WEB-BASED SIMULATION IN SIMJAVA USING REMOTE METHOD INVOCATION

Ernest H. Page
Robert L. Moose, Jr.
Sean P. Griffin

The MITRE Corporation
1820 Dolley Madison Boulevard
McLean, VA 22102, U.S.A.

ABSTRACT

An investigation is underway regarding technologies to support the design, development and use of distributed, web-based simulations. As part of this investigation the Simjava simulation-support package has been extended to utilize the Remote Method Invocation facilities of the Java Development Kit (JDK) 1.1. Current efforts with Simjava are described and future research directions are outlined.

1 INTRODUCTION

The topic of *web-based simulation* spans a variety of areas within the field. (See Page et al. (1997) for an online a survey of web-based simulation.) Fishwick (1996) discusses many potential impacts of web technologies on simulation, giving particular attention to three areas: (1) education and training, (2) publications, and (3) simulation programs. With respect to education and training, the web offers storage and retrieval of supporting material far exceeding the capacity of CD-ROMs or diskettes that may be packaged with a textbook. Such capacity permits more extensive use of resource-intensive media such as audio and video. The learning environment, as a result, tends to be highly interactive. With respect to publications, the web offers new and convenient mechanisms for submitting, refereeing and disseminating research results. The web will impact the traditional revenue process for technical journals, and already is causing changes in our copyright laws (Samuelson 1996). However, it is the last category in Fishwick's discussion that has received the most attention in the web-based simulation area and, arguably, represents the predominant use of the term.

Web-based simulation *programs* generally fall into two categories. Simulation programs that can be accessed remotely through web browsers and forms-based CGI scripts comprise the first category. Typically, these simulations allow the user to tailor (via the forms interface) model execution parameters such as mean service times and arrival rates, number of model replications, and so forth. A single copy of the simulation runs on a server and passes the results of model execution to the invoking client. The second category of web-based simulation programs represents a variation on the first, but with the added feature of code mobility afforded by such network programming languages as Java™. Here, the simulation executes on the client rather than the server. Java-based simulation-support libraries are emerging that permit the creation of simulation programs as Java applications and applets. Among these are Simkit (Buss 1996), JavaSim (Little 1996), JSIM (Nair, Miller and Zhang 1996) and Simjava (McNab and Howell 1996). One of the primary advantages of these packages is that they permit network-based simulation models to be developed using established conceptual frameworks. This paper describes work being undertaken to extend the Simjava package to utilize the Remote Method Invocation (RMI) facilities of JDK 1.1. This extension enables the construction of distributed, web-based simulations using Simjava.

2 DISTRIBUTED SIMULATION INFRASTRUCTURES

An inherent characteristic of the WWW is its distributed nature. Therefore, the marriage of web-based simulation and distributed simulation seems a natural one. Distributed simulation dates to (Bryant 1977; Chandy and Misra 1979; Chandy and Misra 1981) and has evolved into an active research area that includes an annual workshop on parallel and distributed simulation (PADS). The focus of PADS research has historically been centered on parallel and distributed implementations of sequential simulation models that result in *speedup* of model execution at runtime.

Additionally, distributed simulation has been the

focus of the U.S. Defense industry for well over a decade. Here the motivation for distributed computation is not speedup but *interoperation.* These defense-related efforts originated with SIMNET (Alluisi 1991) and evolved into the Distributed Interactive Simulation (DIS) protocol initiative (Voss 1993) and the Aggregate Level Simulation Protocol (ALSP) (Page, Canova and Tufarolo 1997; Weatherly, Wilson and Griffin 1993; Weatherly et al. 1996). DIS provides the environment for networking human-in-the-loop (HIL) simulators. ALSP extends the DIS philosophy to permit the interoperation of higher-level, so-called *aggregate,* simulations and exercise drivers. The primary application of ALSP, the Joint Training Confederation (JTC) has been used to support several large-scale command post exercises annually since 1992.

