SimServer: Simulated Data Streams on Demand via the Web

Douglas Flournoy
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
781-271-2774
rflourno@mitre.org

Robert Mikula
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102
703-983-7168
rmikula@mitre.org

David Seidel
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102
703-983-6828
dseidel@mitre.org

Dr. Richard Weatherly
The MITRE Corporation
7525 Colshire Drive
Mclean, VA 22102
703-983-7203
weather@mitre.org

Keywords:
WWW-Based Simulation, Experimentation Support

ABSTRACT: *Simulated data streams have long been employed to support prototyping and experimentation. These data streams create the operational context within which systems and concepts are demonstrated, tested, integrated, and exercised. Although this context is essential for success, resources are better spent on the focus of the project—not on the simulation support. But that's rarely how it works—applying traditional simulation is expensive. It takes detailed planning, scenario generation, and interface development to provide the simulation capability. Then simulation computers, networks, and knowledgeable operators must be coordinated to execute the simulations. Often, similar work has already been done elsewhere, but there is no clear path to finding and leveraging related work. In this paper we will describe SimServer: an initiative established at MITRE in 2004 to address this situation by offering a means to quickly and cost-effectively meet basic simulation support needs across the company's work programs. By employing a select set of web-inspired computing techniques, SimServer is providing on-demand access to simulated data streams. This means that projects don't need to buy their own simulation support computers, manage the additional network connections, or hire simulation operators. At the SimServer web site, consumers plan, configure, execute, and monitor their data streams. Rather than developing capabilities from scratch, projects use the site to browse available simulation services and reuse or modify them. This common repository of tailorable, on-demand simulation services frees more project dollars to be devoted to prototyping and experimentation activities, facilitating broader and deeper experimentation programs that deliver richer insights for shaping the future of fielded systems.*

1. Introduction

SimServer is:

- A Web site that provides a corporate service for planning, configuring, executing, and monitoring simulated data streams.

As SimServer evolves in actual practice, we are finding that SimServer is also:

- A library of data streaming tools that grows with its user base.
- A virtual meeting place where projects can share data streaming concepts and solutions
- A case study in the practical application of "composable simulation frameworks" and "web-based simulation."

In FY04 we received MITRE internal technology initiative funding to begin working on the simulation resource Web site that is now SimServer. In this paper we begin by discussing the motivation behind MITRE's SimServer initiative. We then discuss the vision for the project, and present a scope of activities for which we believe SimServer to be suitable. Then we provide a technical overview of the current SimServer web site from two perspectives: how a user interacts with SimServer, and the underlying technologies and software structure that make this possible. We conclude with a discussion of our progress on SimServer to date and our proposed path forward.

2. Motivation behind SimServer

Simulated data streams have long been employed across MITRE's work programs to support prototyping and experimentation. These data streams create the operational context within which systems and concepts are demonstrated, tested, integrated, and exercised.

Though the simulated context is essential for success, resources are better spent on the focus of the project. But that's rarely how it works—applying traditional simulation is expensive. It takes detailed planning, scenario generation, and interface development to provide the simulation capability. Then simulation computers, networks, and knowledgeable operators must be coordinated to execute the simulations.

In short, there is a company-wide shortfall in our ability to quickly and cost-effectively meet the basic simulation support needs of our projects. We saw an opportunity to address this shortfall by pooling simulation knowledge, scenarios, and tools in a common network location and packaging these resources so that project engineers could create and manage their own simulated data streams.

3. SimServer Vision

By employing a select set of web-inspired computing techniques, SimServer provides on-demand access to simulated data streams. This means that projects don't need to buy their own simulation support computers, manage the additional network connections, or hire simulation operators. At the SimServer web site, consumers plan, configure, execute, and monitor their data streams. Rather than developing capabilities from scratch, projects can use the site to browse the available simulation services and reuse or modify them.

