An Extensible Component-Based Architecture for Web-Based Simulation Using Standards-Based Web Browsers

DAVID S. MYERS

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

June 14, 2004

Blacksburg, Virginia

Keywords and phrases: Animation, client/server simulation system, extensible markup language, visual simulation, web-based simulation, asynchronous communication protocol, component-based simulation, web standards

An Extensible Component-Based Architecture for Web-Based Simulation Using Standards-Based Web Browsers

DAVID S. MYERS

Abstract

Web-based simulation (WBS) systems offer tradeoffs between user interactivity and hardware requirements striking to seek a balance between the differing concerns. Server-based systems offer little interactivity or concurrent visualization capabilities, while client-based systems have increased hardware requirements asking the user to provide high-end workstations. Concurrent visualization of simulation output proves execution intensive, or unusable in some situations. Creating an execution efficient and user friendly WBS system greatly improves user experience while gaining all of the benefits inherent in a web-based system such as high accessibility and ease of maintenance. In order to provide a usable concurrent visualization WBS this thesis developed the *Web-Based Queuing System Simulation System* (WebQS3). WebQS3 splits the responsibilities of simulation execution and simulation visualization into a client-server environment; the client is responsible for the visualization display and server is responsible for simulation execution. The system differs from many previous WBS systems in that the client-side application is developed using web-standard technologies such as HTTP, XML, SVG, and ECMAScript instead relying on Java Applets and associated technologies. Using web-standards as the foundation of the client agent opens the visualization and model construction functionality to any user that accesses the application using a web browser while also making the application more scalable in terms of user load. Implementing the client with web-standards also included the development of an asynchronous client-server communication protocol as opposed to traditional synchronous communication protocols used by Java WBS systems. The asynchronous protocol demonstrates similar or better execution performance than similar synchronous communication protocols in most quality characteristics. By creating a WBS system using web-standards implemented in most modern web browsers any user may visit the WebQS3 site and have simulation tools available for use. Providing simulation services on the web makes eases the creation of simulation models my making the tools to readily available while facilitating information sharing and collaboration over the web. The WebQS3 system serves as a model to drive research in WBS systems away from proprietary Java technologies to web standards for front-end visualization technologies.

Acknowledgments

I would like to thank John Costigan for his assistance in architecting and coding the statistics collection module. I also appreciate the continuous prodding and encouragement offered by all my family and friends, especially my fiancée Jennifer Thomas, who constantly reaffirmed her belief that I "will be done this weekend."

Table of Contents

List of Figures

List of Tables

List of Acronyms

ASV	Adobe SVG Viewer
B2B	Business-to-Business
CGI	Common Gateway Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
EJB	Enterprise JavaBean
HLA	High Level Architecture
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol (Secure)
J2C	J2EE Connector
J2EE	Java 2 Enterprise Edition
JMS	Java Messaging Service
JRE	Java Runtime Environment
M&S	Modeling and Simulation
MDB	Message Driven Bean
MVC	Model View Controller
PADS	Parallel and Distributed Simulation
PDA	Personal Digital Assistant
RDBMS	Relational Database Management System
RPC	Remote Procedure Call
SIM-ASP	Simulation Application Service Providing
SBD	Scenario-Based Design
SOA	Services Oriented Architecture
SV	Simulation Visualization
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
VRML	Virtual Reality Modeling Language
W3C	World Wide Web Consortium
WAN	Wide Area Network
WAS	WebSphere Application Server
WBS	Web-Based Simulation
WebQS3	Web-based Queuing System Simulation System
WYSIWYG	What-You-See-Is-What-You-Get
XBL	eXtensible Binding Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XML-NS	XML Namespace
XML-RPC	XML Remote Procedure Call
XSLT	Extensible Stylesheet Language Transformation
XSV	XML Schema Validator
XSD	XML Schema Description Language

Chapter 1: Introduction

Many companies now provide web-based and server-based versions of major software products. Placing software products on the web allows companies to reach a new group of customers and provide continuous access to the most recent version of the software systems to current customers. Companies such as Microsoft, IBM, and Sun Microsystems regularly release new toolkits, specifications, and development environments which in turn make the creation of new web-based programs easier; these programs are based on web standards and services that take full advantage of the power and availability of the web environment. Web-based programs can be placed on a multitude of distributed computers allowing any number of concurrent users to access to the program's functionality at any time.

The software development industry currently faces a shift in program availability and usage, from creating software built to run on a localized single computer by a single user to creating a scalable software system for a multitude of concurrent users in a distributed execution environment. Users are becoming more demanding of application availability, reliability, and usability: they expect to be able to connect to the web-based software system at any time, from any web-enabled location, using any device to access the system. Furthermore, they are waiting for many existing applications to be ported to the web and meet those expectations.

Many Modeling and Simulation (M&S) tools exist to provide simulations in a web environment [Page 1999], but few provide an end-to-end system providing model development, model verification, and simulation execution facilities. Those web-based M&S environments that do provide an end-to-end system usually require specialized programming knowledge in order to develop the simulations. For example, Simjava requires a working knowledge of the Java programming language and the Simjava libraries. In addition, most of the toolkits require installation of special software, such as the Java Runtime Environment, in order to execute on the user's machine. The research system presented in this paper seeks to provide an end-to-end M&S environment for web-based simulation (WBS). The system demonstrates that the web environment has matured to a point where developing a pure web-based simulation system utilizing web standard technology is possible. The system is designed as a research prototype to provide insight into further development of a commercial-grade WBS system.

1.1 Web-Based Simulation Literature Review

Researchers began investigating moving M&S environments to the web almost as soon as the web became a mainstream technology. Lorenz, Dorwarth, and Ritter [1997] present a basic approach toward migrating a simulation engine to the web using old style common gateway interface (CGI) technology and Perl scripts, providing insight into the benefits of WBS. Lorenz and Ritter [1997] also present a web-based animation system named Skopeo, for displaying simulation output using Java animation techniques. Combining the CGI scripts with Skopeo created a web-based simulation environment capable of displaying animations of pre-prepared simulations to users.

Web technologies and the range of output from simulation environments have advanced concurrently. Huang, Fanh-Tsou, and Chang [1998] present front-end enhancements with a multi-user animation display environment using the Virtual Reality Modeling Language (VRML). The display tool creates a VRML environment displaying a 3D animation while allowing user interaction with the simulation display. The tool also provides other collaboration tools such as chat windows to foster simulation user interaction. Straβburger, et al. [1998] provide updates to the backend simulation engine using a High Level Architecture (HLA) based simulation engine. This allows distributed simulation with considerations for posting simulation output to a web front-end.

Several researchers created specialized simulation environments utilizing the web as a transport mechanism. De Lara and Alfonseca [2001] created a simulation language called the Object-Oriented Continuous System Modeling Program (OOSCMP). OOSCMP generate Java Applets showing running simulations in hopes that professors would use the language to enhance lecture presentations to students. They later present an application of the tool [Alfonseca, de Lara, and Vangheluwe 2001] that illustrates a visual interpretation of how to solve a series of partial differential equations. Gan et al. [2001] provide a simulation environment for parallel simulation in the domain of supply chain management. This allows enhanced execution on the backend via parallel and distributed simulation (PADS) while visualizing the output using Java Applets using asynchronous server communication.

Simulation languages as well as modeling environments are targeting the web. Healy and Kilgore [1997] created the Silk programming language to foster web-based simulation adoption. Simjava [Howell and McNab 1998, 2003] is a package of Java classes used to generate both back-end and front-end simulations in Java. JSIM [Miller 2003] is yet another Java package targeting WBS developers. Klein, Straβburger, and Beikirch [1998] detail an extension to the old GPSS simulation language named JavaGPSS and providing facilities for web-based distribution. Kilgore [2001a, 2001b] presents the use of open source technology in WBS through the use of OpenSML, an open source simulation language. The development of the simulation languages targeting a web environment helps to foster the development of web enabled simulations.

Research time has also been devoted to providing simulation services on the web, allowing businesses as well as users to access simulation information and functionality. Recently, Wiedemann presents application service providing under the new paradigm of application service providing for simulation purposes [Wiedemann 2001]. Kilgore [2002] presents an overview of web services, discusses the use of web services in the context of simulation, and provides a demonstration of the use of web services for simulation as implemented under the Microsoft .Net framework [Microsoft 2004]. Chandrasekaran et al. [2002] examine the synergy between web service technologies and simulation. Fishwick [2002] emphasizes the importance and benefits of using eXtensible Markup Language (XML) for simulation modeling, a requirement in the field of providing services in a cross-platform nature on the web. Kim, Lee, and Fishwick [2002] expand on the use of XML through the creation of a modeling process for WBS based on an XML model.

There are several encompassing references on WBS. Page [1999] provides a survey of WBS. He supplies lists of simulation packages supporting WBS, as well as online examples of simulations running in a web environment. Page et al. [2000] discuss the viability of simulation on the web and the inherent challenges with the simulation on the platform. More recently Kuljis and Paul [2003] present an overview of the current state of discrete event simulation on the web, and possible future directions in research.

1.2 Software Paradigm Change

During the last five years, a paradigm shift in the software development is apparent. Traditionally, software is developed as a shrink-wrapped product, sold and purchased as a product, and installed on computers as a product. Under the new paradigm, software is treated as a service that runs on a server computer, permitting the users to access the service over the Internet. This paradigm shift has many advantages for the vendors and users [Tao 2001].

Two platforms have emerged recently for developing software as a service: Java 2 Platform, Enterprise Edition (J2EE) industry-standard architecture [Sun 2004a] and the Microsoft .Net Framework [Microsoft 2004a]. We used the J2EE architecture for developing the Web-based Queuing System Simulation System (WebQS3), which can be currently accessed at http://sunfish.cs.vt.edu/webQS3/.

1.3 Statement of the Problem

Most commercial simulation software products are available as stand-alone products, running on a single user's PC. Running simulations on standalone PCs generates a wealth of issues. Running simulations on a single PC requires that the simulation engine be implemented in an operating system and platform specific manner for the PC, limiting the users to specific platform or hardware configurations. The reuse of simulation models is reduced as the simulation model is locked to a specific platform at model creation. More execution intensive simulation applications run slowly on older systems, requiring hardware upgrades or specialized hardware for specific simulations. Maintaining different versions of simulation models on a multitude of PCs becomes a hassle as multiple users make changes to the same model that must be merged and redistributed to all users of the model. By providing a web-based simulation environment many of these issues are resolved while providing many other benefits to users.

1.4 Statement of Objectives

The objective of the research described herein is the development of an extensible end-to-end client-server concurrent visualization web-based simulation environment implemented using web-standard technologies including SVG and ECMAScript available on modern web browsers. The environment should allow component-based model composition using a what-you-see-is-what-you-get direct manipulation paradigm promoting ease of model development. The use of web-standards allows the simulation model to be represented using an XML formatted document description facilitating reuse of readily available tools. The client portion of the system should communicate with the server using standard asynchronous communication libraries and protocols available to any web user. The system should allow the composed simulation model to be executed and the resulting visualization to be rendered on the client concurrently with the execution of the simulation on the server.

1.5 Overview of Thesis

Providing an application based on the distributed web-environment allows for several benefits to both the customer and application developer. Chapter 2 provides an overview of the benefits and drawbacks of providing simulation on the web. To maximize the benefits of web-technology, web-based systems should utilize many of the web standards available from the World Wide Web Consortium (W3C). Chapter 3 describes a standards-based approach to providing a simulation visualization engine and simulator on the web. Chapter 4 discusses the development of an asynchronous visualization protocol employing standard web communication methods. Chapter 5 provides a design of a web-based simulation (WBS) system following the usability engineering approach to system development. The ideas developed in chapters 3, 4, and 5 are demonstrated through the development of a web-based simulation system, described in detail in chapter 6. Chapter 7 describes the verification and validation activities. Finally, conclusions are presented along with suggestions for future research.

Chapter 2: Web-Based Simulation Benefits and Drawbacks

Benefits and drawbacks exist for every implementation of a software system. Web-based systems are no different, but web-based simulation can provide benefits to users and application developers whose simulations are implemented in current non-web simulation software products such as Arena [Arena 2003] and AweSim [Pritsker 2003]. Benefits of porting systems to a web-based environment include more efficient installation and maintenance, decreased overall cost, increased application availability, controlled access, increased integration into current environments, and access to more powerful systems than typical user PCs. Drawbacks include a limited user-interface, a dependence on network/application availability and volume, and reduced simulation execution efficiency. The drawbacks of web-based simulation expose risks that must be considered before implementing a web-based software solution in addition to the difficulties in porting traditional simulation technologies directly to a web environment. The following sections provide an outline of both the benefits and drawbacks of web-based software as described by Tao [2001], with special considerations made for web-based simulation derived from the report Netsim: A Java-based WWW Simulation Package [Veith 1997].

2.1 Web-Based Simulation Benefits

Several benefits exist for web-based systems and, in particular, web-based simulation. The benefits can be grouped into general categories and further divided into a series of more specific benefits:

Increased Application Accessibility

- Allows access on many platforms without recompiling program or models.
- Allows access at distant sites without transporting hardware of software.
- Allows access outside normal business hours.

Efficient Installation and Maintenance

- Requires virtually no user client installation
- Enables frequent modifications to be made and instantly distributed.
- Reduces error potential when copying and distributing models.
- Eliminates virtually all "in-site" maintenance.
- Allows modifications and implementations to be made through the server.

Decreased Cost

- Reduces installation costs
- Reduces multi-platform testing costs
- Reduces maintenance and distribution costs

Controlled Access

- Protects against inadvertent and unauthorized change and duplication of original.
- Allows "copy exactly" model distribution.
- Enables individualized access through passwords and user identification.
- Enables limited time-span access.

Increased Integration

- Interfaces easily with existing web browsers.
- Requires only a web browser.

- Encourages interaction and communication through the web.

Access to More Powerful Systems

- Allows access to server systems that can run the simulation more quickly than PCs.
- Allows models to take advantage of distributed and parallel simulation techniques to increase simulation efficiency.

2.1.1 Increased Application Accessibility

Platform specific models or compilation are often required in traditional, non-web-based, simulation software. Running a simulation through a web-interface removes the need for machine specific software and hardware installations, as well as machine specific model code. If a simulation requires a special machine environment for execution, the requisite hardware can be installed on the single server configuration without needing to change the interface exposed to the user. The web is available on any platform that sees fit to write a web browser, which allows simulation users to quickly and easily access and run the model from multiple, wide-spread locations. Users also have access to the most up-to-date models and information regardless of where they are on the globe. A user could access a simulation from the UNIX workstation at their office during the day, check the status of a long running simulation from a mobile device while at lunch, or add a few ideas to a model in development from a Windows workstation at their home that same night with ease. The presence of a central data/model repository on the server eliminates the need to transport hardware, software, and data between sites in order to run the simulation. Since all the code is executed on a single server, users do not need to recompile the model code to run on different platforms. Most importantly, the Internet is available 24 hours a day. Access to a model or simulation results is not constrained by time. Users can work at their own pace instead of a pace dictated by office hours. People across the globe can instantly access the needed simulation materials, even if they are hours ahead of the time seen at the office.

2.1.2 Efficient Installation and Maintenance

Installation of web-based systems is easier on the user side; all that is needed is a web-browser that usually comes pre-installed with the system. In some cases, a newer version of the browser must be installed. However, that is a concern for the browser developer, not the simulation environment install. The installation of the main environment is on a single server or cluster of machines. By localizing the installation to a single location, the install time and required manpower is reduced.

Additionally, eliminating the need to redistribute the system to client installations can ease maintenance on the system and the individual models. Since most of the environment logic is held on the server, the clients automatically see the changes made to the system and the simulation models each time they connect to the server. The need to inform users of a patch to download and install, or a new model component to include in the simulation, is removed since the environment handles the revision when they login to the server.

Moreover, the centralization of the server virtually eliminates the need for “on-site” maintenance. Nearly all system or model modifications can be made on the server, thereby greatly reducing the amount of work needed on the individual user workstations. The existence of a single working model enables modifications to be localized to a single set of source files. The web-based approach is a much simpler alternative to traditional stand-alone simulation software products where installation and maintenance can be tedious and time consuming.