Both DIS and ALSP are nearing their respective ends of service. The Defense Modeling and Simulation Office (DMSO) has sponsored the definition and development of the High Level Architecture (HLA) for modeling and simulation (M&S). The HLA has been defined to "facilitate the interoperability among simulations and promote reuse of simulations and their components" (DMSO 1996, p. 1). In a 1996 memorandum signed by the U.S. Undersecretary of Defense for Acquisition and Technology Dr. Paul Kaminski, the HLA is endorsed as the standard for all U.S. Department of Defense (DoD) M&S (DoD 1996). The HLA standard supersedes both ALSP and DIS and all DoD M&S efforts must comply with the HLA, receive a waiver, or be retired by 2001.

A primary component of the HLA is its Runtime Infrastructure (RTI). The HLA RTI implements a set of services – defined by the HLA Interface Specification (DMSO 1997) – which are invoked by applications (so-called *federates*) to coordinate the management of both data and time within a distributed simulation environment. (The RTI also invokes services that *HLA compliant* federates must provide.) One of the strengths of the HLA (and the RTI) is its generality. The services it defines (and implements) enable the construction of a well-defined distributed simulation environment from a collection of highly disparate applications, from real-time human-in-the-loop simulators, to faster-than-real-time discrete event simulations, to *instrumented* live vehicles, to simple Monte Carlo models.

One of the prices paid for this generality, however, is a lack of conceptual framework (CF) support. The CF imposed by the HLA, somewhat akin to that provided by an operating system, is typically unlike the CFs available to many candidate HLA applications in their standalone form. One of the activities of the MITRE web-based simulation research program is a collaboration on the design and development of a Java-based RTI. The realization of an RTI in Java will provide the code mobility necessary to provide a platform for distributed, web-based simulation. However, the addition of CF support requires a thorough understanding of the relationship between the OS-level services provided by the RTI and the CF-level operations available to simulations in their standalone form. The remainder of this paper describes the early stages of an investigation regarding the feasibility of providing CF support within a distributed, web-based simulation environment.

3 SIMJAVA

Simjava is a discrete event simulation package, authored by Ross McNab and Fred Howell, that is written in Java and conceptually based on the Sim++ library for C++ (McNab and Howell 1996). A companion package, Simanim, allows the construction of animated Simjava applets. Naturally supportive of the process interaction conceptual framework, a Simjava simulation typically consists of a collection of objects (from the `Sim_entity` class) each of which runs in its own thread within a Java Virtual Machine. Objects are connected via ports (from the `Sim_port` class) and interact by sending and receiving events (from the `Sim_event` class) along these ports. A static class, `Sim_system`, controls the object threads, coordinates the advance of simulation time, and maintains the event queues. Construction of a Simjava program typically involves four steps (McNab 1996):

1. Describing the behavior of the simulation objects by extending the standard `Sim_entity` class and overriding the `body()` method.
2. Instantiating objects via `Sim_system.add(`*obj*`)`, and defining local ports using `add_port()`.
3. Establishing inter-object communication paths with `Sim_system.link_ports()`.
4. Invoking `Sim_system.run()`.

Time flow is regulated using four methods on `Sim_entity`: (1) `sim_schedule()` schedules an event for an object linked to a designated port, (2) `sim_hold()` causes an object to be suspended for a specified duration of simulation time, (3) `sim_wait()` causes an object to wait for the next (future) event posted to its input port(s) by a `sim_schedule()` call, and (4) `sim_select()` allows an object to process events from its *deferred* queue – events that arrive while the object is executing a `sim_hold()` call.

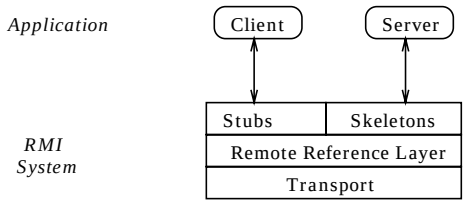

Figure 1: Java RMI Architecture

As a simple illustration, consider Example 1 from the Simjava release documentation (McNab 1996) reproduced here in Appendix A. In this example, a single source schedules arrivals for a single sink. The sink acknowledges each arrival from the source.