The SimServer software features a component-based design that allows different scenario offerings to be combined with a variety of transformers, like interpolators, geographic filters, and formatters. The resulting data stream "configuration" can be emitted using several different network protocols to any number of receiving applications. This online approach to generating data streams allows projects access to the data streams they need at a fraction of the cost and time of previous methods.

With the advent of Service-oriented Architectures (SoAs) and the push toward net-centric enterprise services, a new class of simulation consumer is emerging that requires data that mimics the XML schemas and machine-to-machine (M2M) Web Service mechanisms of net-centric operations. By incorporating lessons learned from the Java- and XML-based web publishing community, SimServer is well positioned to meet the needs of these emerging enterprise engineering projects.

4. SimServer Concept of Operations

SimServer is not intended as an alternative to the today's complex simulations. Rather, it is intended as a vehicle for providing certain data stream products from these simulations in a more accessible and tailorable, yet less complicated fashion. Given this context, the following questions are reasonable to ask:

- What might the SimServer data feed represent?
- For what purpose might SimServer be used?

4.1 Data Feed Representation

The data that flows through SimServer can be any type of time-tagged information. The list below provides a representative set of examples of what is possible:

- **Simulation ground truth playback**. A simulation produces location, movement, and status data during its execution, saving the data to a file periodically. SimServer reads the file, transforms it appropriately, and emits the data.
- **Real world event playback**. An exercise takes place in a monitored environment (a live training facility, for example). The monitoring equipment saves information about the participants to a file periodically. SimServer acts on that file.
- **Sensor simulation output**. During an exercise, a sensor simulation reports on its detections and tracks. That information is captured to a file during the exercise. SimServer uses that file for its execution.
- **Real world sensor output**. During an exercise, a real world sensor reports on its detections and tracks. That information is captured to a file. SimServer uses that file for its execution.
- **Message flow**. C4I messages are exchanged by participants in an exercise. The messages are captured to a file and played back by SimServer.

4.2 SimServer Use Cases

As might be expected, all uses of SimServer entail systems that receive events from external sources over time. These uses can be divided into the following categories: Testing, Demonstration and Experimentation, and Training.

4.2.1 Testing

Testing a dynamic system requires test procedures, a time-evolving scenario that supports the procedures, and a

means to the extract results of the test. If, as expected, some of the tests fail, the procedures must be executed again with the same scenario when the system is fixed. The test scenario should be easy to construct, repeatable, and easy to execute.

Regression Testing is the testing that must be performed during system development and maintenance to make sure that changes and enhancements haven't broken previously correct functionality. If a portion of system input is time-tagged data input, data streams are required input to regression testing. The streams must be configurable, reliable, and repeatable. The normal operation of SimServer provides all of the characteristics needed by regression testing. Once the user establishes the sources, transformers, and emitters needed for a test, they are saved; only that use may modify, delete, or execute them.

Load tests typically begin with a light load on a system and gradually increase it to determine the point at which the system's performance or behavior becomes unacceptable. Ideally, these loads are easy-to-construct, easy-to-reproduce, and easy-to-document. When the load consists of events arriving at the system, SimServer meets all of these criteria; as an example, it could be used by constructing a simple data file or use an existing simple source, then on successive runs, increasing the frequency in the configuration to produce more events per second. Alternatively, multiple configurations could be executed simultaneously with the same source but offset in location, again producing more events per second.

Integration tests are designed to bring together components that are intended to work in a common system. The components are often developed in isolation with little opportunity to test with the rest of the system. As a result, integration events spend a lot of time dealing with interface and minor coordination problems of individual components, reducing the amount of time available for true integration testing.

One solution to this situation is to provide components with tools to perform basic interface testing in their development facility, prior to integration testing. SimServer provides a mechanism for all components to deal with the events in their own facility. One way it could be used is for the integrating authority to prepare "authoritative" scenarios for the components to test against. Passing this test could act as a ticket of admission to the integration event.