2.1.3 Decreased Cost

Reduction of cost is a major benefit of moving applications to a web-based environment. Cost savings are present in every area of development, from installation to maintenance to training to (platform) testing. The installation and distribution cost of the system is diminished since the installation is only performed on a single machine and platform. Since the platform and machine of the server installation is known beforehand, this eases the load on the system and model testers. The maintenance costs are also reduced because the need to distribute the modifications to all the users of the system is eliminated; the changes are localized to a single installation. Training costs can be lowered because users need only be trained to use the environment on a single platform. Also, design costs may be lowered due to the elimination of the requirement for multiple platforms and hardware configurations.

2.1.4 Controlled Access

Modifying a model in both traditional and web-based simulation environments requires access to the model's source code. With traditional simulation packages, users typically access the model directly from the computer on which the simulation is running. When users permanently alter a model residing on their machine significant potential arises for confusion and later difficulties caused by inadvertent changes to multiple copies of a single model. Users are also allowed to change, duplicate, and share models in ways that might violate the license with the model supplier. Multiple models containing various changes can circulate within an organization without the users' knowledge, causing difficulties when users need to collaborate on model results.

Web-based models allow for a more controlled approach to model distribution. When models are accessed and stored in a central location, users can submit model fixes and updates that allow everyone using the model to work off the same standard. This is not to say that there will only be a single version of a model. A model can always be copied or versioned to allow individual users or groups of users to change a model as they see fit without affecting the rest of the organization. Model sharing and maintenance is enhanced through the collaborative nature of the system; anybody is allowed to make a change and may submit it to the shared model. In addition, the access to the underlying model code can be controlled through security protocols; only those people allowed to view/alter the model source will be permitted access. Users not cleared for access to the model source may only run the simulation and view the output.

Security mechanisms offer other features that are attractive in a simulation environment. Login mechanisms through secure password can be used to create a personalized user experience, as if they were using the simulation environment from a customer installation. Also, user sessions can be timed to allow only a specific time span for use of the system if required. These sessions help link the use of the environment to a specific user and not a specific machine or platform as with traditional simulation packages.

2.1.5 Increased Integration

Web browsers and plug-ins that support web-based simulation are readily available from many companies through the Internet. Browsers such as Netscape Navigator, Microsoft Internet Explorer, or Apple Safari are already distributed throughout the web and come pre-installed on most new computer systems. Since many users are familiar with using web browsers and navigating the web, the learning curve for using and navigating the software is minimal. Once the browser is installed, plug-ins optional, no additional software or downtime is required for using the simulation environment. Unlike traditional simulation software products, effective and timely use of a model requires neither the need for proper installation of the software package and models nor a working knowledge of the operating commands. Additionally, the web browser can be made friendly to model user, enabling the creation of simple models for use by non-technical users. Likewise, the browser minimizes the need for assistance by skilled programmers.

The web can help to increase collaboration between simulation users. Issues that arise during active use of the system or a specific model can be raised and relayed to the appropriate people through the use of e-mail. Web-based help forms can submit bug reports to the needed developers. Collaboration between model users can also be increased through the use of online forums or chat rooms that are bundled with the simulation environment. Users can talk with other users worldwide about issues that they have with model design, testing, or implementation decisions. System and model documentation can also be updated quickly without the need to print and send out new copies to all the system users.

2.1.6 Access to More Powerful Systems

If a simulation runs in four hours on a normal computer, it can possibly run in two hours if the simulation runs using two computers simultaneously. The use of a web-based front end to a simulation environment allows the server to take advantage of parallel and distributed simulation (PADS) techniques to improve simulation efficiency. Multiple servers can be used to allow a simulation to run in real time where a single server would not be powerful enough. The server can also be a more powerful machine than the normal users' workstation. A more powerful machine would allow even better performance using PADS techniques. The possibility of execution efficiency improvement is almost limitless, far surpassing execution efficiency speedups offered by traditional simulation software restricted to a single machine.

2.2 Web-Based Simulation Drawbacks

While several benefits to web-based simulation exist, the drawbacks must also be considered. The drawbacks represent risks that must be weighed before web-based simulation is used instead of traditional simulation software products:

2.2.1 Limited User-Interface

The user interface available on strictly web-based interfaces is extremely limited using the default set of available widgets. HyperText Markup Language (HTML) interfaces can include buttons, images, and text box inputs using the set of provided tags, but not complicated data inputs or simulation visualization components. Developers may create more complicated inputs through the use of large amounts of dynamic HTML and scripting or via plug-ins enhancements. The plug-in and scripting enhancements can improve the interface by allowing custom user-interface components that allow for a higher level of control, but the components add more complexity to the development and installation process and create a new piece of functionality that must be maintained. Also, the new widgets may create browser lock-in by utilizing features only available on particular browsers and denying the use of the widgets to groups using other browser vendors. Using plug-ins shifts some of the simulation visualization logic to the client and moves the simulation environment from a thin-client application to a thick-client application. Thick-client applications generally cost more due to some of the costs of distributing maintenance patches to users' returns.

2.2.2 Dependent on Network Availability and Volume

Using the Internet as a platform creates a dependence on network availability. If the network that connects the users to the application is broken, the access to the simulation models is interrupted. Users are unable to use the application when they cannot connect to it. There is also connection speed to consider. When the limited bandwidth is available between client and server the application may appear slow or unresponsive decreasing the usability of the overall system. The problem of maintenance time must also be considered: no one can gain access to the simulation environment when the simulation execution engine is shut down for maintenance. Execution efficiency may also be impacted due to network or application volume. If several users are logged in and running simulations concurrently, the execution of the simulations may begin to slow as increasing amounts of simulations are run on the server. The problems here can be rectified by offering reliable backups or a series of clustered computers to the main system: backup servers, extra network lines, load balancing routers, etc. Yet extra hardware backups add extra costs to the overall system.

2.2.3 Reduced Simulation Execution Efficiency

The simulation server runs all of the simulations that are executed in the environment. Particular simulations may run in a more efficient manner using hardware and services provided on platforms other than the current server environment. For example, a highly complex visualization simulation may execute very efficiently on an SGI workstation, yet the web-simulation server is an Intel server. Thusly, the execution of the simulation may be not as efficient as possible wasting valuable computer cycles. Since the simulation execution platform is set when the server platform is chosen, the simulation models cannot be specialized for platforms and environments that may provide more efficient execution.

2.3 Migrating Traditional Simulation Technologies to the Web

While simulation developers would like to take advantage of many of the benefits provided by migrating to a web-based system, moving traditional simulation technologies directly to the web is extremely difficult to perform and must be carefully adapted to work within the web environment. Many of the challenges that surface in migrating simulators to the web occur because the characteristics of traditional simulation environments are vastly different from the characteristics of web-based environments. This is illustrated by the comparison of simulation and web application characteristics provided by Wiedemann [2001], shown in Table 1.

Table 1. Web-Based Simulation and Traditional Simulation Characteristics

	Common Web Technologies	**Traditional Simulation Technologies**
Common Standards	Yes (HTML, XML, TCP/IP)	No common standard for model and results
Data Handling	By unique URL	Proprietary
Information Structures	Tree structures and lists	Very complex (model dependent)
Specialist Knowledge Required	No (only clicking and reading)	Yes (from a lot of scientific areas)
Navigation	Easy	Difficult
Ease of Use	Very good	Very difficult
Type of Operations	Read & analyze information	Synthesize models and analyze results

While the table is true in some ways, Wiedemann's analysis assumes that all available simulation technologies need to be migrated to a web-environment. In reality, many of the existing simulation technologies can remain as they are and simply provide a consistent communication protocol for interfacing with web clients. Using a consistent communication protocol and recent standards, a web-based client for a backend simulation system can be easily implemented. For instance, using the extensible markup language (XML) [XML 2004] and web services through XML Remote Procedure Calls (XML-RPC) over SOAP [SOAP 2003] allows standards-based communication between client and server. Also, many new modern technology standards are being developed to aid in bypassing the traditional drawbacks of web-based systems. For example, Scalable Vector Graphics (SVG) [SVG 2004a], an XML specification for interactive vector graphic description, enables the creation of complex user interface designs, a grand improvement over simple HTML interfaces. Integrating SVG with XForms [XForms 2004], a very recent specification for creating and displaying interactive user input applications, offers several attractive additions to web-based user interface development, especially in the realm of simulation composition. While the tools implementing the specifications are still in their infancy, the functionality provided by the tools is enough to demonstrate the overall ideas and provide a framework for plugging in new functionality as the tools mature. The rest of this document provides insights and descriptions of a web-standards-based simulation tool.

Chapter 3: Usability Design

The system design followed Scenario-Based Design (SBD) [Rosson 2002] techniques in an attempt to gain insight into how users interact with the system. SBD involves the creation of user scenarios and personas, providing a real face to the users of the simulation system. Starting with a set of user personas, the design moves through the stages of activity, information, interaction design in an attempt to create a system focused on users' needs and levels of experience. Activity design extracts a set of activities that the system users will typically want to perform. The activities can then be analyzed to determine the system functionality that should be provided. Information design seeks to provide focus pertaining to the information the system provides to the user. The design may include development of visual metaphors, screen information layouts, or color choices, as well as any other aspect of information delivery to the user. Information design is followed by interaction design, which seeks to determine the interaction points the system provides to the user. Interaction design aids in shaping the methods a user will employ in providing information to a system, as well as the system's provided feedback. The design may include determining mechanisms for user input (keyboard, mouse, pen input), displaying action feedback, or creating undo mechanisms. Following the process of SBD aids developers in creating a usable and user-focused system.

This chapter breaks down into seven sections. Firstly, problem scenarios are written, illustrating the need for the WebQS3 system and allowing a root concept, or system purpose, to be extracted. From the scenarios, user personas are developed to gain insight into the users of the system. Next, activity design scenarios and claims are created illustrating the system's required functionality. The fourth and fifth sections discuss model composer and model visualizer functionality in terms of information and interaction design with associated claims. The sixth section details prototype interfaces used to test the functionality of the WebQS3 system and extract information needed for developing the final system interfaces. The last section outlines a usability-testing plan employed to confirm the usability claims made during the design phases.

3.1 Problem Scenarios

The creation of problem scenarios is essential to any system in development. Problem scenarios are an effective way to illustrate the need for the system being developed. Scenarios can help generate ideas about types of users and the activities that the users perform that could be integrated into the system. The creation of the WebQS3 system began with two straightforward problem scenarios demonstrating the needs of different types of users. The first scenario demonstrates problems with the availability of simulation resources to non-technical users. The second scenario expresses the lack of collaboration that typical M&S environments provide.

3.1.1 Problem Scenario 1

Ben Coleman is the manager at the local Burger King. He joined the company immediately after graduating high school, and has risen to management level at the restaurant after several years of faithful, diligent service. Ben awakens one morning, and begins to read email on his new iMac computer. Ben has used computers at work for about five years, but this is the first computer purchase for home use. After a quick scan, he notices a message from his assistant manager, John, a BIT student at the university nearby. Ben opens the message, reading that John has created a simulation he wants Ben to read and review. It seems that many customers have approached John to complain of lengthy waits for service during lunchtime. John has brainstormed some ideas on process improvements that could be put into practice. The simulation is implemented in a program called VSE, so Ben navigates to the VSE website to locate a version for either Macintosh or Linux, the OS he uses at work. VSE only supports the Windows and NEXT operating systems, so Ben sends John a note asking him to bring a demo into work, so they can discuss the simulation then.

When Ben arrives at work, he finds that John has prepared a stack of papers full of tables and graphs. John wanted to provide a live demo of the simulation, but the office's machines are quite antiquated in computer years, and so are not powerful enough to run the simulation. Ben has always had difficulty understanding tables and graphs, so John takes some extra time to ensure that Ben understands how everything fits together. Once Ben gains a comfortable understanding, they delve into the details together. A few minutes later, Ben notes that the simulation contains six cashier lines, when only five cash registers exist in the restaurant. A simple mistake on John's part, but it invalidates all the data collected. Ben asks John to rework the simulation, and present the data again tomorrow.

3.1.2 Problem Scenario 2

Dr. Smith has been working to create simulation component code for hours, but he is no programmer. He recalls the day he volunteered to create a simulation model for a human immune system, and attempts to remember what motivated him to do such a thing. Then he realizes: oh yes, the grant money. Additionally, he is working toward tenure in the Biochemistry department and on other big projects, as this looks good on the vita. Dr. Smith spent weeks designing a simulation model and has reduced the entire system to simple queuing theory. Now, if only the components would cooperate. The simulation environment he is using, Arena, provides the necessary objects, but requires that the components be connected and configured using the Arena programming language. Dr. Smith struggles to understand the syntax and eventually has to enlist the aid of a graduate student from the CS department.

After a few days, the graduate student has completed the simulation code and sends Dr. Smith a letter including the simulation model. During lunch on Friday, Dr. Smith receives the file and launches the model, watching the simulation execute on the screen. A few of the output numbers are different from what he expected, so Dr. Smith calls the CS student to ask about the discrepancies. The grad student at his parents' house for the weekend, and so has no access to a computer that will run the simulation. To accommodate, Dr. Smith schedules a meeting for Tuesday so they can meet and discuss the model in more depth.

3.1.3 Root Concept

Create an easily accessible simulation engine that can be utilized by any user, allowing easy creation, visualization, and analysis of simulation data.

3.2 User Personas

Using the problem scenarios, user information for the different types of users can be inferred and individual user personas created. There are three basic expected users of the system. The first type of user wishes to view a simulation output. The second type wants to easily create simulations. The third type wants to collaborate with other users about model output.

3.2.1 User Persona 1

Ben Coleman is the manager at a local Burger King. After joining the company directly after high school, he has been given manager status after several years of faithful service. Ben has worked with computers for about five years, mostly at work, but he just bought his first home computer, an iMac. Ben is limited in his experience with simulation, a severe setback since he relies mostly on experience to determine how to run the restaurant.

3.2.2 User Persona 2

John is a BIT student at the local university and assistant manger at the Burger King managed by Ben Coleman. John has been taking classes in programming and simulation as part of his coursework requirements and is looking to apply his learning in the real world. John feels comfortable programming, but often makes errors due to his inexperience.

3.2.3 User Persona 3

Dr. Smith is a professor seeking tenure in the Biochemistry department. Because of the overall cost effectiveness of simulation, his research is relying increasingly on simulation rather than lab tests. Dr. Smith is not a programmer, but wants to create simulations to validate his research and demonstrate concepts to colleagues.

3.3 Activity Design

The design process began by utilizing the activity checklist [Kaptelinin 1999] and activity theory to extract design requirements and user activities from the problem scenarios. Several interviews with simulation package users were conducted to extract types of functionality that should be offered to compete well with existing products. The reasoning behind using the activity checklist has to do specifically with the design problem. The design problem is not focused on providing assistance in performing specific tasks or matching current business processes to a software model. Rather, it is more focused on a software paradigm shift from a single PC rich-client environment to a collaborative web environment. Introducing a paradigm shift is a radical change in the social and environmental settings users expect, and also changes the way that software development takes place. A comprehensive understanding of the current context is required to best develop a product that allows for easy transitions to a new medium. That is, people have been using simulation products for several decades. Learning about the packages that are currently used, and about the features offered, will greatly aid in moving the simulation products to a new environment.

The main tasks considered for the system are the means to provide visualization/analysis of simulation data and model creation. Simulation visualization and analysis can be accomplished in many ways, and as such, have no set task structure. A method of simulation analysis must be discovered. This method must lend itself to the new paradigm, while allowing sufficient information to be conveyed to the user for comprehension. An exploration such as this also blends well with using activity checklist, since a detailed understanding of context is required.

3.3.1 Activity Scenario 1

Ben Coleman is the manager at the local Burger King. He joined the company right after leaving high school, and has worked his way up to the top spot at the restaurant after several years of faithful service. Ben wakes up one morning, and begins to read his email. After a quick scan, he notices a message from his assistant manager, John, a BIT student at the university nearby. Ben opens the message and reads that John has created a simulation that he wants Ben to view and comment on. It seems that many customers have been coming to John to complain of the long waits for service during lunchtime. John has come up with some ideas on process improvements that could be implemented. The email includes a URL link that Ben can use to access the simulation. Ben opens the URL in his web-browser and is greeted by the simulation John created. Ben explores the simulation and finds a note saying that the simulation is running on a cluster of 16 Pentium III PCs, much more powerful than the 300 MHz iMac he is working on at home. Once Ben is comfortable with the simulation interface he begins to run the simulation.