4 JAVA REMOTE METHOD INVOCATION

A Remote Method Invocation (RMI) capability has been included with the Java Development Kit (JDK) release 1.1 (Sun 1997). Remote Method Invocation is the object-oriented analogue of the traditional Remote Procedure Call (RPC) for distributed computation. In the Java distributed object model, a *remote object* is one whose methods can be invoked from another Java Virtual Machine, potentially on a different host. An object of this type is described by one or more *remote interfaces,* which are Java interfaces that declare the methods of the remote object. All remote interfaces extend, either directly or indirectly, the interface `java.rmi.Remote`. A method in a remote interface must: (1) declare `java.rmi.remoteException` in its throws clause (in addition to any application-specific exceptions), and (2) declare remote objects passed as an argument or return value as the remote interface instance, *not* the class of the actual remote object, which is known as the *implementation class.* Architecturally, the RMI system consists of three layers:

1. The *stub/skeleton layer* – client-side stubs (proxies) and server-side skeletons.
2. The *remote reference layer* – remote reference behavior (such as object invocation).
3. The *transport* layer – connection set up and management, and remote object tracking.

The relationship between these layers is noted in Figure 1, an illustration from (Sun 1997).

In this context, a *server* is any object whose methods are invoked remotely; a *client* is an object that invokes a remote method. Thus, it is easy to envision applications of RMI (including distributed simulation) in which many clients and servers exist. Furthermore, a given object may – and commonly will – function as both a server and a client. Such an object will both invoke remote methods and have its methods invoked remotely. (The example in Section 6 relies on dual-function objects of this nature.)

A client invoking a method on a remote server object actually makes use of a *stub* (or proxy) for the remote object. The stub, an implementation of the remote interfaces of the remote object, forwards invocation requests to the server via the remote reference layer. The skeleton for a remote object makes an up-call to the remote object implementation which carries out the actual method call. The return call is sent back through the skeleton, remote reference layer, and transport on the server side, and then up through the transport, remote reference layer, and stub on the client side. Stubs and skeletons handle all marshaling for arguments and return values.

Also notable in the Java RMI: The remote interface must be public; the implementation class must create and install a security manager (even for Java applications); nonremote arguments to, and results from, a remote method invocation are passed by copy rather than reference.

5 ADDING REMOTE METHOD INVOCATION TO SIMJAVA

The task of adding RMI capabilities to Simjava can be described in a straightforward manner. The primary changes made to Simjava to incorporate RMI functionality are highlighted here. A comprehensive presentation of these details is given in (Griffin, Page and Moose 1997). The first step is to establish a conceptual architecture for distributed Simjava applications (and applets). A simple master-slave architecture provides the baseline for our Simjava modifications. In this architecture, a master server that encapsulates the `Sim_system`, coordinates the activities of objects (`Sim_entities`) that are distributed across the network. Given this architecture, it is evident that remote interfaces must be constructed for the `Sim_system` and `Sim_entity` classes. Entities create ports and the `Sim_system` links them together. Therefore `Sim_port`, as well, must have a remote interface. Likewise, entities create events which are manipulated by the `Sim_system`. Therefore, four remote interfaces must be defined: `Sim_systemIF`, `Sim_entityIF`, `Sim_portIF`, and `Sim_eventIF`. Note that these four interfaces are sufficient to enable the implementation of the example given in Section 6 and a variety of Simjava applications. Use of Simjava applets requires interfaces for several objects in the Simanim package. Refer to (Griffin, Page and Moose 1997) for details. For purposes of illustration, the

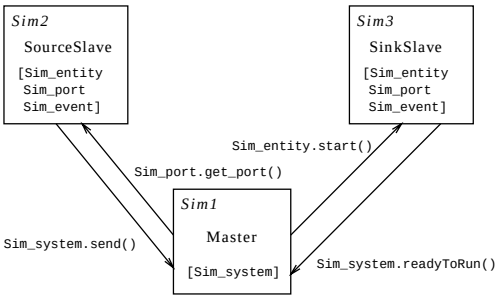

Figure 2: Simple Master-Slave Architecure

code for `Sim_portIF` is given below.

```
package eduni.simjava;

import eduni.simjava.*;

public interface Sim_portIF extends java.rmi.Remote {
  void connect(Sim_entityIF dest)
    throws java.rmi.RemoteException;
  void set_src(int s) throws java.rmi.RemoteException;
  int get_dest() throws java.rmi.RemoteException;
}
```

Since only objects, and not classes, can be invoked remotely, `Sim_system` must be modified to make its interface methods instance methods. The master program creates an instance of `Sim_system` and places it in the RMI registry. In the original Simjava package `Sim_entity` extends `Thread`. To be remotely accessible, `Sim_entity` must extend `rmi.remoteObject` (or one of its realizations, in our case `rmi.server.unicastRemoteObject`) in addition to implementing `Sim_entityIF`. Since Java does not permit multiple inheritance, `Sim_entity` can be modified to implement `Runnable` by providing a `start()` method as follows:

```
public void start()
  throws java.rmi.remoteException
{
  new Thread(this).start();
}
```

This code creates a thread and runs an instance of the creating object in the thread. The `thread.start()` method invokes the `run()` method of the creating `Sim_entity`.