4.2.2 Demonstration and Experimentation

A **standalone demonstration** of a C4I system, user interface, or Common Operating Picture (COP) needs to be stimulated by dynamic data that shows the benefits of the system being demonstrated. That drives the need for an easy-to-construct, reliable, consistent scenario that always does what it's supposed to. Since demonstrations are often interrupted by questions, control over the data stream is important (pause, resume, repeat).

A **synchronized demonstration** is similar to a standalone demonstration except that several (or lots of) systems are participating, all of which should receive and act on the same data stream. Its data requirements are similar to those of standalone demonstrations with the added complication of multiple receivers. Given the increased likelihood of failure of one of the systems, it must also be possible to restart the data stream very quickly after restart.

Background Activity is what's going on in the background while the star of the show is performing. For example, if a C4I system is designed to identify and designate a portable missile launcher on the ground, there should be lots of cars and trucks moving around, providing a realistic background for the missile launcher and challenges for the C4I system. Since the demands for precise detail aren't present, a simple, repeating data source run through SimServer satisfies this need.

Experiments, at least those intended to evaluate human responses to stimuli, are similar to demonstrations. They require the execution of a known and repeatable scenario to present to human scenario participants. Each set of experiment subjects should get the same scenario. The set-up time between experiments should be minimized.

4.2.3 Training

Some training systems could use SimServer as their dynamic input. For example, consider a missile defense system. The dynamic elements are the target missiles which, after launch, fly in a well-behaved manner. So, the variability in the training environment results from (1) changing the launch location and time, and (2) which threat missiles are destroyed and when. SimServer could serve in this environment: all possible missile trajectories are constructed as separate configurations; the training staff executes configurations when appropriate for the training situation; and the staff deletes a configuration when they determine that the threat has been destroyed.

5. Using SimServer

SimServer operations are based on the premise that data streams can be produced by "configurations" of composable software components (see Figure 1, below). A configuration consists of a single data *source* (shown in yellow), some transformers (blue triangles) to manipulate the data, and one or more *emitters* (red lightning bolts) to send the data to user systems. Transformers and emitters typically require control parameters.

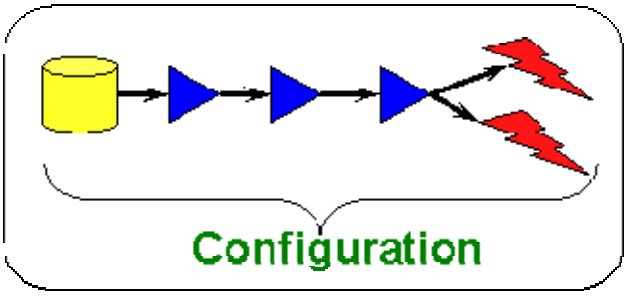

Figure 1. SimServer Data Stream "Configuration" Concept

Built on this configuration concept, the process for using SimServer to feed simulation data to a system involves these steps, discussed in more detail below:

- Plan
- Configure
- Execute
- Monitor
- Map Display (if desired)

5.1 Plan

The first step in using SimServer to provide data feeds to a user system is to understand what is needed for the activity. This includes:

- The format of the data that the system expects
- The frequency with which the system expects the data
- The types of objects that the system will deal with
- The activities of the objects

Armed with that knowledge, a user can access SimServer Web site and view the site documentation to determine the degree to which existing SimServer capabilities (data formats, sources, transformers, and emitters) meet his needs. If all needed capabilities are available, the user can proceed to the next step and begin to configure a data stream. Otherwise, he can contact the SimServer team to create and post any new components needed.

5.2 Configure

The next step is to create a SimServer configuration. At this point, the user must log into the SimServer system using his corporate username and password (password access is required to create, edit, or run SimServer configurations).

Using the *Configure* page of the SimServer Web Site, a user builds his configuration by selecting from available data sources, transformers, and emitters. He chooses the data source, then adds and edits the parameters of the transformers he needs to manipulate the data. Finally, he adds one or more emitters that direct the transformed data over the network to his receiving applications.