He begins to run the simulation and an animation begins showing customers enter and exit the building, being served by cashiers, or exiting the building madly without service. As the simulation progresses, Ben can easily understand how the simulation resembles the lunch shift at BK thanks to the animated explanation. When the simulation stops, Ben notices a problem. The simulation is set up to run with 6 cashier lines, when only 5 are available at the restaurant. He opens the simulation setup, and changes the number of available cashiers to 5 and then reruns the simulation to obtain new results. Satisfied that the improvements are worth looking into, Ben emails John to say that he thinks the ideas look good and they will look into implementing them at work later in the day.

3.3.2 Activity Scenario 2

Dr. Smith leans back in his chair feeling at ease for the first time in weeks. He thinks back to the day he signed up to create a simulation model for a human immune system and tries to remember what motivated him to do such a thing. Oh yes, the grant money. Also, he is working for tenure in the Biochemistry department and working on big projects such as this look good on the vita. After spending weeks on developing a simulation model for the immune system that could be broken down into a simple queuing system he felt he had the right to take a break. The design of the system may have taken weeks, but putting together the code and animation was simple. He logged onto the web-based simulation composer and easily set up the simulation by selecting a series of components from the available simulation components and configuring a few components using his web browser, keyboard, and mouse. While there weren't tons of components available, the components he needed were present. Feeling proud of his accomplishment, he calls his graduate assistant on her cell phone to boast.

When his assistant picks up the line Dr. Smith wants to show her the model. He tells her to logon to the simulator website and locate the simulation he created. Even though she is using her parents' computer (she went to her parents' house for the weekend) she is still able to view the simulation animation. The assistant notices that part of the simulation output seems off from the expected results and comments to Dr. Smith. Dr. Smith notices the same thing and strikes up a conversion about what could be wrong with either the expectations or the simulation model.

3.3.3 Activity Design Claims and Tradeoffs

Table 2. Activity Design Features

Proposed Activity Design Feature	***Hypothesized Pros (+) or Cons (-) of the Feature***
Browser-based access to simulation toolkit	+ Increased application availability across platforms + No need for specialized hardware or software - Limits the user interface options - Browser incompatibilities may offer inconsistent UI Observation – Users should be consulted to determine what aspects of other simulation packages are valuable and would be useful in a web-based simulator. The user survey should be conducted during the initial stages of the design effort. An obstacle to this effort would be gaining access to users.
Server-based simulation execution	+ Allows access to more powerful computers for more efficient execution than on a single PC + Allows execution of simulation and visualization / analysis to be executed separately - Large user load may slow down execution Observation – Users need not specifically be worked with on this issue. The focus of this feature is on computational power and performance of hardware, which will benefit the user, but is ultimately a system architecture issue and not a usability issue.
URL accessed simulation model	+ No need to distribute models to users + No need to worry about model versioning as a single version is held at the URL - Security restrictions on model editing is a major concern Observation – This feature facilitates easy access to users in that all they have to do it click on a URL in order to use a functional system. This feature is a facilitator to participatory design.

Proposed Activity Design Feature	***Hypothesized Pros (+) or Cons (-) of the Feature***
Animated simulation execution	+ Aids in simulation understanding by user + Allows replay of parts of simulation needed to be examined more in depth - Needs animation facilities (e.g., JavaScript) to be enabled on user's computer - Animation may be slow depending on speed of computer Observation – The creation of this feature should be a focus of the participatory user design. The users should be consulted at all stages of design to ensure that the users' voice is heard and design ideas are implemented. Conducting design and feedback sessions with possible users should facilitate this effort.
Interactive simulation allowing users to change model parameters (simulation variables)	+ Allows many different aspects of a simulation to be investigated without recreating the model + Allows novice users to explore simulation changes without having to change model code + Creates a richer simulation execution experience - Not all model parameters are easily changeable Observation – This feature should be investigated with user involvement. A possible obstacle is that different user needs should be investigated and integrated into the overall product. That is, the needs of the simulation creators in providing the parameters to alter, and the simulation analysts altering the specific parameters.
Easy composition of model by setting component properties using standard input mechanisms	+ Allows novice users to easily create simulations + No need for experienced programmer - Novice users may not know how what setting the various component properties means - Not all simulation components can be set via simple properties Observation – The specific means of setting model properties should be thoroughly investigated with user involvement in an attempt to make it easy for the users to understand the process of configuring components
Increased access to models on shared system	+ Allows users without high-end systems access to model execution - Limitations on model execution environment and component availability

Proposed Activity Design Feature	*Hypothesized Pros (+) or Cons (-) of the Feature*
Increased collaboration through shared models	+ Allows multiple users at multiple sites access to models concurrently - security concerns with releasing model to outside world

3.4 Model Composer Design

The design of the model composer went to great lengths to ensure that the composition of models was unproblematic for any type of user. The information design attempted to create a model where users could simply glance at the screen and determine the composition of the simulation model. The interaction design sought an interaction model that was familiar to everyday users and allowing users to adapt their experience with normal computing tasks to model composition.

3.4.1 Information Design

Three issues drive the information design of the composer interface. The first issue is how to visually separate the composition area, the area where the model components are placed, from the composer control panel. The second issue deals with how the user will determine the functionality available in the control panel. The remaining issue is how to quickly covey to the user the current composition of the model.

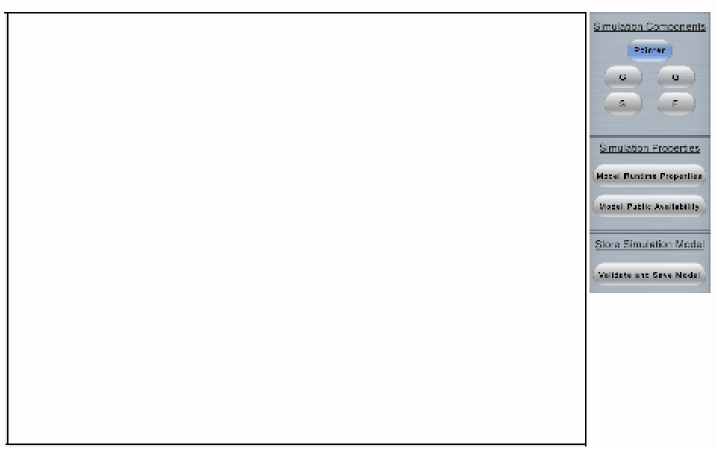


Figure 1. Composition Area Separation

Visually distinguishing the composition area from the control panel can be accomplished through the combination gestalt principles [Rosson 2002] and Vanderdonkt's [2001] layout techniques. Using the properties of closure, proximity, and area, the composition area can be distinguished in two ways. Firstly, closure to the composition area itself is established by creating a visual box around the outside of the composition area. Secondly, moving the control panel to one side of the composition area, grouping the controls together, and providing a different background color, creates proximity between the simulation controls and distinguishes the two areas from each other. As shown in Figure 1, the "squint test" [Mullet 1995] verifies that the two areas are noticeably separate entities.

The functionality of the control panel is easily observed through a series of Gestalt principles and Watzmann's [2002] ideas on typography and readability. Figure 2 demonstrates how the basic functional areas of the composer are broken down and distinguished once again through area and proximity, thus allowing simple visual identification of the button groupings. The principle of similarity is also used to enhance the visual grouping. Buttons are visibly broken into three groups and clearly labeled to indicate the functionality provided by the grouping. Typography and readability are important in this instance, due to the fact that the buttons have text displays. It would have been possible to use abstract images to convey the purpose of the button, but as Rosson [2002] notes, in some instances it may be rather difficult to use abstract images and words are more appropriate. To illustrate, what would an appropriate abstract image be to represent model availability and would the image convey the same meaning across cultures and countries? In the first grouping, one of the buttons is highlighted. The highlighting indicates the current component tool selected for inclusion in the simulation model providing a quick indication of what component may be added to the simulation model next.

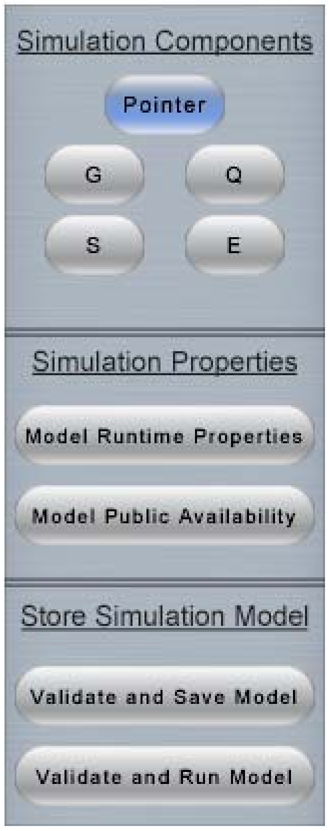


Figure 2. Composer Control Panel

The current composition of the model is accomplished thorough several different means, most noticeably color, as noted in Figure 3. The user can quickly scan the composition area and determine which components are different from other components through the different colors present. If more detail is needed, the user can check the name displayed in the component box to identify the component more accurately. Shading also provides a visual indication of the elements in the composition area, correlating with the opinions on transparency expressed by Watzmann [2002]. If a component is opaque, transparent looking, or lightly colored against the background, it is an indication that the component is not a part of the model.

The color indication applies to the components as well as the lines connecting the components. The lines connecting the components represent just that, connections between components. If a line exists between components, then a connection exists between the component elements in the direction indicated by the arrow on the end on the line. The arrow points in the direction from which the connection originated. If a connection line is not pointing directly to any other component, then a connection does not exist. An additional visual indication of a non-existent connection is the display of the line; if the line is dashed, there is no connection. If the line is solid, a connection exists. Combining all the indicators allows the composer area to quickly convey the state of the model composition to the user through use of color and simple connection lines.

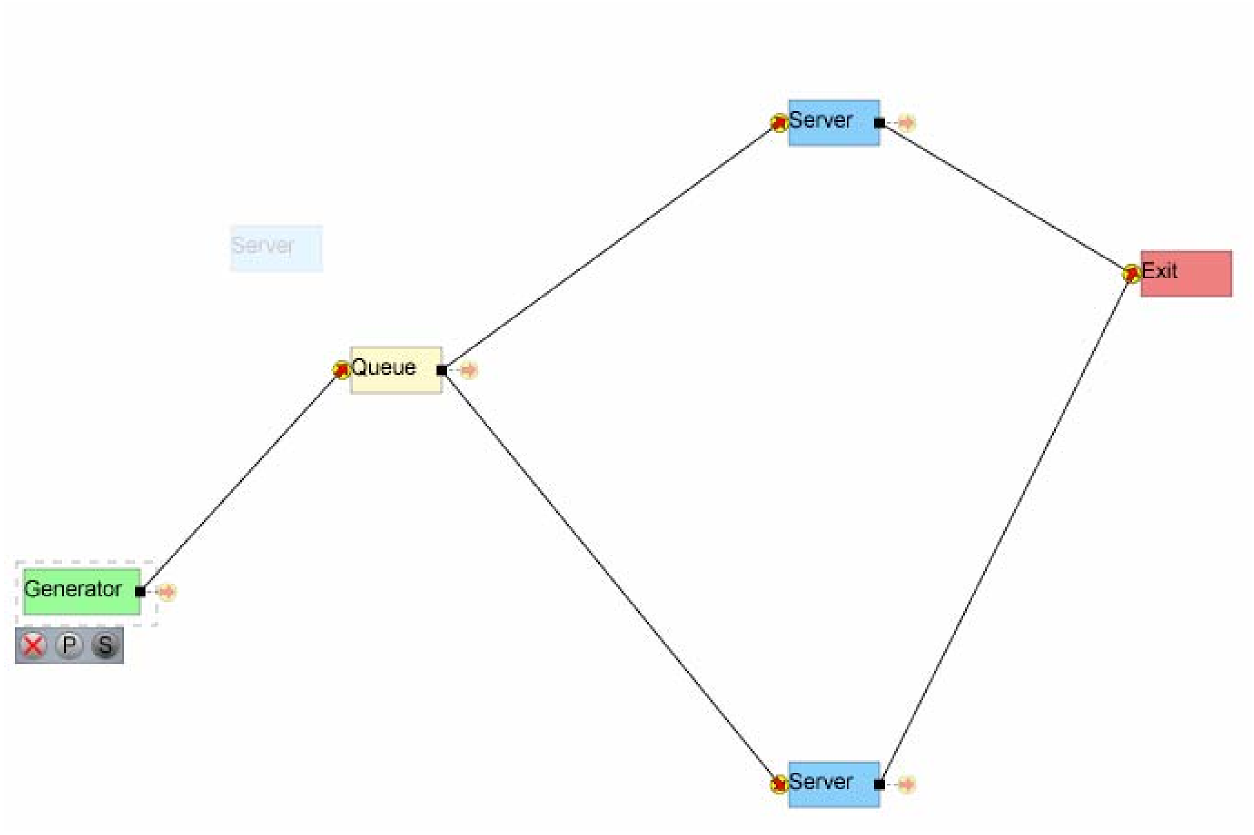


Figure 3. Use of color and connection lines to convey simulation model composition

3.4.1.1 Information Design Scenario

Dr. Smith leans back in his chair feeling at ease for the first time in weeks. He thinks back to the day he signed up to create a simulation model for a human immune system and tries to remember what motivated him to do such a thing. Oh yes, the grant money. Also, he is working for tenure in the Biochemistry department and working on big projects such as this look good on the vita. After spending weeks on developing a simulation model for the immune system that could be broken down into a simple queuing system he felt he had the right to take a break. The design of the system may have taken weeks, but putting together the code and animation was simple. He logged onto the web-based simulation composer and easily set up the simulation by selecting a series of components from the available simulation components and configuring a few components using his web browser, keyboard, and mouse. He was currently working to finish composing the simulation. He quickly glanced around the screen and saw that he had already placed several components of different types in the composer area, where there were several boxes of different color present. Some of the components had lines connecting two or more of the components together, but others had no lines coming out at all. Dr. Smith noted a missing connection and created a line to connect two of the components together. He knew the connection was added to the model when the line changed from being dashed to solid.

Dr. Smith began to focus and inspected the model a bit closer. He saw from reading the component names that he had only 4 “virus” servers and needed 5. He quickly went to the control panel, distinguished by its different color background and separated from the composition area by a series of lines. He saw that the Server tool was activated because the "S" button was highlighted, so he went back to the composition area where a lightly colored server component box appeared. Dr. Smith added the Server to the model and the component changed from lightly colored to dark colored, indicating that the component had been added to the model. Opening the model properties dialog, he looks at the colored table of available properties eventually reaching the "service Time" property, which he sets to a uniform random distribution with lower bound 0 and keeping the default upper bound of 10. Dr. Smith continued adding model components and connecting the components together to form a complete executable model.

