Native Simkit Web Service is a Simulation Web Service that will, based on information in the control file, start a Simkit based simulation by executing the main method of the given class. This is the same main method that is run from the command line to execute the simulation in its original implementation. To accomplish this, calls to Simkit simulation engine are redirected to the Simkit Web Service.

Similar to NSS, the Native Simkit Web Service wraps events to avoid sending data to the Simkit Web Service that will only be used by the Simkit simulation. Again, the event wrapping mechanism is discussed below.

5.7 Data Translators

While the use of standard transport and data encoding (SOAP and XML) methods solves the syntactic problems of interoperability, the challenge of semantic differences among data still remains. To overcome the semantic differences, we introduce the concept of data and event translators. Event based simulations may define hundreds of unique events. Each of these events may also have unique data objects associated with it. Some method was needed

to translate the meaning of the events and data so that they are understood by all simulations in the composite. However, there are many (if not most events) that do not need to be processed by any but the source simulation and therefore do not need to be translated.

The design defines a default event for each simulation to represent those events that are not translated. To other simulations, these non-translatable events appear to all be of the same type. This allows these events to be ignored outside of their source simulation.

For events that are made available to other simulations, translation is required. Each Simulation Proxy maintains a registry of local event names that are to be translated. This includes a mapping between the local name and the name of the event exposed to the composite. In addition, any data available to the event has a data translator object registered. This data translator takes data that is part of the original event and extracts data that will be made available with the event to the composite. One type of translation that may be necessary is the conversion of units of distance and time as each simulation and the composite may be using different unit of measure systems.

5.8 Event Adapters

Through the inter-simulation listener pattern, a simulation will be asked to processes an event translated from another simulation. In some cases, the architecture of the simulation may allow an entity in the simulation to directly handle the event. In other cases, an Event Adapter is required. An event adapter will implement the event it is responsible for and listen to events from the Simulation Proxy. The adapter can then do any translation of the composite event and event data as needed to support local processing of the event.

5.9 The Static Data Problem

During developmental testing, issues arose that were the result of the architecture of web service containers. In one case, an error inside of an existing simulation caused the Tomcat web service container process to exit. In another case, when a composite simulation was rerun on the same Tomcat server, the state of the simulation was not correctly restored to the starting state of the simulation.

In most cases, calls to web service methods are considered stateless calls. In other words, the current method invocation does not depend on calls made prior to it or other methods. In the case of this project, that is obviously not the case. This statelessness is reflected in the architecture of the Tomcat server in that when the simulations are initialized, they are not started as a separate process, but are run as the same process as the server. In the case of the Java based portion of the simulation, it is run in the same Java virtual machine (JVM) and is initiated as a method call from the servlet container. Therefore, there is no concept of the beginning or end of a simulation execution. In the case of NSS, once the NSS dynamic library is loaded, is stays loaded. In the case of Java based code, once classes are loaded, they stay loaded. This implies that any static data is only initialized the first time a simulation is run. Second and subsequent executions are done with any static data in an unknown state.

The work around for this problem was to implement a web service (the Tomcat Controller Web Service) that allows the controller software to start and stop instances of the servlet container. Therefore, when the controller desires to start a composite simulation run, it directs the Tomcat Controller Web Service to start additional Tomcat processes. When the run is complete, it directs the Controller Web Service to shutdown the processes.

5.10 Technology Used to Implement the Web Services

In keeping with the spirit of XMSF, it was desired to use some form of XML messaging implemented by Open Source software. The web service calls between the simulations and the Simkit Web Service were implemented using SOAP messaging. Specifically, the Apache Axis SOAP implementation was used. The web services were deployed in the Apache Tomcat servlet container.

6 SUMMARY AND ONGOING WORK

We have described a new way of implementing distributed DES models using XML and web services. Currently, the web services discussed above have been implemented and tested. The final simulation web service using COMBAT^{XXI} is in development as of this writing. Although COMBAT^{XXI} uses Simkit's Event List engine, enabling considerable reuse of effort, work is still needed to create a fully featured implementation.

ACKNOWLEDGMENTS

The authors wish to acknowledge insights and informative discussions from colleagues, particularly Don Brutzman, Don McGregor, Curt Blais, Rick Goldberg, and Michael Bailey. This research was sponsored by OPNAV (N81) as part of the World Class Modeling initiative. This support is gratefully acknowledged.

REFERENCES

Blais, C 2002. Extensible Modeling and Simulation Framework (XMSF) Exemplars in Analytic Combat Modeling, 2004 Spring Simulation Interoperability Workshop.

Buss, A. 2000. Component-Based Simulation Modeling. In *Proceedings of the 2000 Winter Simulation Confer-*

ence, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds. 964-971. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Buss, A. 2001. Discrete Event Programming with Simkit. *Simulation News Europe* (32/33): 15-25.

Buss , A. and P. Sanchez. 2002. Building Complex Models with LEGOs (Listener Event Graph Objects). In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 732-737. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Buss, A. 2004. Simkit Analysis Workbench for Rapid Construction of Modeling and Simulation Components. 2004 Fall Simulation Interoperability Workshop.

Law, A. M., and W. D. Kelton. 2000. *Simulation Modeling and Analysis*, 3rd edition. New York: McGraw-Hill.

Schruben, L. 1983. Simulation Modeling with Event Graphs, *Communications of the ACM* 26: 957-963.

AUTHOR BIOGRAPHIES

ARNOLD BUSS is a Research Assistant Professor in the MOVES Institute at the Naval Postgraduate School. He received his MS in Systems Engineering from the University of Arizona and his PhD in Operations Research from Cornell University. His research interests include Discrete Event Simulation and component-based modeling. His e-mail address is <abuss@nps.edu>.

JOHN RUCK is a Software Engineer with Rolands and Associates Corporation in Monterey California. He received his BS in Biomedical Engineering from Tulane University and his MS in Operations Research from the Naval Postgraduate School. His e-mail address is <jlruck@nps.edu>.