In the example illustrated below, a SimServer user creates a configuration representing an attack on Kuwait by sea. For a data source, he finds a SimServer Boston attack scenario that fits his basic needs for vehicle types and routing. By adding SimServer transformers to this configuration, he relocates the scenario to the Kuwait area, specifies that platform locations be updated every 20 seconds, synchronizes the activity to wall clock time and transforms the data to the "Cursor on Target" XML schema format. Finally, a SimServer emitter is added to direct the data stream to the appropriate destination computer.

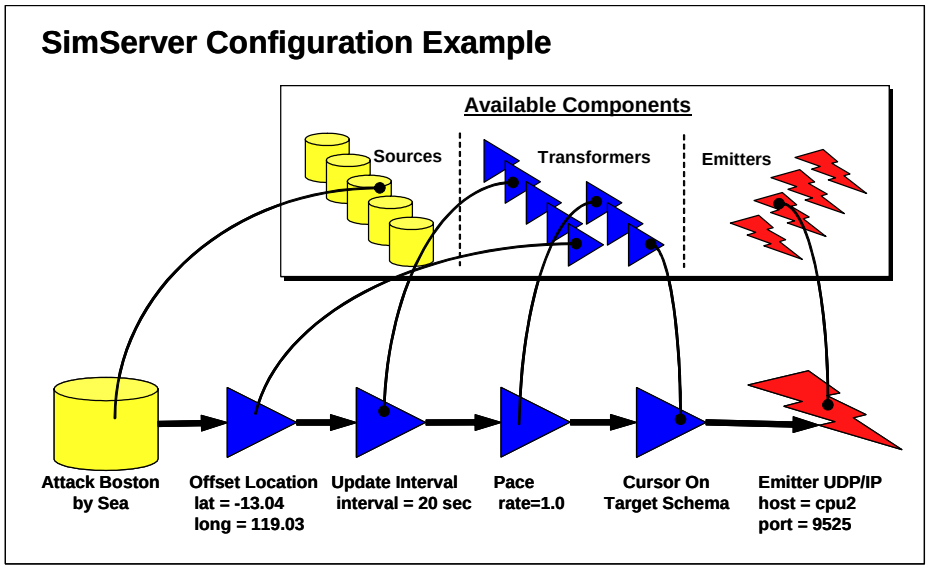

Figure 2. Example configuration: Kuwait Attack

Figure 3 shows this configuration as displayed on the configuration editing page of the SimServer Web site. The source, transformers, and emitter were added to the configuration by selecting them from the pull-down windows across the top of the page. Then, the parameters for each component of the configuration were edited by selecting the component and using the *edit* button to the right. The order of components in a configuration can also be manipulated by using the *Up* and *Down* buttons to the right.