3.4.1.2 Information Design Claims

Table 3. Composer Information Design Claims

Design Feature	***Possible Pros (+) or Cons(-) of the Feature***
Use background color to distinguish control panel	+ Control panel is separate from simulation elements, intermingling with simulation content - The color selected for background may blend into simulation content background, making the controls easy to lose for the user (if the colors of the components match the color of the background)
Use of visual "box" to identify composition area	+ Creates distinct area to which simulation components can be added. + A uniform area to provide consistency across simulations - Limits the sized of the simulation that can be created to the bounds of the composition area
Grouping of simulation controls using separator lines and proximity	+ Grouped controls link grouped functionality in the control panel - Extra visual space taken by the grouping lines
Use of words to identify functionality buttons	+ Words can describe functionality distinctly - Graphics may appeal to more users and quickly convey meaning without having to read - Word descriptions take up horizontal space
Highlighting of selected button indicates current "tool" selection	+ Visual indicator of tool selection - Users may be confused by not being allowed to deselect a tool
Color distinguishes component types	+ Quick indication of types of component types present in model + Easy means of distinguishing between different component types - Not available to users with color-recognition problems (color-blind) - Indicator colors may become limited as number of available components increase
Name identifier for component	+ Settable property used to uniquely identify components - Relies on user to provide appropriate name

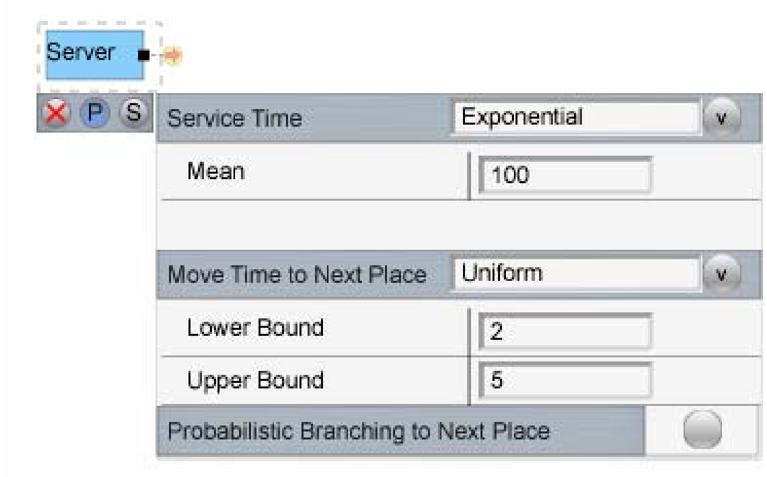
Design Feature	*Possible Pros (+) or Cons(-) of the Feature*
Shading (opacity)/ line style indicates elements that are included in the model	+ Easily distinguishes between model components and inclusion tools. + Non-included elements blend into background so that the added components are more noticeable at first glance. - Making a component too opaque may make it difficult to identify.
Arrow points in direction of connection	+ Easily detectable means of determining connection direction. - User must look at both ends of line to determine connection direction. - Overlapping arrows for multiple connections must determine which arrow to show
Table of component properties	+ Allows easy scanning of property display + Requires no user interaction to view all available properties - Length of property table may overlap visual bounds of the composer
Coloring of rows distinguishes separate rows	+ Allows easy scanning of property display + Takes no extra visual space away from the display - Colors may not be visible to color-blind users
Default property values for components	+ Creates a valid model without setting any component properties + Properties do not need to be set if the default value is OK. - Default may be different from what the user desires for the simulation

3.4.2 Interaction Design

The composer is a highly interactive application, allowing user to compose simulation models through various interaction sequences. The composer follows a what-you-see-is-what-you-get (WYSIWYG) model allowing direct manipulation [Rosson 2002] of model elements. Clicking on a model element selects the model element ands allows the element to be dragged around the composition area by moving the mouse with the left mouse button depressed. The interaction mimics the dragging of folders and files in a direct manipulation GUI operating system. The metaphor should transfer to the composer tool. Both components and connection lines allow drag-and-drop functionality. By grabbing the arrow at the end of a connection line and dropping the arrow on another component, a connection between elements is established.

The composer interface provides a variety of feedback for every user interaction. When a user moves the mouse over a component, the component is highlighted using a box around the component indicating that the component can be selected. When the user clicks on a component the color of the selection box changes to indicate that the component is now in a "selected" state. The display of a selection box corresponds with showing a box of buttons providing various operations that can be performed directly on the component. Establishing connection lines provides feedback to the user as well. When the user grabs a connection line and moves over a component, the highlight box around the component is displayed, indicating that if the user drops the connection, a connection will be established. Error feedback is provided for every operation that is attempted in error, such as establishing a connection with an illegal component. A final type of feedback is demonstrated on moving components connected to other components. The connection lines are maintained throughout the move process, displaying to the user what the model would look like if the component were dropped at any point in the process.

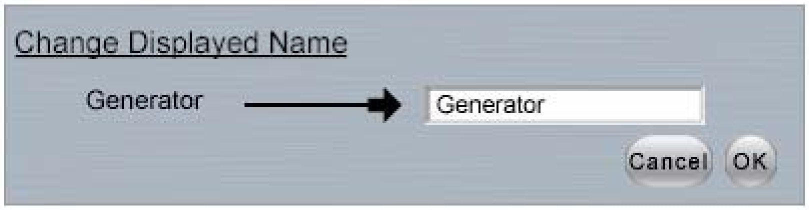
When a component is clicked, a box indicating the operations that can be performed is displayed next to the clicked component, as displayed in Figure 4. The selection box contains a series of buttons. Including buttons in the selection box indicates to the user the affordance [St. Amant 1998] of clicking. The user knows that clicking on the buttons will cause something to happen in the system. For instance, clicking the "P" button displays the list of properties for the component. Clicking the button and receiving feedback via a button color change notifies the user of the activation of the operations on the component.



Service Time	Exponential
Mean	100
Move Time to Next Place	Uniform
Lower Bound	2
Upper Bound	5
Probabilistic Branching to Next Place	

Figure 4. Selected Component with Properties Dialog

A final piece of interaction design is the affordance of double clicking. By double clicking on a component a dialog is displayed, as shown in Figure 5, indicating that the name of the component can be changed. By entering a new name in the input box, the name displayed in the component can be changed to a custom (non-default) value. The affordance of double clicking is prevalent in many operating system applications and it is hoped that the affordance caries over the composer tool.



Change Displayed Name

Generator → Generator

Figure 5. Name change display dialog

3.4.2.1 Interaction Design Scenario

Dr. Smith leans back in his chair feeling at ease for the first time in weeks. He thinks back to the day he signed up to create a simulation model for a human immune system and tries to remember what motivated him to do such a thing. Oh yes, the grant money. Also, he is working for tenure in the Biochemistry department and working on big projects such as this look good on the vita. After spending weeks on developing a simulation model for the immune system that could be broken down into a simple queuing system he felt he had the right to take a break. The design of the system may have taken weeks, but putting together the code and animation was simple. First, he logged onto the web-based simulation composer by loading the web page in his IE web browser and entering the username and password at the logon screen. He then started to set up a new simulation by clicking on buttons in the control panel to select a component tool and then clicking on the composer area to place the component in the model. After placing the component in the model (by noting the color change of the component), Dr. Smith could then click on the "pointer" tool to allow him to manipulate the model directly.

As Dr. Smith moved his mouse over various components and saw that a blue box was displayed around the component and then clicked on the component. The box around the component changed color to red and a new box was displayed next to the component containing a series of buttons. He attempted to click on the "P" button to show the properties dialog, but missed and clicked the "X" button indicating to delete the component. The system queried Dr. Smith to ensure that he wanted to delete the component, to which he responded no. Attempting again to click on the "P" button he successfully clicked the button, noting so when the properties dialog was displayed and the button turned blue. He set various properties by clicking check boxes and entering values into the supplied input boxes. When he finished with the properties he again clicked the "P" button, which returned to normal color and hid the properties dialog.

Dr. Smith continued to build the model by adding more components using the various tools and connecting the components together by dragging connection lines between components and dropping the connection when the underlying component was highlighted. He continued to move components around the composition area by dragging components like he would drag files around his Windows desktop. The dragging of components connected to other components was slightly different in that when he moved a component with connections the connections stayed in place, retaining the connections to other components but moving the connection lines.

Dr. Smith began to focus and inspected the model a bit closer. He saw from reading the component names that he had only 4 "virus" servers and needed 5. He quickly went to the control panel, and moved his mouse over the "S" button. A tool tip was displayed indicating that the current button activated the Server tool. Dr. Smith then clicked on the "S" button to activate the Server tool and moved his mouse over the appropriate place in the composition area, noting that a lightly colored version of a server component followed his mouse movements. When he moved his mouse to the appropriate location he clicked the mouse to add the new component to the model. Once the component had changed color (indicating it was part of the model), he doubled clicked on the component to bring up the name change dialog and changed the name to "virus server 5." Dr. Smith was now satisfied with the model and clicked the "validate and save" button to save the model on the server.

3.4.2.2 Interaction Design Claims

Table 4. Composer Interaction Design Claims

Design Feature	***Possible Pros (+) or Cons (-) of the Feature***
Direct Manipulation interface via drag-and-drop	+ Follows interaction paradigm of most GUI environment + Allows users to feel in control of simulation design - Does not work well with keyboard input (arrow keys)
Selection/hover box display feedback	+ Direct feedback to user indicating an operation is possible/allowed + Alerts user of which component is selected when multiple choices are available (overlapping components) - Color may be hidden to color-blind users - Color of selection box may blend with color of other components
Use buttons to initiate functional dialog displays	+ Buttons afford clicking to initiate actions - Buttons are difficult to click when the button is small - Indicting button function is difficult in limited visual space.
Error Message feedback	+ Direct response to user action so the user knows something went on. - May confuse the user if they thought the operation was possible - Need meaningful error display messages
Tool tips to display extra information	+ extra information can be hidden and displayed when need to explain operations or buttons more in depth - Tool tips not available in all environments - Large tool tips may hide pieces of the screen from view. - Tool tips display may annoy more experienced users (Future) Enhancement – Have a composer option to disable tool tips.
Single click selection	+ The affordance of clicking is linked to selection operations in many computer programs
Double click to display name change dialog	+ The affordance of double clicking is available in many computer programs - The affordance of double clicking means different things to different operating systems and software programs and may not be associated with using the composer
Color change on model inclusion	+ Feedback to the user that the inclusion operation has been performed - Color change may be hard to notice depending on the colors in use

Design Feature	*Possible Pros (+) or Cons (-) of the Feature*
A light colored component is present under the mouse when a component tool is selected	+ Direct indication of selected tool. + Follows direct manipulation paradigm - Moving mouse over components included in simulation model may obscure the view of the components underneath.

3.5 Model Visualizer Design

The visualizer component is difficult to engineer in a usable manner because much of the interface is not under developer control. The information design deals mainly with the control panel aspects of the interface and how to visually distinguish the controls from the rest of the simulation.

3.5.1 Information Design

The main information design issue deals with a user interpreting the difference between the simulation control panel and the actual simulation content. This problem arose during the audit phase of Watzmann's Information Design Process [Watzmann, 2002]. Watzmann asks, "How can you most efficiently and effectively present information required for ease-of-use of this product?" This question highlights many users' main task: actual execution of the simulation. Most of the interface a simulation analyst sees is not under direct control, but this does not mean that information design is unnecessary. The layout of the interface is important for the standard elements that are seen by all users, such as the simulation controls allowing start, stop, and rewind functionality. The simulation controls must be used by all users, and should be presented in a consistent manner in all simulations, regardless of the content of the simulation.

The simulation control layout is designed using both the Gestalt principles [Rosson 2002] and Vanderdonkt's [2001] layout techniques. Watzmann's theories about typography, legibility, and readability [Watzmann 2002] are irrelevant in this area, as the control panel should not contain large amounts of text to display to the user. Using Gestalt principles aids in creating similarity and grouping inside the control panel, thereby creating a sense of unity. The panel grouping also shows that the control panel is acting as a cohesive unit and not as part of the simulation animation content. Vanderdonkt's principles of balance, symmetry, and transparency can additionally help to disassociate the control panel from the rest of the simulation.

A major problem encountered is that the content of the simulation is unknown. This leads to problems in use of color or typography encouraged by Watzmann, as there is no way of knowing if the color or typography blends well into the simulation content. For example, if the simulation control panel was simply differentiated by a gray background and the simulation content also had a gray background, the simulation control panel would blend in and inhibit the user's interpretation of the controls. Other techniques must be used, or specific layouts created, to use in separating the controls.

3.5.1.1 Information Design Scenario

In the morning Ben opens his email to see that John has sent him a message. He opens the email in his mail client clicks on the URL that John has sent, which subsequently opens John's simulation model in a new web browser window. Ben can tell that John's model has been opened by the title at the top of the browser ("Burger King lunchtime simulation – By John H."), but also by a gray box under the browser's address bar that displays the same. Ben takes a look around the screen and sees that the images on the screen of cash registers and waiting lines are in roughly the same layout as the BK. He also notices a small box in the upper right corner containing several buttons like a TV remote control (play, stop, rewind, pause, etc). The box is framed by a set of double lines to make it stand out from the rest of the screen. He moves his cursor over to the button that is similar to a play button on a TV remote and clicks it to see if the simulation will start.

Several new images appear on the screen, moving in and out of cashier lines. Some of the images change as they move to different areas of the screen. Some of the images are shown as mad faces of people as they exit the restaurant. Ben reasons that these must be unhappy customers. Other images are shown as happy person faces with a tray of food underneath. These must be served customers. The images of people help Ben to easily recognize and understand what is going on in the simulation.

3.5.1.2 Information Design Claims

Table 5. Visualizer Information Design Claims

Design Feature	***Possible Pros (+) or Cons (-) of the Feature***
Identification of simulation in browser title bar as well as above simulation content	+ Allows quick recognition of simulation by placing title in several places + Allows recognition of simulation even in framed browser mode - Repetition of title wastes screen space - Need to maintain consistency (of wording) in title display
Allow use of real images in simulation element representation	+ Gives simulation a real feel + Allows for quick recognition of real simulation elements - Simulation objects may be better realized using abstract images and shapes
Use of remote control metaphor	+ Gives users with experience using remote controls to easily move experience to running simulation. - Remote controls do not imply that the simulation can be edited, only run
Use layout to locate the control panel	+ Gives standard place for simulation controls to appear in all simulations. - Reduces screen space that can be used for simulation content

Design Feature	***Possible Pros (+) or Cons (-) of the Feature***
Use of double border to frame control panel	+ Provides separation of control panel from other simulation elements - Reduces usable screen space around control panel

3.5.2 Interaction Design

The described interaction design issue deals with the Execution phase of Norman's stages of action [Norman 1986] in that the simulation toolkit allows users to alter certain simulation parameters. By altering the simulation parameters before simulation execution, the simulation can be tailored to specifically analyze the problem the user wishes to simulate. For instance, the simulation is, by default, set up to handle 6 cashiers, but another restaurant with 4 cashiers would like to perform the same analysis. The restaurant can change the simulation parameters in order to meet the goal they are attempting to accomplish. By allowing simulation parameters to be altered, the simulation can be used by several different users. It also makes creating simulations easier. The execution issue is how to signal to the user that the new simulation parameter is valid and has been accepted by the system.

3.5.2.1 Interaction Design Scenario

Ben opens John's email in his mail client by clicking on the URL John has sent, which subsequently opens John's simulation model in a new web browser window. Ben moves his cursor over the standard HTML "play" button on the simulation controls, clicking it with his mouse. The simulation begins to run as images of customers move about the screen. Ben wants to know more information about a customer waiting in line number two, so he moves his cursor over the image of the customer and tries to click it. The customer moves from underneath the cursor before Ben has a chance to click on it. Ben realizes that he has miss-clicked by seeing that nothing happens, and repositions his mouse and clicks again. A box appears at the right side of the screen listing status information on the customer, including the customer's wait time. A graph is also shown below the status information; it lists the average wait time of all the customers.

Ben continues to watch the animated execution of the simulation. He notices there are six cashier lines being used. This must be an error in the simulation, since his eatery only has five cashier lines available. He then searches for a way to update the number of cashiers. Ben clicks the "Simulation Parameters" link, which opens a dialog box and stops the simulation. He knows the simulation is stopped because the stop button on the simulation control panel becomes grayed out, and the simulation status images changes to a stop sign image. He searches the parameters dialog and changes the value of "cashiers" from six to five. He notices that the simulation image in the background changes by removing one of the cashier lines. Once Ben is satisfied that the parameters are correct, he closes the dialog, and restarts the simulation by clicking on the activated "play" button.

3.5.2.2 Interaction Design Claims

Table 6. Visualizer Interaction Design Claims

Design Feature	***Possible Pros (+) or Cons (-) of the Feature***
Use of standard HTML buttons for control panel	+ No need for custom code + Gain affordances inherent to system buttons - No control over button styling and display (Future) Enhancements – create custom button styling for more visual control
Clicking on simulation objects images to gain more information	+ Easy to associate displayed information with the object being clicked on - Clicking errors are more likely with objects moving in the animation Enhancements – Add clicking affordances (St. Amant, 1999) to lower error rate
Use of graphs to display information	+ Help users understand data history ± Multiple graph types for customized information display - Difficult to associate graph with an individual object Enhancements – Display graphs only on user command
New dialog boxes appear on user click	+ Direct feedback to user interaction - Multiple windows can confuse users Enhancements – derive framed interface for associating separate windows
Stopping/Pausing simulation on clicking of link	+ Provides time saving method by not requiring user to stop simulation before clicking - Clicking error may stop simulation inappropriately Enhancements - Create clicking affordances to lessen clicking error
Graying out of buttons	+ Simple notification to users of unavailable options - May confuse users who have not clicked on the option Enhancements – Create notification system with high comprehension to display button status

Design Feature	***Possible Pros (+) or Cons (-) of the Feature***
Status Bar	+ Displays current status of simulation to users in easily understandable manner + Use of abstract images conveys simple meaning - Requires screen space that could be used for simulation content Enhancements - Use fading mechanism to display system status
Altering simulation content on parameter change	+ Direct feedback to user to signal that execution of command is successful + Limitation of changeable simulation parameters does not overload or confuse users - Simulation parameters that are allowed to be changed must be set up before hand by simulation designer - Change may not be noticed if the simulation content is covered by other windows Enhancements – Create a notification mechanism to show simulation content changes no matter the context

3.6 Prototype Development

Prototyping the interfaces for the final system led to the development of three prototype models. Each prototype represents a different phase of the design process: activity, information, and interaction design. Each prototype model was developed after the completion of each design phase in order to test the concepts and ideas flushed out during the design. The prototypes span a wide range in completeness and fidelity with respect to the design of the final product. Creating the prototypes helps to extract details that were omitted during the design phase, and helps in performing usability testing later in the design process.