6 EXAMPLE

Using the architectural concept described in the previous section a distributed version of the example in Appendix A may be defined as depicted in Figure 2 (annotated with representative remote method calls).

The master program, running on a SUN SparcStation *Sim1,* contains the single method, `main()`, which accepts a commandline argument representing the number of distributed clients (in this case two), creates an instance of `Sim_system` and binds it in the RMI registry. Following the RMI conventions, a single object is used to bootstrap the system. Remote references for the remaining objects are acquired through passed parameters and return values.

```
class Master {
  public static void main(String args[]) {

    if (args.length < 1) {
      System.out.println("Usage:  Example1 <numClients> ");
      System.exit(1);
    }

    System.setSecurityManager(new RMISecurityManager());

    try {
      eduni.simjava.Sim_system simSystem = new
        eduni.simjava.Sim_system(Integer.parseInt(args[0]));
      Naming.rebind("//sim1:1099/SimSystem", simSystem);
    } catch (Exception e) {
      System.out.println("M: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

The Source and Sink programs, running on Sun SparcStations *Sim2* and *Sim3* respectively, each contain their constructor as in Appendix A which is extended to accept an instance of the `Sim_system` (through the interface type). Port creation and addition is wrapped in a `try-catch` block for the `java.rmi.RemoteException`. The `body()` methods are unchanged. Each has a `main()` method that installs a security manager, retrieves the `simSystem` instance from the RMI registry, and instantiates itself. Finally, the `add()` and `readyToRun()` methods are invoked on `simSystem`. Once the number of calls to `readyToRun()` equals the number of expected clients, a user-supplied method that links the entity ports is invoked by `simSystem`. The `main()` method for `SourceSlave` is given below. The complete source for this example is available from http://ms.ie.org/websim.

```
class SourceSlave extends Sim_entity {
.
.
.
  public static void main(String args[]) {

    Sim_systemIF simSystem;

    System.setSecurityManager(new RMISecurityManager());

    try {
      simSystem = (Sim_systemIF)
        Naming.lookup("//sim1:1099/SimSystem");
      SourceSlave sourceSlave = new SourceSlave("Sender",
        1, SourceSlave.SRC_OK, simSystem);
      simSystem.add((Sim_entityIF)sourceSlave);
```

```
      simSystem.readyToRun();
    } catch (Exception e) {
      System.out.println("S: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

7 CONCLUSIONS

Distributed simulation has been a focus of U.S. DoD simulation research and development for well over a decade. Unlike the parallel and distributed simulation (PADS) community, however, the objective of distributed computation within the DoD is typically not speedup, but *interoperability.* Web-based simulation is a topic that encompasses a wide range of areas in simulation, including education and training, publication, as well as simulation programs. The marriage of distributed simulation and web-based simulation seems a natural one, and initiatives are underway to incorporate web-based code delivery into the next-generation DoD standard, the High Level Architecture. However, the services defined for the High Level Architecture resemble Operating System-level calls. The conceptual framework support available in modern simulation programming languages (SPLs) and simulation support environments (SSEs) is absent in in the HLA Runtime Infrastructure (RTI). An implementation of the RTI in Java will provide code mobility, and web-based invocation, but the integration of conceptual framework support requires additional steps be taken.

Fully extending Simjava to utilize the Remote Method Invocation capabilities of JDK 1.1 would represent one "proof of concept" for *transparent* distributed, web-based simulation. The efforts described in this paper represent the first steps in that direction. Ongoing and future work in this area include: (1) the further extension of RMI to Simjava applets; and (2) the distribution of `Sim_system` to use conservative and optimistic synchronization protocols. While the primary goal of distribution is interoperability rather than speedup, the addition of these PADS synchronization mechanisms will permit the accommodation of a wider range of applications and objectives, and will be necessary, in many cases, in order to meet minimum performance requirements.