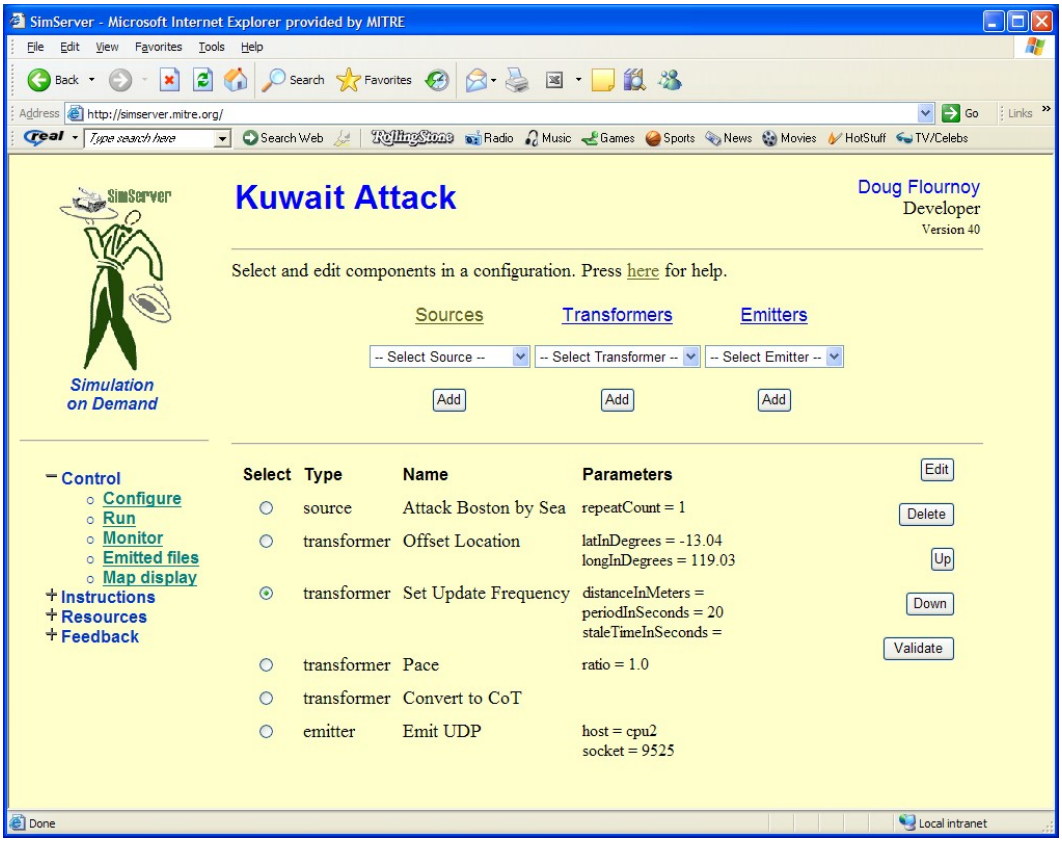

Figure 3. Building the Kuwait Attack Configuration in SimServer

5.3 Execute

From the "Run" page on the SimServer Web site, the user can now select the Kuwait Attack configuration from the list of available configurations, give it an execution name, and trigger its execution. For example, an execution name for the Kuwait Attack configuration might be "Kuwait Wednesday."

5.4 Monitor

After a configuration is executing, its progress can be monitored and controlled from the Monitor screen as shown in Figure 4. Each execution is represented by a row in the monitor table. The monitor screen provides an indication of the status of each execution (*Launched*, *Running*, *Paused*, or *Complete*), start time, and the most recent time of activity for each execution.