The first activity prototype is a low-fidelity paper and pencil participatory storyboard. The focus of the prototype is to step through the general sequence of events that a simulation analyst would go through in accessing and running a simulation. It is also useful to discuss with the user if any major features found in other simulation packages users have found useful are missing from the online version, or if any information display screens are missing from the overall sequence.

The second information prototype is a medium-fidelity PowerPoint prototype. The goal of this prototype is to demonstrate sample layouts and information design principles, which could be applied to providing the control panel to users. Several prototypes were created to demonstrate various layout techniques, one of which is shown in Figure 6. The set of prototypes would give users a better idea of the finished product's appearance, without providing too much fidelity to lock users into a final product. Users can view different control panel models and then approve, disapprove, or redesign the suggested information layouts that designers have developed. The initial information display is another issue to be resolved. Questions such as the following could be asked:

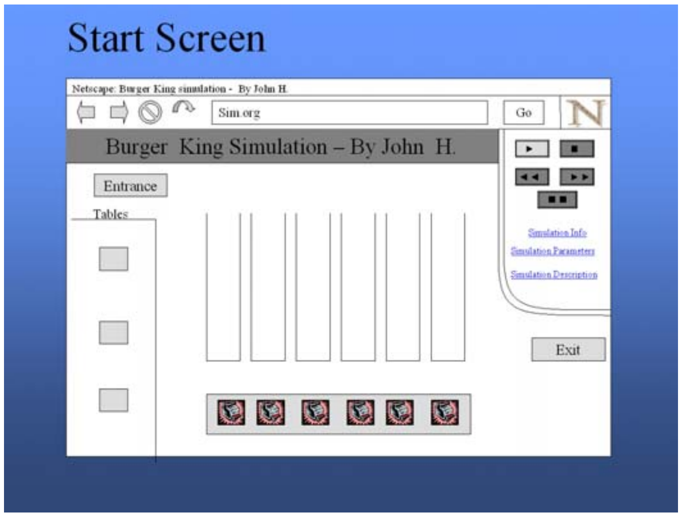


Figure 6. Information Design Prototype

- Should a splash screen be shown that gives a description of the simulation?
- Would a simulation setup screen, allowing the user to set parameters instead of relying on defaults, be more appropriate?
- Should the user simply be shown the start of the simulation?
- Is the simulation executing or paused on initial display?

These are a good set of questions with which to start off users, continuing with other issues that arise. All the information gathered here will help refine the prototypes in the information design phase.

The third prototype is a medium-to-high fidelity web-based representation of the simulation product, shown in Figure 7. At this stage, most of the functionality is built into the prototype, allowing for a vertical test to be run. The goal of this prototype is to determine user reaction to feedback mechanism impacts on user experience. One of the main drawbacks of a web-browser based system is that communication between server and client is a major bottleneck. The prototype can explore methods that can be used to hide this bottleneck from the user. The prototype should also explore if the feedback mechanisms are appropriate and well placed in the task cycle. Another issue that can be explored is the performance of a semi-workable system animation, and how users respond to difference performance levels.

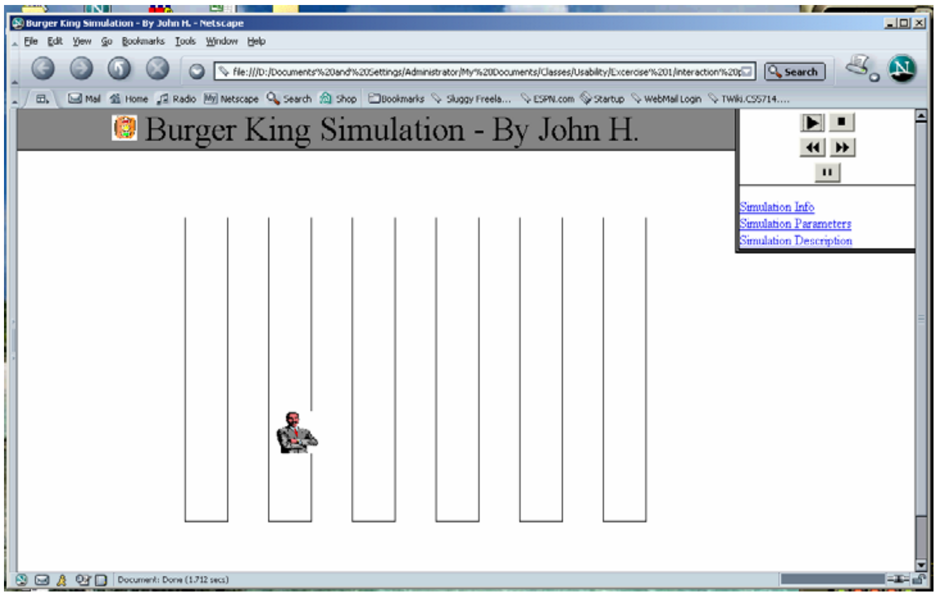

Burger King Simulation - By John H. - Netscape
File Edit View Go Bookmarks Tools Window Help
Search
Burger King Simulation - By John H.
Simulation Info
Simulation Parameters
Simulation Description
Document: Done (1.712 secs)

Figure 7. Interaction Design Prototype

3.7 Usability Testing

Performing usability testing using the individual prototypes aids the designers in confirming claims made about the interfaces and underlying system functionality. In addition to confirming claims, the testing can highlight problems the designers had not considered during the design phases. Due to constrained time limits, only the visualizer component underwent the usability testing process, and only through pilot testing. A comprehensive set of tests could be performed using the test plan developed, so the test plan is presented for completeness.

Under the time constraints, the decision was made to work with the visualizer for usability testing. It was thought that the visualizer would be the tool that most system users would be interacting with a majority of the time, and hence stood to gain the most through the testing process. The information contained in this section details the prototype systems. The prototype systems are explained and series of empirical and analytic tests are designed to qualify claims in the design. Finally, results of a pilot test study are detailed to determine if the tests are valid and working toward their stated goal.

3.7.1 Analytical Testing

The analytical testing focuses on a single information design issue and three claims derived during the information design phase of prototype development. The issue is the user's detection and interpretation of the simulation control panel in respect to the rest of the simulation content, which the simulation tool designers have no control over. This is a critical issue because the simulation control panel is the user's only means of executing the simulation and needs to be quickly and easily detectable and understandable at a glance.

3.7.1.1 Test Focus

The prototyping effort derived a series of three claims attempting to aid the user in resolving this issue. First, layout techniques presented by Vanderdonkt [Vanderdonkt 2001] were used to provide a consistent location for the control panel across all simulations, despite the content. Second, following Gestalt principles [Carroll 2002], a double line was drawn bordering the control panel to provide a visual distinction of separation from simulation content. Finally, a remote control metaphor was used to allow users with experience in using remote controls a more easy learning experience. The testing focuses on determining if these three claims are valid and sufficient for dealing with the control panel interpretation.

Scriven [1967] describes an approach known as mediated evaluation. This approach emphasizes that analytical testing be performed "early and throughout the design process," which motivate later empirical evaluations. Acknowledging that the main issue at this point is an information design issue (occurring in the middle of SBD) and following Scriven's approach, an analytic evaluation should be performed. In order to determine if these claims are valid and sufficient, an analytical usability inspection is required. Real user data is not what is being sought, but opinions on whether the information design techniques employed are useful. Users and usability experts will view the interface and give opinions and comments on whether the claims are valid, or if other techniques are needed to allow resolving the issue.

3.7.1.2 Test Plan

With consideration of the goals set forth in the test focus, it was ascertained that a heuristic evaluation [Nielsen 1992] would be an appropriate test to perform utilizing the existing information design prototype. Heuristic testing allows a series of experts to evaluate the prototype with respect to set usability principles; it also determines whether the interface is effectively conveying the simulation control panel information to the user. Heuristics critique the means by which the prototype was constructed and perhaps gather new design ideas and alternatives for failings in the design.

Nielsen [1992] argued that performing a heuristic evaluation using "double experts," that is experts in the domain as well as in usability, is more effective than using typical experts or regular users. In this situation, using double experts is a demanding task. The browser-based simulation toolkit's focus is a change in software domains, facilitating a move from the PC to the web. Very few, if any, simulation toolkits are available in a web environment, so finding a double expert in both simulation and web design is a challenge. To circumvent this issue, it is beneficial to utilize two sets of experts, one set specializing in simulation and the other specializing in web design. A group of five simulation experts and a group of five web design experts will be asked to individually evaluate the prototype according experience to the following heuristics as set out by Nielsen [1994]:

- Use simple and natural dialog
- Speak the users language
- Minimize memory load
- Be consistent

- Provide feedback
- Provide clearly marked exits
- Provide shortcuts
- Provide good error messages
- Prevent errors
- Include good help and documentation

The evaluations will be collected and analyzed to determine existing usability problems with the interface, and recommendations will be made on improving the interface. This evaluation can be implemented under the exempt considerations with respect to the Virginia Tech IRB.

3.7.1.3 Test Execution

A small, 15-minute pilot heuristic test was performed to judge the validity of the testing method. A web design expert met with the prototype designer and the general design situation was described (an interactive web-based simulation animation). Next, the web design expert was asked to evaluate and discuss the prototype interface with respect to Nielsen's principles, as well as web design best practices. Several observations were collected:

- The use of the remote control metaphor works well
- The remote control metaphor may need further research and development as some remotes are not presented the same as the prototype remote
- The use of the double line does create a visual distinction for the remote control
- The simulation modeler may take advantage of the set layout to make the control panel appear to be part of the simulation (e.g., causing animation to run along the border of the control) lessening the visual distinction.
- The links provided to other sections of the simulation should be grouped and use consistent language
- It may be useful to hide (instead of disable) buttons offering unavailable functionality
- Offer means to hide the remote when it is not in use to create a larger screen area for the simulation content

3.7.1.4 Test Analysis

The results from the pilot test reinforce that the claims created are valid, but also state that the prototype could use further development to portray the design claims accurately. For example, the tester raised the point that the remote metaphor worked well, but on some physical remote controls the play button also acted as the pause button in certain situations. So it may be useful to combine the play and pause buttons into a single button. It was also stated that the double lined borderline does create a visual distinction for the control panel. The concern about allowing animation to occur around to the left and under the control, thereby lessening the visual distinction of the control panel, was new to the designer. If many of the experts agreed that allowing the simulation content to use the area under the control panel lessened the visual distinction, then it could require a redesign of the overall simulation toolkit layout.

The focal point of the pilot test was to determine if the overall test execution would proceed smoothly and if the test was appropriate for the issue(s) being investigated. Conducting the pilot test introduced several helpful ideas and can be seen to be a valid testing method when using web experts for the information design. An idea that was not considered before the pilot was that the process of heuristic testing is highly dependent on expert experience. Selection of the expert evaluators is an important concern. Also, another pilot test using a simulation expert would be useful in determining if simulation experts are appropriate to use in this type of testing.

The conclusion that the remote control metaphor may need further refinement was unexpected. Instead of performing heuristic evaluation, it may be more beneficial to formulate a user model for the remote control metaphor. Creating a model with actual user participation will assess whether the remote control metaphor is appropriate under a web-based simulation conditions.

3.7.2 Empirical Testing

The claims relate to design issues important to the ease of use of the system. Any user should quickly and easily be able to find extra information relating to any object in the simulation so as to gain a better understanding of what is occurring in the situation. Also, appropriate feedback should be given to the user to signal that actions being taken are being accepted by the system.

3.7.2.1 Test Focus

The empirical testing concentrated on two design claims. The first claim relates to acquiring additional information on simulation objects, the objects being animated and moving around the screen. The claim states that the user can gain extra status information concerning any simulation object simply by clicking on the object. The second claim involves direct feedback to the user on parameter change input. The claim says that altering the simulation content directly after a parameter change, as opposed to asking for confirmation, will be as successful and noticeable as waiting for user confirmation before changing the simulation content.

The interaction design phase of SBD introduced a user model for how to obtain extra information and feedback from the system. In this phase of the testing, the model and feedback mechanisms should be evaluated to determine if it is indeed unproblematic for the user to accomplish the system goals, and also if the feedback is noticeable and useful in accomplishing these goals. The evaluation requirements blend well with an empirical laboratory study. Performing an empirical study, using specific performance targets, allow the designers to determine if the system goals are being met.

3.7.2.2 Test Plan

The goal of the test focus is twofold; the first goal is to determine if the amount of system feedback is detectable and sufficient, and the second goal is to evaluate the ease of the mechanism by which extra object information is shown to the user via clicking. The test will consist of a series of tasks that the user is asked to perform. For a full-scale usability test, a group of 40 – 50 typical users comprised of students, teachers, and a variety of community members will be gathered and asked to perform the series of timed tasks on the interaction design prototype. In addition, designers and usability experts will observe some of the users, taking notes on exact user actions. At the end of the task session, the evaluator will ask the users questions about the experience as well as answer any questions the observers may have.

The tasks will be randomized and repeated two or three times for each participant. Performing the tasks multiple times and randomizing the order is an attempt to normalize the learning effect from performing one task to the next and render learning an independent variable. Each participant will perform the tasks on a different simulation chosen randomly from a group of five different simulations. Using various simulations allows the results to be generalized across all simulation content and bring validity to the claims in general, not with respect to a single simulation. In using different simulations, it is necessary to create reference tasks [Whittaker 2000] so that the differences in the simulation content do not skew the test results significantly.

For future testing, it may be advantageous to test across different input devices (e.g., mouse, trackball, pen input, voice commands for disabled users), browsers (e.g., IE, Mozilla, Opera), and platforms (e.g., Windows, UNIX, Mac) to determine if the type of input device has any impact on user performance. Device considerations are out of the test scope in this case, so every user will be asked to use the same setup type while performing the evaluation. This type of testing qualifies under the exempt qualifications of the Virginia Tech IRB in that no danger is coming to the participants.

3.7.2.3 Test Execution

A simple pilot test of a single user was performed using a 400 MHz Windows PC using a mouse on the Mozilla web-browser. The user was asked to perform the following tasks, under evaluator observance, and timed:

1. Start the simulation
2. Find information relating to the user in line two.
3. Find information relating to the moving user.
4. Change the number of cashier lines from six to five.

The user took 3 seconds to perform task 1, 3 seconds to perform task 2, 22 seconds to perform task 3, and 7 seconds to perform task 4. The critical parameter [Newman 1997] in this evaluation is correct click. If the number of missed clicks is high, it may require a method redesign for showing user information. When the user had completed the tasks, a series of questions were asked and responses recorded:

<u>Rate from 1-5, 5 meaning you completely agree, 1 meaning you completely disagree:</u>

1. How easy was it to tell when the simulation was running (1-5)? Response: 4
2. Did the status bar help in determining the simulation was running? Response: what status bar?
3. Did you find the statistics graph helpful? If no, why? Response: No, there were no labels on the axes

4. Could you easily tell when the cashier change took effect? Response: I didn't notice the background change until I closed the dialog. I usually get an "OK" and "Cancel" button that makes the change occur.
5. How easy was it to click the user in line 2 (1-5)? Response: 4
6. How easy was it to click the moving user (1-5)? Response: 1

3.7.2.4 Test Analysis

The results of the pilot test are mixed. On the positive side, the pilot test shows that the empirical evaluation is appropriate to gather the type of data described in the test focus. Improvements are required, however, for tasks that require a large amount of time, such as clicking a moving user. For simple tasks such as starting the simulation, the tasks are completed in a short amount of time, signaling that they are easy to perform.