APPENDIX A: SIMJAVA SOURCE FOR EXAMPLE 1

```
import eduni.simjava.*;
class Source extends Sim_entity {
  private Sim_port out;
  private int index, state;
  public static final int SRC_OK = 0;
  public static final int SRC_BLOCKED = 1;

  public Source(String name, int index, int state) {
    super(name);
    this.index = index; this.state = state;
    out = new Sim_port("out");
    add_port(out);
  }
  public void body() {
    Sim_event ev = null;
    int i;
    for (i=0; i<100; i++) {
      sim_schedule(out,0.0,0);
      sim_wait(ev);
      sim_hold(10.0);
      sim_trace(1,"C Src loop index is "+i);
    }
  }
}

class Sink extends Sim_entity {
  private Sim_port in;
  private int index, state;
  public static final int SINK_BLOCKED = 0;
  public static final int SINK_OK = 1;

  public Sink(String name, int index, int state) {
    super(name);
    this.index = index; this.state = state;
    in = new Sim_port("in");
    add_port(in);
  }
  public void body() {
    Sim_event ev = null;
    int i = 0;
    while(true) {
      i++; if(i>50) break;
      sim_wait(ev);
      sim_hold(1.234);
      sim_schedule(in,0.0,1);
    }
  }
}

class Example1 {
  public static void main(String args[]) {
    Sim_system.initialise();
    Sim_system.add(new Source("Sender", 1, Source.SRC_OK));
    Sim_system.add(new Sink("Receiver", 2, Sink.SINK_OK));
    Sim_system.link_ports("Sender", "out", "Receiver", "in");
    Sim_system.run();
  }
}
```

REFERENCES

Alluisi, E.A. 1991. The Development of Technology for Collective Training: SIMNET, a Case History. In *A Revolution in Simulation: Distributed Interaction in the '90s and Beyond,* L.D. Voss, 168-187. Arlington: Pasha Publications.

Bryant, R.E. 1977. Simulation of Packet Communications Architecture Computer Systems. MIT-LCS-TR-188, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Buss, A.H. and K.A. Stork. 1996. Discrete Event

Simulation on the World Wide Web Using Java. In *Proceedings of the 1996 Winter Simulation Conference,* ed. J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain, 780-785, Coronado, CA, 8-11 December. (Simkit homepage: http://131.120.142.115/~stork/simkit_home.html)

Chandy, K.M., and J. Misra. 1979. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering* SE-5(5):440-452.

Chandy, K.M., and J. Misra. 1981. Asynchronous Distributed Simulation Via a Sequence of Parallel Computations. *Communications of the ACM* 24(11):198-206.

Defense Modeling and Simulation Office (DMSO). 1996. High Level Architecture for Modeling and Simulation Management Plan, Version 1.7, April.

Defense Modeling and Simulation Office (DMSO). 1997. High Level Architecture Interface Specification, Version 1.1, February.

Fishwick, P.A. 1996. Web-Based Simulation: Some Personal Observations. In *Proceedings of the 1996 Winter Simulation Conference,* ed. J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain, 772-779, Coronado, CA, 8-11 December. (http://www0.cise.ufl.edu/~fishwick/tr/tr96-027.html)

Griffin, S.P., E.H. Page and R.L. Moose. 1997. Implementation Notes for a Distributed Simjava. MITRE Technical Report, The MITRE Corporation, McLean, Virginia. (Report number pending.)

Little, M.C. 1996. JavaSim Homepage. (http://marlish.ncl.ac.uk:8080/JavaSim/)

McNab, R. 1996. A Guide to the Simjava Package, Department of Computer Science, University of Edinburgh, UK. (http://www.dcs.ed.ac.uk/home/hase/simjava/simjava-1.0/)

McNab, R., F.W. Howell. 1996. Using Java for Discrete Event Simulation, In *Proceedings of the Twelfth UK Computer and Telecommunications Performance Engineering Workshop,* 219-228, University of Edinburgh, UK. (http://www.dcs.ed.ac.uk/home/hase/simjava/UKPEWpaper.ps)

Nair, R.S, J.A. Miller, and Z. Zhang. 1996. Java-Based Query Driven Simulation Environment, In *Proceedings of the 1996 Winter Simulation Conference,* ed. J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain, 786-793, Coronado, CA, 8-11 December. (http://www.cs.uga.edu:80/~rajesh/postscript/nair.ps)

Page, E.H., B.S. Canova, J.A. and Tufarolo. 1997. A Case Study of Verification, Validation and Accreditation for Advanced Distributed Simulation. *ACM Transactions on Modeling and Computer Simulation* 7(3), to appear.