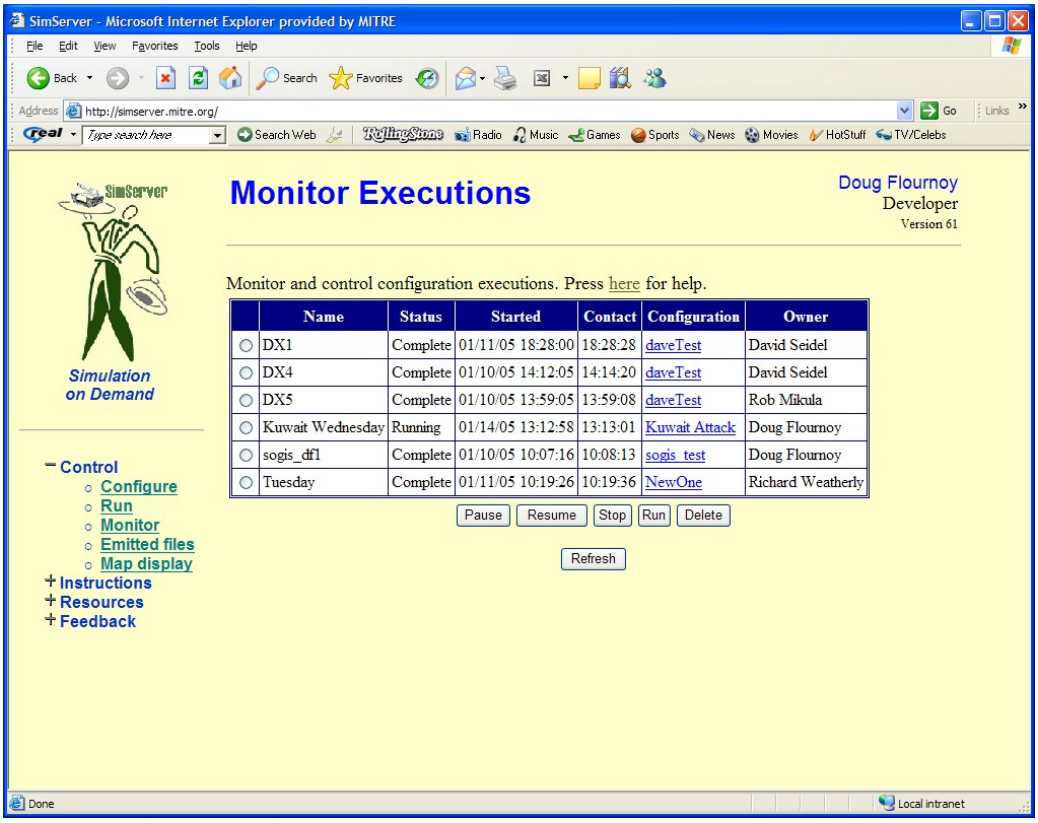

Figure 4. Monitoring the Kuwait Wednesday Execution

The Monitor screen also provides controls over an execution. Executions may be paused, resumed, stopped (terminated), run (started over once complete), or deleted from the list.

5.5 Map Display

Often it is useful for a SimServer user to "see" his configuration execute on a plan view display. For this purpose, SimServer provides a map applet, and instructions for its use, based on the BBN Corporation's OpenMap™ open source mapping software. [1] Figure 5 is an illustration of the results of the Kuwait Wednesday sample execution shown on the OpenMap™ plan view display.