While some results were expected, other results were unusual. It was expected that clicking on the moving user would be difficult, but not as difficult as the user found it to be. The user was quickly able to determine that clicking on the user was the means in which to display extra information. The user experienced no difficulty in clicking on an unmoving simulation object, but did experience difficulty in clicking on a moving user. Consequently, the user was only able to click on the simulation object only after it stopped moving. It may be necessary to implement clicking affordances or other mechanisms to assist in clicking on moving objects.

An expected issue was that the user was completely unaware of the background simulation content change during parameter change. This was a concern during prototype design; the pilot test proved that the design feature deserves further testing. If the feature is not discerned by a large percentage of the users, it may be removed from the design entirely.

Overall, the pilot test proceeded as planned. While the prototype was simple, it was effective as a means for testing. The user found the display easy to read and understand. The animation proceeds irregularly at times and so could be made smoother for a more enjoyable user experience.

3.7.3 Test Plan Summary

There are two parts to the initial testing plan. The testing begins with an analytical heuristic test utilizing a series of five web design experts and five simulation experts who evaluate the information design prototype following Nielsen's [1994] general usability heuristics. This testing should aid in reinforcing the information design claims concerned with user interpretation of the simulation control panel. The heuristic testing is also useful because it notes any design pros and/or cons the experts may detect.

During heuristic pilot testing, it was determined that the prototype's display of a remote control metaphor needed refinement. Different remote controls function in different ways in relation to the sequence of buttons being pressed. An analytical test may need to be performed to construct a user model of a remote control interface in order to properly display the controls to the user.

The heuristic testing, and the user model creation, will be useful in the formative evaluation of the interface.

The second test to be performed is an empirical test utilizing 40–50 users. The users will be asked to perform a series of observed and timed tasks, some of which will be repeated, to determine if the critical parameters of quick and accurate clicking are met. Also, an exit survey will be given to the users to obtain user comments on the ease of the tasks using the interface. To limit the number of dependent variables, the tasks' order will be randomized to minimize learning effect. Also the users will all work on randomly chosen simulation content, so that the results can be generalized to all simulations. All the tests will be performed on the same type of machine and platform so that cross platform considerations will not come affect the outcomes.

This test is more of a summative evaluation to determine if the information retrieval mechanisms and feedback built into the system is sufficient for the users. The tests will determine if the interface aids in meeting the system goals in a smooth manner. A few problem areas were already revealed through the pilot testing, such as when the user takes an extraordinarily long time with clicking on a moving system object. These existing problems need to be corrected before the system is released. If it is determined that the task is taking too long for a high percentage of users, a redesign may be required. The testing plan is summarized in Table 7.

Table 7. Design Concerns and Testing Methods

Design Concern	Methods of Testing	Pilot Test findings	Potential Design Impacts
Constant layout for all simulations	Heuristic testing (analytical)	Content creators may take advantage of layout to undermine the visual separation of content and controls.	May need to create a more effective layout than current prototype offers so new users will not get confused
Use of double line for visual separation of content and controls	Heuristic testing (analytical)	Use of double line is effective	Design is good
Use of remote control metaphor for simulation controls	Heuristic testing (analytical) / (possibly) User Model Creation	Different types of remote controls have different uses for the same buttons at different times	Redesign of the remote's buttons to overload functionality and hide non-usable buttons
Clicking on simulation objects to obtain more in-depth information	Empirical testing in laboratory with users	Moving simulation objects very difficult to click	May need to design in functionality to slow down moving objects or create clicking affordances to ease user clicking
Display of content change on parameter change	Empirical testing in laboratory with users	Change is not noticeable	May eliminate the functionality for a confirm mechanism

Chapter 4: Web-Based Simulation Strategy

The design and development of a web-based application is not a trivial task. Web-based systems are more than a set of Extensible HTML (XHTML) [XHTML 2004] tags randomly placed on a page. Rather, they are an intermingling of a family of technologies, each technology with a specific purpose and set of best practices. With the breadth of technologies and standards available to web-developers, a careful balance between the use of the different technologies must be maintained to create a usable, maintainable, and execution efficient system.

Any modern web-based system should take the available standards put forth by the W3C into consideration, and a web-based simulation is no different. A web-based simulation should be developed firmly on the basis of web-standards, utilizing the full potential of the web platform while making the application available to a wide variety of platforms as described in section 4.1. Section 4.2 describes how component-based development of simulation models can be used to facilitate easy model creation for non-simulation experts. Section 4.3 details how XML can be used as a base for the simulation model, allowing easy communication of model information between system components. Extending the simulation system to handle a greater set of executable simulation objects is important to the maintainability of any simulation. Section 4.4 describes how the overall architecture and the XML model format take the extensibility of the overall simulation system into consideration. The creation of a consistent and extensible communication protocol between simulation components is provided in section 4.5.

4.1 Web-Standards Usage

The design and implementation of a modern web-based simulation system should be based on web-standard technologies, such as XML and DOM [DOM 2004], and targeting standards implementing tools and devices. By using web-standard technologies and development processes the availability of the simulation application is greatly increased. Anyone with a standards-based device is able to access the application. Various users may view a different display depending on the device used to access the system (e.g., a fully fledged interactive SVG interface for browser users verses a simple text display for PDA users). Regardless of the device, all of the information generated by the simulation should be accessible to handicapped users by following practices set forth by the WAI initiative [WAI 2004].

Using web standards to develop a web-based system creates an understandable development platform in which the system developers and maintainers need not be versed in the platform specific implementation of a tool, but simply the standards that the tool implements. For example, a web developer need not know the differences in implementation of the rendering engine of Netscape Navigator verses Microsoft Internet Explorer, but instead that they both understand the XHTML that the developer generates.

Teaching developers to rely on standards opens up the range of platforms available to the system, in addition to making the system more “future proof” and maintainable. The W3C endured great hardship in developing a family of technologies with specific intended use and purpose. Each W3C technology was created for a definite purpose in developing web applications, allowing high component reusability and making maintenance of web-based systems easier than old-style web-systems. For example, technologies such as Cascading Style Sheets (CSS) [CSS 2004] have been developed to create an abstract and reusable, yet exact, styling platform allowing site-wide style information to be cached on the client. The technology has been specially engineered to allow wide-scale reuse in an efficient manner, making site-wide or individual page style maintenance of an entire site as easy as changing a line in a text file. Newer technologies such as XForms allow the creation of information models in XML that are independent from the view of the information displayed to the user. The information model and view need not be linked. This creates an environment in which one aspect of a form-based application can be easily upgraded and modified without affecting the rest of the system. Technologies developed by the W3C are created to foster easy maintenance and high reusability in applications using those technologies, while promoting application availability to a wide variety of platforms.

In addition, the W3C goes to great lengths to ensure that newly developed specifications are compatible with the existing versions of specifications. Therefore, if an application is developed using HTML 2.0 and scripted using the Document Object Model (DOM) 1.0, the application should still work as intended with a viewer capable of XHTML and DOM 3.0. While not all W3C technologies have met the goal of high reuse and easy maintainability, the creators of more modern technologies have taken time to ensure that the technologies achieve this goal. Developing modern systems using modern web-standards technologies greatly aids in the creation of a forward-thinking, understandable, and maintainable system.

4.2 Component-Based Simulation Model Composition

Component-based development eases the task of model composition for model developers. Model developers can freely pick and choose from a set of prepackaged simulation objects and compose a simulation by setting various properties of the individual objects. Model developers are not responsible for the implementation and testing of individual simulation model components only configuring the components used by the simulation model. The modeler composes the various simulation components to create the expected model behavior. Using a set of readily-available and well-tested components for model composition reduces the responsibilities of the model developer to only implementing the model logic.

Providing model components for composition purposes makes it possible to break down the model development process into a visual component-composition process. A visual composition process allows model developers to assemble simulation models through direct manipulation of model components in a WYSIWYG environment. A visual component composition process opens the model composition process to model developers that need not be experienced in a particular simulation modeling language. Visual composition also allows the simulation system to separate the model developer from the specifics of the backend implementation technology. To illustrate, a model developer can compose a model using visual component composition assembling model components where the component implementation is written in different programming languages. The backend system can translate the differing model components and assemble the components into a working system without the modeler knowing the simulation engine implementation specifics.

The downsides of using a component-based composition process are the limitations imposed on the modeler by the properties exposed by the components and composing complex models. The components available to the modeler may not fulfill the requirements for the model development. In that case more simulation components need to be developed to facilitate the modeling effort delaying the modeling process. Even if the components needed by the modeler are available the configurable properties on the components may not allow the modeler to describe the model logic appropriately. For example, a modeler needs to include a bounded first-in-first-out queue in a simulation where if 10 items are in the queue then more items attempting to be added to the queue are rejected. The simulation environment only offers unbounded queue components. Therefore a new component needs to be developed especially for the modeler. The model logic may also be especially complex and had to describe adequately using a simple model composition process.

4.3 XML-Based System Components

XML is a natural starting point for the design of a web-based system, as most of the modern web-standards specifications are based on XML. Under the simulation system strategy developed here, XML is used as the simulation model representation format and the communication format between components. Representing the simulation model in XML creates several benefits regarding component coupling and system maintenance, but forces model developers to adhere to the XML format. Using XML as a communication mechanism allows easy cross-platform and cross-language implementations to be developed, but create problems when execution efficiency is considered. Discussion of the issues involved in using XML as a basis for system components follows.

4.3.1 Simulation Model

XML is a document format where the data stored in the document is marked with a series of metadata tags to add semantic meaning to the data elements. An XML document can be used to represent a simulation model by creating a set of metadata tags to accurately describe the data held by the XML document. Parsing the XML document allows the semantic meaning of the data to be extracted and translated into an executable simulation model. Representing the simulation model in XML allows easy composition and parsing using a variety of web-standard APIs and tools. XML also provides a human-readable model description for easy debugging or generation by hand, for those developers not wishing to use provided tools. Utilizing XML as the model representation format brings many benefits in document interpretation and construction and a few drawbacks when it comes to mapping the XML to the simulation representation model.

A major benefit of utilizing XML is the availability of a wide variety of tools, platforms, and other technologies that can be used to enhance the model representation. Dozens of XML parsers are available for a variety of needs, programming languages, and platforms. Many are specialized to deal with distinct execution conditions (e.g., small memory footprint parsers built for mobile devices). Document construction and navigation tools such as DOM 3.0 and XPATH [XPATH 2004] make it easy to navigate to specific document content, or create new content dynamically. XML Schema Description (XSD) [XML Schema 2004] and associated description formats allow document instances to be described in order to exactly match a set of given conditions. This allows XML document content specifications to be maintained outside the executable code. Reference technologies such as XInclude [XInclude 2004] allow content in multiple documents to be associated with each other. A multitude of XML tools are available for use, as noted in Table 8, allowing easy creation of XML document interpretation logic and freeing developers from having to hassle with the construction of custom simulation model parsers and validators.

Table 8. Sampling of Available XML Technologies

XML Technology	Purpose
DOM	XML document parsing, construction, serialization, and dynamic content updating
SAX	XML document parsing (following push paradigm)
StAX	XML document parsing (following pull paradigm)
XML Encryption	Specialized encryption/decryption processes for XML content
XInclude	Method of general purpose inclusion of outside content into an XML document to create a single document instance
XML Key Management	Allows obtaining key information (values, certificates, management of trust data) from a web service
Xlink	Series of XML elements and attributes used to create and describe links between resources (XML or otherwise)
Xpointer	XML document fragment identification technique
XML Query	Flexible query facility to extract data from XML documents (local or networked document), allowing access to collections of XML documents like databases
XML Schema	Extensible means of defining XML document structure, content, and semantics
XML Signature	XML compliant syntax used for representing the signature of web-based resources (including procedures for computing and verifying such signatures)
Xpath	Language for addressing specific parts of the an XML document
XSLT	Transformation and presentation language for XML

A drawback of relying on XML as a representation language is that XML inherently uses a hierarchical data model. A strict hierarchical data model proves difficult to map over to other data structures, especially graph data structures where circular references exist. It is possible to map data structures over to a hierarchical structure using mathematical graph embedding techniques while still creating data structures within the bounds of the XML hierarchy. For example, a graph structure can be embedded in an XML hierarchy through utilizing "id" references for circular references. A circular reference is created in cases where model components reference other model components that have already been declared in the model. Every element in the XML is assigned an "id" attribute. When a reference to an existing element is needed, an "idref" attribute is created to reference the appropriate id. Creating a structure utilizing idrefs allows non-hierarchical data structures to be embedded in XML, but requires additional processing or a specialized parser, since the data model is not conforming strictly to the expected XML hierarchy. Creating non-hierarchical data models in XML is possible thorough various graph embedding methods but requires additional post-parsing processing to setup the appropriate data links when converting from XML representation to executable object model, making the document interpretation less efficient than a strict hierarchical data model.

The benefits of extensive tool availability make the construction and maintenance of simulation models easy. Utilizing graph embedding techniques helps work around the difficulty of mapping simulation object models to the XML hierarchy. Depending on the complexity of the simulation model, the embedding problem may be so cumbersome that creating a model description in XML is extremely difficult or even impossible. The use of XML for the simulation model looks to be a workable solution when considering that most WebQS3 simulation models are easily reduced to a simple object graph of component connections. The use of XML is especially convenient in the case given the availability of tools on the client available for model construction.

4.3.2 System Communication Protocol

The individual components of the system architecture communicate using XML messaging. Relying on XML messaging instead of specific binary APIs decouples the system modules from one another facilitating a more maintainable system. The system components need not know any information about the specific implementations of other components in the system. They need only know what set of XML tags the other system components accept as input. Additionally, it is possible for the system components to work off of the same XML document instance, accessing the specific pieces of the document that the component needs to operate. For instance, the visualizer components of the system can access the visualization information stored in the document while the execution engine accesses the simulation model information. All of the information is stored in a single XML document and reused across components.

XML is also a good basis for a simulation system, especially in the frame of a web-based system, because XML is built to be cross platform and cross language. The use of XML allows the developers of the simulation architecture components to use any platform and programming language that fits the needed conditions. Case in point, the simulation engine server can be constructed in Java on a UNIX Box while the visualization manager is built using C# on a windows client. The implementation language and technology of individual components in the system architecture can be tailored to the needs of the component, while relying on XML communication between the components. The only requirement for each component is an available XML parser to parse the communication messages, a certainty in most modern programming environments.

An added benefit of utilizing XML as the communication format is the ease of porting to a web-services application thereby creating a services oriented architecture (SOA) for the entire simulation system. Any XML communication can be packaged in an XML-RPC request to activate the service. The functionality of the simulation engine can be exposed through a web-services API allowing additional clients to use the engine in a Business-to-Business (B2B) service application, reusing the functionality of the execution engine as the caller user deems. Other simulation service providers can reuse the simulation execution engine to provide extended simulation services in a competitive environment, or specialize the environment for specific purposes. To illustrate, a company can setup a web site that use the simulation service to specialize in manufacturing simulations, or provide French and Spanish language front-ends to international users. Both examples include specific application functionality that the original WebQS3 system does not provide, but that can be built using the execution engine as a backend. The simulation web service needs only to be passed an XML message or XML model that the simulation engine can interpret. Providing simulation services through web-services is the next iteration of "Simulation-as-a-Service" proposed by Wiedemann [2001].

A disadvantage inherent in XML is the verbosity of the format due to its text base. XML formatted documents are significantly larger in terms of file size than other description formats, especially binary formats. All of the extra XML element tag information associated with each piece of simulation information adds additional content to the document. In the end, the simulation model file size may increase dramatically in order to accurately describe the model or a simple simulation message. The format also takes longer to parse than simple binary objects. Every stage in which the XML is converted from text to binary, the document will need to be re-parsed. This requires additional processing time, which may be substantial depending on the size of the document.

However, both of these intrinsic shortcomings have workarounds. For example, XML formatted documents are easily compressible using standard utilities such as gzip. Such compression tools provide a means to decrease document file size and the overall network resource requirements when communicated over a network. In one instance, a 350MB XML formatted simulation execution record was compressed to 18MB to save on network bandwidth. The compression represents a 95% compression ratio in that instance. Researchers are attempting to workaround the drawback of long parsing time through specialized XML parsing hardware [Fontana 2004, DataPower 2004, Tarari 2004a, Tarari 2004b] allowing performance-optimized XML handling. Special hardware reduces the execution time of the XML parsing process freeing the rest of the system to perform other needed operations.