Page, E.H., T.M. Crews, S.L. Rother, and S.P. Griffin. 1997. Survey of Web-Based Simulation. (http://ms.ie.org/websim/survey/survey.html)

Samuelson, P. 1996. Regulation of Technologies to Protect Copyrighted Works. *Communications of the ACM* 39(7):17-22.

Sun Microsystems, Inc. 1997. Java Remote Method Invocation Specification, Mountain View, CA, February.

U.S. Department of Defense (DoD). 1996. DoD High Level Architecture (HLA) for Simulations, Memorandum signed by USD(A&T), September.

Voss, L.D. 1993. *A Revolution in Simulation: Distributed Interaction in the '90s and Beyond.* Arlington: Pasha Publications.

Weatherly, R.M., A.L. Wilson, and S.P. Griffin. 1993. ALSP – Theory, Experience, and Future Directions. In *Proccedings of the 1993 Winter Simulation Conference,* ed. G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles, 1068-1072, Los Angeles, CA, December 12-15.

Weatherly, R.M., A.L. Wilson, B.S. Canova, E.H. Page, A.A. Zabek, and M.C. Fischer. 1996. Advanced Distributed Simulation Through the Aggregate Level Simulation Protocol. In *Proceedings of the 29th Hawaii International Conference on Systems Sciences,* 1:407-415, Wailea, HI, 3-6 January.

AUTHOR BIOGRAPHIES

ERNEST H. PAGE is a Lead Scientist in modeling and simulation at The MITRE Corporation where he is currently working on the Aggregate Level Simulation Protocol (ALSP) program and is the Principal Investigator for a MITRE Sponsored Research project on web-based simulation. He received the Ph.D., M.S. and B.S. degrees in Computer Science from the Virginia Polytechnic Institute and State University in 1994, 1990 and 1988 respectively. He is active within the U.S. DoD modeling and simulation community, serving on the Defense Modeling and Simulation Office (DMSO) Verification, Validation and Accreditation Technology Working Group (1996 -) and on the Testing (1995, 1996), Interface Specification (1995) and Time Management (1996) Working Groups within the DMSO High Level Architecture (HLA) initiative. He has also served on the Planning and Review Panel for the Simulation Interoperability Workshops (1997). Dr. Page has served as the Secretary/Treasurer (1995-1997) for the Association for Computing Machinery (ACM) Special Interest Group on Simulation (SIGSIM) and cur-

rently holds the position of Vice Chair. He is on the Program Committees for the 1998 SCS International Conference on Web-Based Modeling and Simulation and the 1998 SPIE AeroSense Symposium Technical Conference on Web-Based Simulation. His research interests include discrete event simulation, parallel and distributed systems, and software engineering. He is a member of ACM, SIGSIM, IEEE CS and Upsilon Pi Epsilon.

ROBERT L. MOOSE, JR. is a Principal Engineer in communications and networking at The MITRE Corporation, where his primary focus is internetwork protocol engineering with recent emphasis on nomadic networking and quality of service. His other research interests include network performance analysis and discrete event simulation. He has held positions on the faculty of Clemson University and West Virginia University, and has worked for Bell-Northern Research. He received his M.S. and Ph.D. degrees in Computer Science from the Virginia Polytechnic Institute and State Universiry in 1983 and 1988, respectively.

SEAN P. GRIFFIN is a Senior Staff member in modeling and simulation at The MITRE Corporation where he is currently supporting the Aggregate Level Simulation Protocol (ALSP) project and the MITRE Sponsored Research project on web-based simulation. He received the B.S. degree in Applied Mathematics from the University of Rochester in 1991 and the M.S. degree in Software Systems Engineering from George Mason University in 1997.