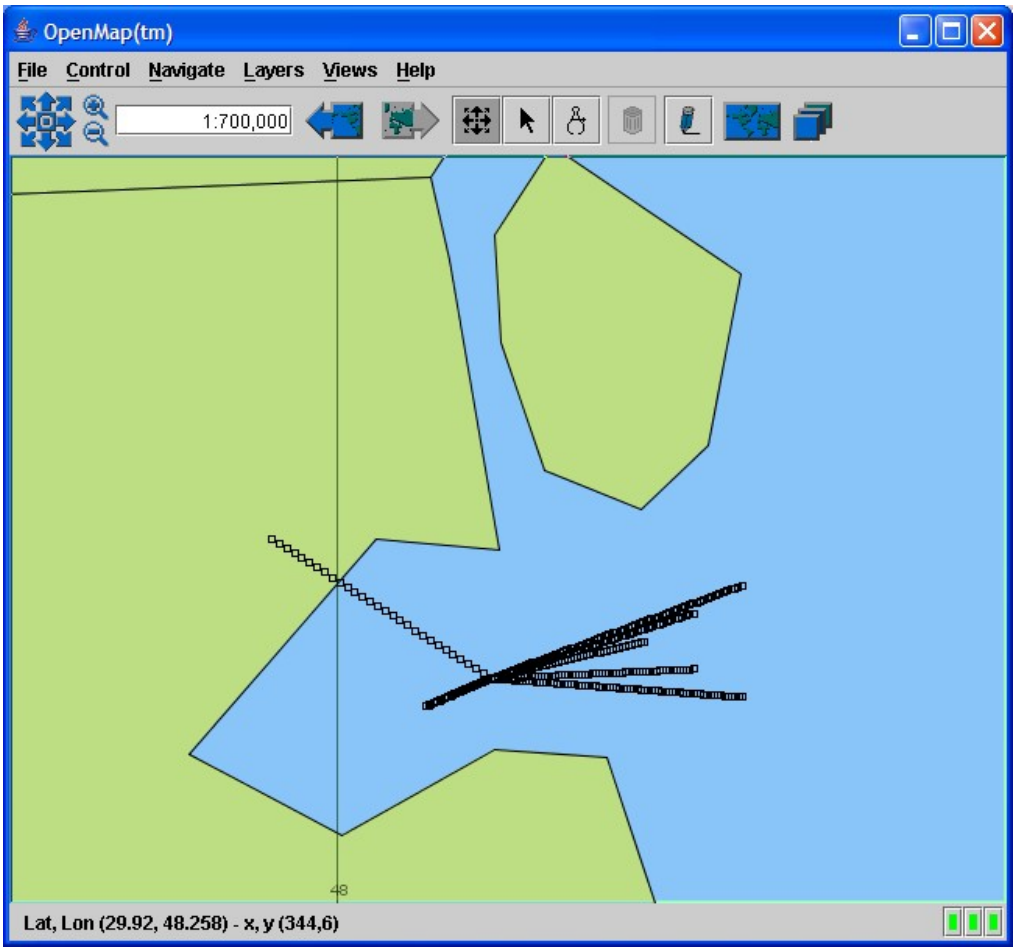

Figure 5. Plan View Display portion of the OpenMap™-based map display

6. How SimServer Works: What's Under the Hood

Careful consideration was given in determining the underlying software structure for SimServer. The software underpinnings had to, among other things:

1. support many dynamic web page functions and
2. deploy easily within a Web Server environment, and
3. provide a framework for software components to be developed and employed in different user-definable combinations.

This led us to consider the Cocoon Project from the Apache Software Foundation [2]. In fact, our vision of a SimServer configuration exhibits a remarkable similarity to the component-based software execution offered by the Apache Cocoon Project.

Apache Cocoon is often described as an XML publishing framework, in which the flow of execution is described as a pipeline, consisting of a generator step, one or more transformers, and concluding with an emitter. This is open source software freely available at http://apache.cocoon.org.

As delivered, Cocoon provides a number of examples. However, none of these were specific to simulation. However the use of this open source infrastructure intrigued us. We believed that the later versions of the Cocoon framework could be extended to incorporate components of specific benefit to simulation.

The latest versions of Cocoon use a SAX-based processing model for XML data streams. This provides the advantage of event-based processing, allowing each element of a data stream to pass from one stage to the next, rather than waiting for the entire data set to complete one stage before moving on to the next.

With such a model, we were then able to write a series of Java programs to begin processing source data, trigger SAX-based transformations in a desired sequence, and finally to emit the data to desired destination(s).

This framework for mixing and matching Java-based software components turned out to be remarkably flexible and robust, as can be seen from the following partial list of supported SimServer components.

Example Sources can be:

- Static documents in XML or flat files
- RDBMS queries
- Queries from Native XML Databases
- Results of Web Service Calls

Transformers can fulfill the following functions:

- Modify temporal aspects of the data stream
- Modify XML tags or their attributes
- Filter or modify data
- Change languages or target schemas

Emitters can be:

- TCP sockets at any IP address
- UDP sockets at any IP address
- Log files
- Browser displays
- Alternative displays such as PDF, JPEG, etc.

Use of the Cocoon infrastructure allows each of the Java components written for SimServer to be quite compact, usually less than a page of Java code, almost always less than two pages.

The ability to combine these features on demand leads to a robust suite of potential configurations, which are deployed as a Tomcat web application.

We then needed a way to allow users to view all the available components, and to build, store, and execute their own configurations. By defining these in XML, we are able to reuse much of the XML processing logic within Cocoon, including linking Cocoon to an open-source native XML database, known as eXist (http://exist.sourceforge.net), to store SimServer resources. The XML Server Pages, known as XSPs, used to display the data incorporate a security and management structure governing the execution of configurations.

Using Java programs, we were then able to initiate configuration execution without a browser. With Apache Axis, we were then able to offer a web service to initiate execution.