The benefits of extensibility and the innate cross-platform nature of the XML specification create an environment particularly tailored to the web. While problems exist in using XML as the simulation model representation format, they can always be applied to take advantage of the benefits provided by new technology. In the case of web-based simulation, the benefits provided by utilizing XML outweigh the drawbacks. Such is the case especially on the client side where the availability of tools and services is restricted. Utilizing an XML format to communicate between the simulation layers is one of the few communication services available on a wide variety of client systems without having to install specialized software.

4.4 Simulation System Extensibility

The simulation engine for any simulation system should be extensible, allowing addition of new simulation components to broaden the functionality of the system. Increasing the functionality of the system opens it to a new faction of users who wish to use model components not initially available in the system. The WebQS3 system provides two facilities for simulation engine extensibility: further developing the simulation engine by including a new simulation object package, and replacing the simulation engine entirely with a new engine. Not only must the simulation engine be extensible, the XML model description and the tools acting on the model must include extensibility mechanisms to handle new simulation elements. The facilities for providing system extensibility are discussed below.

4.4.1 Simulation Engine Extensibility

The first option for extending the simulation engine is to develop and include a new set of simulation objects. The current simulation engine, which is based on the Java language, allows new simulation objects to be included via packages of components implementing a provided **`SimulationComponent`** interface. A simulation object designer can create a new set of simulation objects implementing the **`SimulationComponent`** Java interface and package the objects in a new Java package. Adding the designers' package to the simulation engine at load time allows the individual components to be discovered via runtime inspection methods (Java Reflection). The Java Runtime Environment (JRE) can then load the needed simulation components into the simulation engine using normal class-loading methods.

The second option for extending the engine involves completely replacing the engine with a new package capable of executing the simulation objects needed. The new engine must conform to the input/output specification of the engine component interface. Implementing decoupled system components allows the engine to be removed and replaced easily without affecting other objects in the system. Creating a decoupled system where the individual system components do not rely on specific functionality of other system components allows new functionality to be added to the system or completely replaced. This allows uncomplicated maintenance of system components.

4.4.2 Simulation Model Extensibility

The XML simulation model can reference the new simulation objects in the same manner as existing simulation objects. Every simulation component has a component name and component module. The simulation component name corresponds to the XML element name of the simulation object and the simulation model corresponds to the XML element namespace (XML-NS). The XML-NS represents a means in XML of uniquely identifying the scope of an XML element; just as a Java package or C++ namespace represents a means of identifying the scope of an object. Following the convention described, the XML simulation model can reliably identify any simulation object available to the simulation engine, or identify when simulation objects are not available. Figure 8 provides an example of the module and component referencing. As shown, the XML simulation model references a **`<queue:Server />`** element. The simulation engine currently has **`Server`** executable objects available: a **`Server`** object in the **`Web Systems`** module and a **`Server`** object in the **`Queue`** module. Noting the XML-NS of the XML element references the **`Queue`** module, the simulation engine can provide a unique mapping between the XML representation and the executable representation of the simulation model.

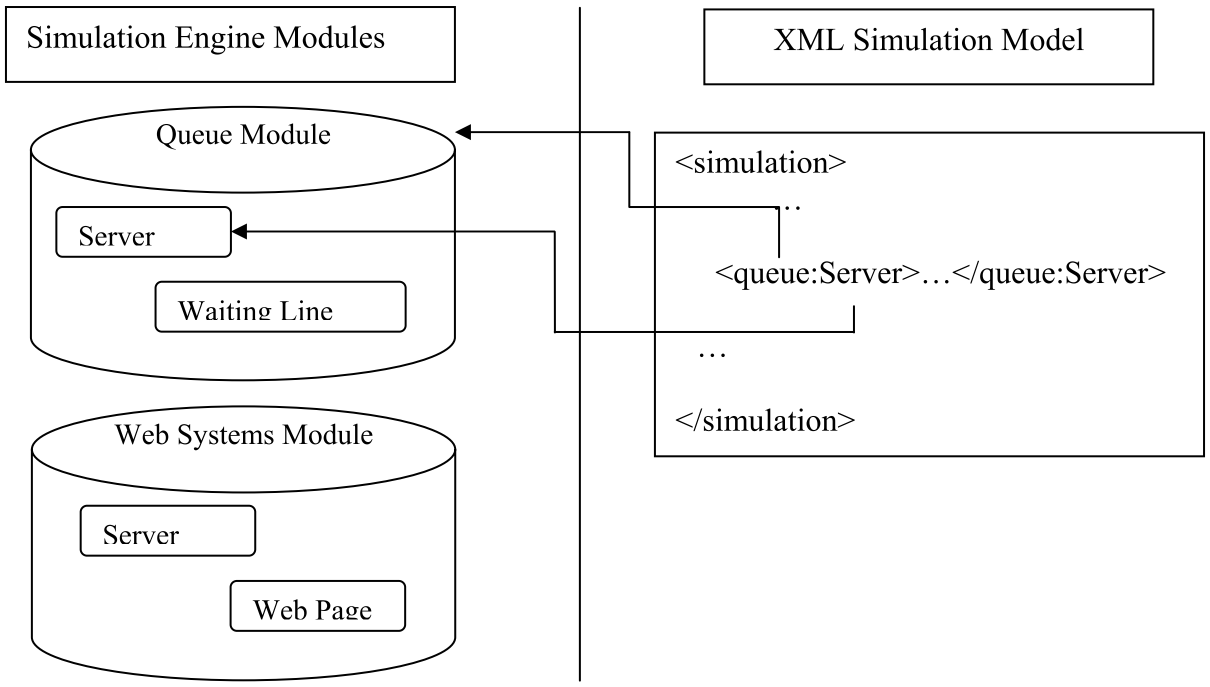


Figure 8. Referencing a Simulation Module Object from the XML Model

Tools in the system working on the XML simulation model should also consider how to handle new XML simulation elements. For example, creating an XSD description of the new simulation package and the appropriate XML-NS extends the simulation model representation validation process to include new element types. Adding an XSD description to the XML validation components of the system allows the system to appropriately validate the XML representation of the new XML simulation elements whenever processing and validating the XML representation of the simulation model. Using a series of system-wide XML validation tools lets the entire system take advantage of adding the new package XSD description to a single validator component. Also, adding the new package to the system should not affect other components in the system, since the other components reference either a specific XML representation and simulation object instance or a generic object representation and XML representations available in every simulation object XML description (common tags and attributes). Other tools can handle new XML elements in a similar manner by allowing dynamic configuration or easy addition of additional processing logic for new XML elements based on the XML-NS and element name.

Chapter 5: Asynchronous Visualization Protocol

In older web-based simulation (WBS) environments, most simulations run to completion and output the results with no user interaction. A user sets up a simulation model, starts the simulation, and expects results to be available at some point in the future when the simulation completes. With the advent of Java Applets and other web technology, users can connect to a WBS environment, create a simulation, and execute the simulation at any time they wish. The movement into web technology pushes simulation companies into the realm of Simulation Application Service Providing, or SIM-ASP, where companies host simulation servers and allow other companies or individuals to utilize the simulation services for commercial purposes [Wiedemann, 2001]. The problem with this is that users are still restricted by the paradigm of expecting results at a point in the future as opposed to a real-time display of simulation results, which is crucial for today's impatient web-user. Web-based simulation concurrent visualization methods, such as animation displaying the state of simulation objects, are still in their formative years, but several techniques have been proposed to navigate the major problem inherent for concurrent visualization on the web, the large and variable network latency.

Latency in concurrent display of simulation visualization during model execution over a wide area network, or WAN, such as the Internet, causes havoc in user-simulator communication. Communication between server and client over the Internet, which is essential for concurrent visualization, is typically slow and unreliable. Researchers have been developing techniques and tools in an attempt to hide the latency issues for over a decade. Many of the older visualization techniques required little communication between the user and the server and offered little interactivity; the user simply uploaded a simulation model, and received a multimedia presentation as the output. By not allowing large quantities of communication between user and server, the latency issues did not compromise the performance of the system. As new technologies have matured, researchers have proposed methods of connecting to web-based simulators using synchronous connections and custom display technology, such as Java Applets or streaming-multimedia presentations, in an attempt to mask the latency. These methods offered more interactivity by allowing the user to communicate with the simulation server and allowing the user to alter simulation state as the simulation progressed. The problem with a concurrent display using synchronous connections is that the traditional, standards-based web-technologies (like TCP/IP and HTML) were not built to support such a synchronous connection. Because the communication protocols do not rely on traditionally available communication libraries, the user is forced to download execution intensive runtime environments, i.e. the Java Runtime Environment, or JRE, in order to utilize the simulation application. The web is inherently asynchronous. Relying on custom JRE technologies decreases the availability of the environment since Java is not available to the common user. Also, synchronous connections do not scale because they consume a constant amount of network and operating system resources to maintain the connection, even when the connection is not in use. This in turn causes problems as the number of users increase.

The sections below discuss the creation of an asynchronous communication protocol for the WebQS3 web-based simulation system relying on traditional communication protocols available in every user's browser. Section 4.1 outlines several other techniques that have been proposed for web-based simulation and shows how the field has grown as Internet technologies have matured. Section 4.2 presents the design of a synchronous and asynchronous communication protocols for concurrent visualization. Then, in Section 4.3, performance results from a simulation are provided as a means of comparing the two protocols.

5.1 Methods for Web-Based Simulation Visualization

There are three basic approaches to simulation visualization (SV) on the web as detailed by Lorenz, Dorwarth, and Ritter [1997]. The first approach, known as *remote SV*, relies on the remote server to execute the simulation and create an animation output, which is then sent to the client. The second approach is *local SV* where the client downloads both the simulation and the visualization components on the users' local computer. The third approach is a hybrid between the first two approaches, *remote simulation/local visualization*, in which the simulation is executed remotely and sends visualization commands to a visualization engine on the client. The three methods are explained in more detail in the following sections.

5.1.1 Remote Simulation Visualization

The remote SV approach follows the traditional web paradigm of request and response over a HTTP (form) post, and can be considered a job request on a batch processing system. As shown in Figure 9, the user specifies values of parameters for a simulation model by filling in an HTML form. The form parameters are submitted to a web server through a Common Gateway Interface (CGI) script or similar server-side technology where the parameters are interpreted and the simulator is started. Once the simulation has finished, the CGI script returns the results to the user. These results differ depending upon the simulation software being used, varying from a simple HTML page with tabled data and embedded graphics [Schumann 1997] to a complex dynamically generated multimedia animation using a technology such as Flash, Windows media, or a 3D VRML model [Ritter 1997].

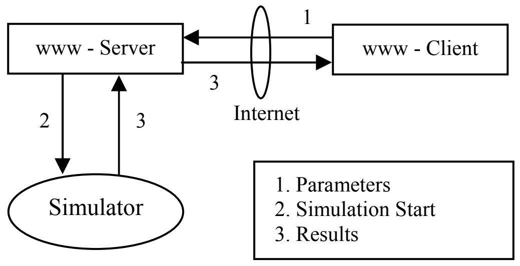


Figure 9. Remote Simulation Model

The remote SV approach works well for adapting existing simulation software products, such as Arena, to a web-based environment. The web server simply acts as an adapter to the use of the simulation system exposing the simulator functionality to the remote users. The added benefit of a centralized application exists, making maintenance easier for developers. Remote SV does not function well for observing dynamic processes at work or allowing the user to interrupt a running simulation. While there is no explicit communication between the user and the simulator during the simulation execution, a problem arises with the latency of long running simulations as well as the server load under a large number of users. The simulation results are only available to the user after the simulation has completed, or at specific predetermined points in time, for example, every 2 minutes. Long running simulations may take an extended time to generate output, especially under heavy server load. This results in causing the user to wait long periods of time or having the client timeout by waiting too long for a server response. The server can easily be overloaded if all of the responsibilities for the simulation are placed on the server for several concurrent user simulations. Complex visualizations are notoriously processor intensive to generate, and when combined with the simulation execution, cause a heavy server load even with a small number of users accessing the simulation server.

5.1.2 Local Simulation Visualization

As client-side technology advanced, more responsibility over the simulation visualization could be shifted to the client. Figure 10 shows how local SV allows a user to connect to a web server and download a simulation to run on his or her own computer. The simulation begins on the client machine and the visualization rendered. This approach shifts the responsibility for execution completely from the server to the client, making the server a centralized distribution point for the simulation, but performing no real work. In order to avoid most cross-platform issues, Java is the implementation technology for the majority of local SV simulation engines, and hence reliance on the availability of an appropriate JRE. Several researchers have developed toolsets such as OOCSMP and SimJava to facilitate local SV web-based simulation. OOCSMP [de Lara and Alfonseca 2001] is a simulation language that generates Java applets as output for display to users. SimJava [Howell and McNabb 2003] is a simulation engine, which includes a package for developing visualization applets on top executing simulations. These types of toolsets offer a dynamic alternative to the simple form parameter upload seen in the remote SV approach provided earlier.

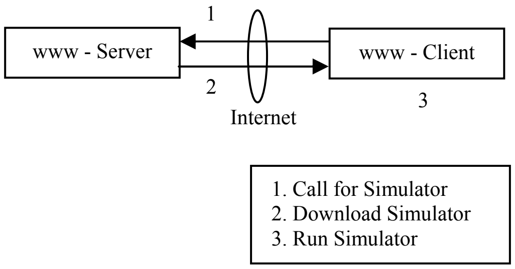


Figure 10. Local Simulation Model

The local SV approach lends to more user interaction and animation as the latency between user and simulator is reduced to nothing. A shortcoming to this approach is that it relies on the power of the client to execute the simulations efficiently. Some users may lack the appropriate hardware in order to run the simulations in a timely manner, causing much frustration on the user's part. Also, many of the existing toolsets simply allow visualizations of predetermined simulations, limiting flexibility in simulation parameters. For example, OOCSMP only allows simulations to be altered through compiler options, thus not impeding parameterized use by average users.

5.1.3 Remote Simulation / Local Visualization

By combining the approaches of remote simulation and local visualization, we can create a hybrid approach that yields the benefits of both. The approach, first introduced by Berger and Leiner [1997], utilizes a Java server and Java Applets to display the animation component to the user. As shown in Figure 11, the simulation engine runs remotely behind a web server. When the user connects to the server, a visualization engine is downloaded to the client. Once the visualization engine has completed loading on the client, a dedicated data connection is established back to the server. The results of the simulation are transferred to the client over the connection allowing the visualization engine to display results to the user in a dynamic nature. The data can change continuously, delayed only by the executing simulation model and the latency present on the network connection.

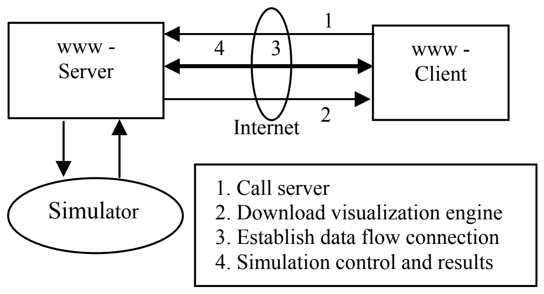

www - Server
www - Client
1
2
3
4
Internet
Simulator
1. Call server
2. Download visualization engine
3. Establish data flow connection
4. Simulation control and results

Figure 11. Remote Simulation/Local Visualization Model

The hybrid approach combines the best features of remote SV and local SV. The majority of the simulation execution is performed on the server. This allows the simulation to take advantage of more powerful hardware and eases maintenance with a centralized application. The visualization portion of the simulation is sent to the client, reducing the total workload on the server and allowing more user interaction than traditional remote SV approaches. The user interaction can be detected on the client and sent back to the server through the dedicated connection. The drawbacks of the model are twofold. First and foremost: the latency issue. The communication between client and server is delayed by the network latency, which may be several seconds on a WAN. Any user interaction is usually delayed by a few seconds as the communication is returned to the server. Many of the generic simulation servers on the market today circumvent the issue of latency by offering no interactivity whatsoever. The second issue is the use of standards. Many of the systems developed using this approach, such as the system developed by Gan et al. [2001], utilize Java Applets through a browser interface. The user must have the specific JRE installed in order to use the application, not a certainty on the Internet today. Additionally, the application uses non-standard communication protocols for client-server communication causing the user to rely on the JRE exclusively for communication.