7. Status and Observations

SimServer is supporting a first round of MITRE project consumers in 2005. Some of these first users of SimServer are enterprise services-related projects that can take advantage of SimServer's emphasis on Web technologies. But we are also seeing immediate benefits to small projects like MITRE internal research efforts that require relatively straightforward, repetitive data feed support within tight budget constraints.

As more projects turn to SimServer for support, the envisioned pattern of growth is emerging. That is, we are finding that many of the features new users need are already available in SimServer. However, each new project brings a few new requirements that we meet by adding a new source or transformer, or extending an existing component. In this manner, the set of capabilities that SimServer offers to new users is indeed growing more comprehensive over time.

As we evolve SimServer capabilities to meet each new requirement, the Cocoon-based component software structure is proving to be a solid, straightforward base for each modification. Because much of the software "engine" behind SimServer is powered by the XML-based capabilities of Cocoon and eXist, the SimServer-specific software we need to develop and maintain is very lightweight. In fact most of the configuration SimServer configuration components are less than 100 lines of code (on the order of 1-2 printed pages). We have found Cocoon and the rest of the SimServer software to be a ready framework in which users can compose and reconfigure their own simulation streams without further software development.

"Web-based simulation" is another ongoing simulation community thrust area for which SimServer can contribute lessons learned. SimServer's mission pushed us to focus on the benefits that web technologies can provide the end users of simulation products—for instance, 24/7 access to simulation support and the power of XML for transforming data. Indeed, these features of the Web can be harnessed and provided to simulation users with positive results.

Feedback from early SimServer users indicates that we have a solid structure on which to continue to build. The availability of this common repository of tailorable, on-demand services will free more project dollars to be devoted directly to prototyping and experimentation activities, facilitating broader and deeper experimentation programs and resulting in richer insights for MITRE's sponsors.

8. References

[1] BBNT Solutions LLC. *OpenMap Architecture*, http://openmap.bbn.com/doc/openmap-arch.html, 2001.

[2] Bill Brodgen, Conrad D'Cruz, Mark Gaither. *Cocoon 2 Programming: Web Publishing with XML and Java*, Sybex Incorporated, 2003.

Author Biographies

DOUGLAS FLOURNOY is a Principal Simulation and Modeling Engineer within the Center for Acquisition and Systems Analysis at the MITRE Corporation in Bedford, Massachusetts. In addition to SimServer, Doug is engineering a Link-16 communications simulation and investigating HLA federation performance prediction methods. He also has experience in human behavior and process modeling, simulation-to-C4I system interfacing, and user interface prototyping. Doug holds a Bachelor of Science Degree in Mechanical Engineering from the Pennsylvania State University and a Master of Science Degree in Operations Research from the George Washington University.

ROBERT MIKULA is a Senior Software Systems Engineer within the Center for Innovative Computing and Informatics of the MITRE Corporation in McLean, VA. He currently supports the Center for Enterprise Modernization with the implementation of Service Oriented Architectures within the federal government. Rob has experience in web-based software development, XML, human computer interfaces, databases, and object request broker technologies. He holds a Project Management Professional Certification and is currently pursuing doctoral studies at George Mason University.

DAVID SEIDEL is a Senior Principal Simulation and Modeling Engineer for the MITRE Corporation. He has participated in the development of several distributed simulation projects including Aggregate Level Simulation Protocol, HLA prototypes and standards, DMSO outreach federations, the Joint Simulation System (JSIMS) and the Joint Distributed Engineering Plant (JDEP).

DR. RICHARD WEATHERLY is a Consulting Engineer with The MITRE Corporation. He received a Ph.D. in Electrical Engineering from Clemson University in 1984. He led the technical development of the Aggregate Level Simulation Protocol, DoD High Level Architecture (HLA) for Modeling and Simulation, and the HLA Runtime Infrastructure Verification Facility. He served three years as Chief Engineer of the JSIMS project. Currently he is Principal Investigator of an effort to improve the performance of process-oriented, Java-based simulations and the Software Architect of the Meteor autonomous land vehicle project. He has published over 25 papers and a text book on distributed simulation.