5.2 Protocols for Web-Based Simulation Concurrent Visualization

As shown in the previous section, simulation on the web has advanced from a display of results of a simulation to an interactive application as the front-end for a web accessible simulator. The interactivity of web-based simulation has blossomed due to communication protocols that allow for the user to communicate with the server and vice versa. A dilemma with all of the systems under development currently is the reliance on the JRE for user interaction and communication. By moving to a strictly standards-based protocol, the reliance on the JRE can be eliminated thus allowing any user with a browser to utilize the application. This section will provide descriptions of the communication protocols used in remote simulation/local visualization applications, which will act as a basis for comparison of the two protocols later.

Initially, a system model is developed and described thereby establishing a framework for evaluating the various protocols. Subsequently, because almost all modern concurrent visualization tools are based on synchronous protocols, a synchronous communication protocol is described. The synchronous protocol relies upon add-on technology, such as custom plug-ins or Java applets, to function. As an alternative, an asynchronous protocol is provided.

The asynchronous protocol relies on traditional web technologies, such as the hypertext transfer protocol (HTTP), to function.

5.2.1 System Model

Most modern web-based simulation applications follow the remote simulation/ local visualization model. The model decomposes into four basic components, as shown in Figure 12, closely following the client-server (producer/consumer) model. The *visualizer* represents the animation engine on the client, while the *simulation server* resides on the server and houses the simulation engine. The simulation server and the visualizer are connected by a *communication medium*, a variable latency network. This network allows *visualization messages*, defined by a processing time and a byte size, to be sent from the simulation server to the visualizer.

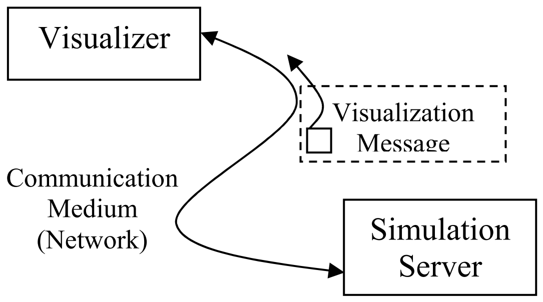


Figure 12. Web-Based Simulation Model

The simulation server creates visualization messages that reflect the events occurring in the simulation. The visualization messages are transmitted over the network to the visualizer for processing and time-stamped with the simulation time at which they occur. The visualizer executes the messages in time-stamped order as appropriate and creates a visualization output display for the user. The simulation is stopped when an end-of-simulation message is generated and executed by the visualizer.

Given that the visualizer always needs visualization messages in order to execute, it is assumed that the simulation server executes faster than the visualizer. This does not mean that all visualization messages arrive at the visualizer in time stamp order. The variable network latency means that some messages may be out of order, especially in cases in which the messages are sent over the network in quick succession. The messages are buffered on the client side, and put into time-stamp order as they arrive. If a message arrives with a time-stamp prior to the current simulation time at the visualizer, the system enters an invalid state and stops execution.

5.2.2 Synchronous Protocol

A synchronous communication protocol establishes a dedicated connection between the visualizer and the simulation server that allows for data to be sent between the two processes at any time. When the user downloads or launches the visualization tool, the visualizer contacts the simulation server and establishes a connection that lives for the extent of the communication session between the two processes. This connection is then used to pipe information to the visualizer in a greedy fashion, i.e., as soon as the visualization message is generated it is placed on the network for delivery to the visualizer. Using this type of communication protocol puts the server in control of flow of data, while the visualizer simply acts as a receiver buffer for data. Therefore, the client is required to have enough memory to buffer all of the visualization messages it receives.

As the visualizer receives messages from the network, the messages are placed into a time-ordered queue based on the simulation execution time of the message. After the first message is received, the visualizer begins retrieving messages from the front of the queue and executing the messages at the appropriate time-stamp until an end-of-simulation message is received. When the simulation server has completed execution, an end-of-simulation message is sent to the visualizer. Once this occurs, the simulation stops operation, permitting server resources to be allocated to another simulation for a new user.

5.2.3 Asynchronous Protocol

An asynchronous communication protocol operates differently from a synchronous protocol in that it first puts the visualizer in control of the flow of data. As the simulation runs, visualization messages are created and buffered in timestamp order on the simulation server. The visualizer requests a certain number of visualization messages from the server when data is required. As the server receives requests for data, the server sends a batch of the requested number of messages over the network to the visualizer as a single large message; if the requested number of messages is not available, the server simply sends the number of messages present in the buffer. When the visualizer receives a batch of messages, the messages are added to the message queue for execution and are executed in a similar manner to the visualizer in the synchronous protocol.

The issue of control is paramount in this protocol. By allowing the visualizer to control the flow of data, both the client and the server can optimize performance based runtime system parameters. To illustrate, if the client can process only a few visualization messages at a time, only a few messages need to be requested from the server. Likewise, if the client can handle more message buffering, more can be requested. Allowing the visualizer to request a certain number of messages helps to ease the visualizer message buffer management (e.g., help prevent buffer overflows from too many messages arriving in a short period of time).

A disadvantage of the asynchronous protocol is that the latency between the server and the visualizer can possibly be doubled for receiving messages from the server. For every data packet sent to the visualizer, a request must be sent to the server. The extra request latency is capable of causing delays in the visualizer's startup time as messages are requested and sent from the server. The delay in the start of visualization may anger those users who expect immediate results. Also, the message request algorithm needs to be adaptive to estimate a schedule for requesting data in order to ensure that buffer underflows will not occur at the visualizer and cause pauses in the simulation. If the visualizer does not request messages frequently enough, the visualizer will run out of messages to execute and possibly enter an invalid state. If the visualizer requests messages too often, the resources needed to buffer the messages may exceed the limit of the client system. The message request algorithm must adapt to the visualization situation to optimize the resource usage of the client.

5.3 Evaluation and Comparison of Protocols using Simulation

In order to provide a means of comparison for the two communication protocols, a series of simulations were executed and statistics were collected for several quality characteristics of the running systems. These simulations were performed using a custom-made simulation system. This system was developed by using the Java programming language and by following the process-interaction conceptual framework of simulation. Each component of the WBS system model, described in section 5.2.1, is instantiated as an object with its own flow of control and set of operations to perform. The objects communicate via an abstract message-passing interface, which allows for easy construction of a variety of simulations via configuration files. Objects can be inserted into the simulation both at compile-time and run-time, permitting the same simulation environment to execute simulations for both synchronous and asynchronous communication protocols with minimum changes to the simulation setup.

Simulations are assembled at runtime using a set of parameters in a configuration file, which is loaded via a command line argument at startup. When the configuration file is loaded, the simulation system parses the simulation parameters and assembles the specified objects into a complete simulation. The configuration file also allows the user to specify parameters for individual objects included in the simulation. For example, changing the random variate parameters, e.g., mean network latency, allows the simulator to quickly and easily adjust for different simulation setups. Once the simulation is assembled, the simulation runs and statistics are collected and output for analysis by a software tool such as Microsoft Excel.

5.3.1 Simulation Setup

Two separate model representations were created using the simulation system: one model for an asynchronous communication WBS system and one model for a synchronous communication WBS system. Both simulation models used the same set of random variate parameters for generating simulation data in an effort to achieve the same set of circumstances in both models. Using the same set of parameters also provides an accurate comparison of the communication protocols alone. Both simulations were constructed at runtime using custom developed objects and statistics collectors initiated from separate configuration files. The simulations were run ten times to gain a more statistically valid picture of the quality characteristics being measured.

The simulations were run using a set of four randomly generated values for various object parameters created throughout the simulation runs. The simulation servers generated visualization messages at a time determined by normal distribution, with a mean of 300ms and a standard deviation of 40ms. The simulation messages generated with a simulation execution time determined by a normal distribution with an average of 1200ms after the last message was set to execute. These distributions were chosen in an attempt to avoid buffer underflows on the visualizer and measure more of the effect of the protocols on the network. If the server generated many messages with similar execution times the visualizer would be taxed more than the network, owing to the message storage and execution requirements imposed on the visualizer. The visualization message size was determined using a uniform variate between 30 and 60 bytes. The distribution for message size is based on the XML simulation message format for the WebQS3 system developed later in this thesis, assuming a 1 byte - 1 character encoding method is used.

The network latency was set to be 3000ms on average with a standard deviation of 1000 ms, determined by a normal distribution. This distribution was chosen so that the messages would be generated at a much quicker rate than the transport rate, inherently making the visualizer lag the simulation server by a wide margin. It should also be noted that the asynchronous protocol used the simple algorithm of "request up to 50 messages from the server when only 15 messages are left in the message buffer" to request visualization messages from the server. This simple algorithm worked extremely well given the existing simulation conditions and allowed an ample number of messages to be sent to the visualizer at any time.

Using the setup described, the simulations collected a variety of statistics for all portions of the web-based system model. The following sections detail the results of the statistics collection for the two communication models. Specifically, three important statistics are compared between the systems: Average network bandwidth usage, simulation server execution time, and visualizer execution time. These statistics illustrate the tradeoff involved between the two protocols.

5.3.2 Network Bandwidth Usage

Figure 13 presents the average network bandwidth usage of the two protocols. As the figure shows, the synchronous protocol uses almost quadruple the bandwidth of the asynchronous protocol (428 bytes/s vs. 112 bytes/s) simply for transporting the individual messages. This does not include the overhead required for maintaining the dedicated live connection between server and visualizer. That the synchronous protocol requires more bandwidth is not surprising, as the synchronous protocol sends a message over the network whenever a message is generated while the asynchronous protocol waits for a request. The greedy nature of the synchronous protocol places more responsibility upon the network for handling a large number of messages at a time. Conversely, the asynchronous protocol places fewer messages on the network; this allows the visualizer to control the data flow as needed, relying less on network resources.

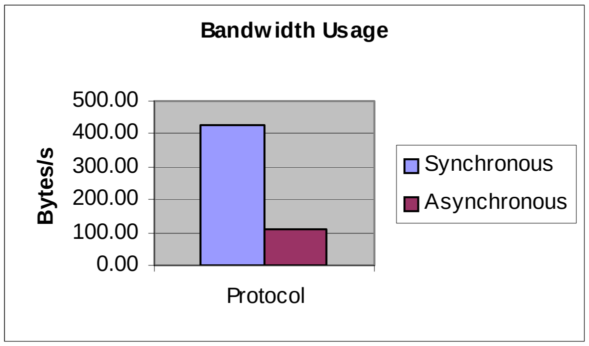


Figure 13. Bandwidth Usage

Looking at this statistic, there is cause for concern regarding the synchronous protocol on two counts: network usage and network failure. In environments where network usage is in high demand or available only with limited bandwidth, the asynchronous protocol should have heightened performance when given its reduced reliance on network resources. The possibly of network failure causes further problems in synchronous protocol systems. If the network fails and the system has to recover, the server will have to re-send all messages in existence on the network at the time of failure. To illustrate, almost 4 times as many messages have to be recovered after a failure under a synchronous protocol than an asynchronous. In environments where network failure is a possibility, the server will be required to buffer messages that are being sent over the network. The server will perform this function until an acknowledge (ACK) message is received. This means that server resources will have to be devoted to maintaining the non-ACKed message buffer, taxing the synchronous server more than the asynchronous. The network efficiency gives a great advantage to the asynchronous protocol in terms of communication performance.

5.3.3 Simulation Server Execution Time

Figure 14 shows execution time statistics for the simulation server under the two protocols. The figure shows that the simulation servers end message generation at nearly the same time, yet the asynchronous server lives for almost four times as long as the synchronous server. This is due to the fact that the asynchronous server continues to live through the buffering of the visualization messages even after the server has completed generating messages to send. The synchronous server uses fewer server resources under this model, allowing more resources to be devoted to other users once the simulation has completed execution.

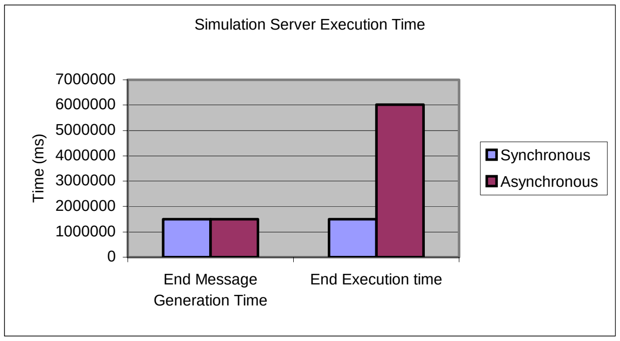


Figure 14. Simulation Server Execution Time

The synchronous server naturally provides faster execution time as the server shuts down once its message generation is complete. The asynchronous server must stay active until the visualizer nears display completion in order to buffer the generated visualization messages. After examining the issue more closely, however, it is evident that the asynchronous server is only active as a memory buffer for the outgoing visualization messages. This means that the simulation server utilizes negligible processing power; instead, it chiefly uses memory resources or persistent storage resources. Memory and persistent storage are the most plentiful resource on modern web servers and can be considered almost inexhaustible with respect to executing processes. In terms of processing power the two protocols are comparable, but the asynchronous server has storage requirements much exceeding that of a synchronous server.

5.3.4 Visualizer Execution Time

As Figure 15 illustrates, the visualizer is active for nearly the same amount of time under both protocols. The figure validates the assumption that the synchronous server stops execution long before the visualizer completes operation. The visualizer takes four times longer to execute than the server under the synchronous protocol. If the server ends operation before the visualizer, it poses no problem until you consider user interactivity messages that may be sent from the visualizer back to the server at random intervals. If the visualizer made any requests back to the server (e.g., requesting a simulator stop

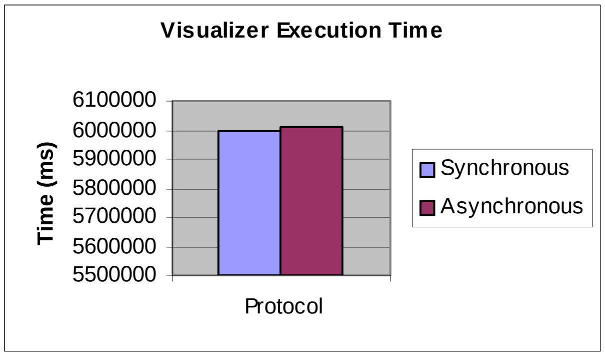


Figure 15. Visualizer Execution Time

and parameter change), the synchronous simulator would have to reactivate to perform further calculations. Reactivating the simulation server after de-allocation of all its resources is execution inefficient, especially if the server resources are in contention between several other simulation servers already in execution. Taking this point into consideration, the asynchronous server is in a better position to handle these types of random user communications. The asynchronous server is already active and has some knowledge of where the visualizer is in the display of the simulation through the length of the outgoing message buffer.

When concurrent interactivity is considered in a variable latency environment, utilizing an optimistic simulation engine must be considered. Fujimoto [2000] describes such an engine, where the simulation can execute events as soon as possible while still considering events that may occur in the past. A simple interactive WBS can be thought of as a distributed simulation system with 2 nodes. One node represents the optimistic simulation engine on the server and the other node represents the visualizer component on the client. The two nodes communicate under the optimistic simulation protocol to facilitate interactive simulation. For example, a user sends an event to the server at visualizer time *t*. The server, operating faster than the visualizer, is at time *t* + 1000. The server must therefore rollback the system to time *t* in order to handle the event that occurred in what it perceives as the past. Fujimoto states that this system rollback can be performed through anti messages. These anti messages provide a means to track events that have occurred in the simulation. When a rollback is needed, the simulation engine cancels the events that have executed in reverse time order up to a designated place in time. Utilizing an optimistic protocol, the server must maintain a rollback message buffer to handle user events. An advantage of the message buffer under the asynchronous protocol is that the buffer can double as a rollback message buffer. The messages being stored for output to the visualizer can have associated anti-messages to use during a rollback. Under the synchronous protocol, no message buffer is available, thus making it harder to perform the rollback without extending the system. Also, since the synchronous server has more messages flowing over the network at any time than the asynchronous, extra resources must be devoted to maintaining messages already on the network. The synchronous server needs to be extended to receive messages from the visualizer, providing notification of the current visualizer time. The server may then remove messages from the rollback buffer up to the designated visualizer time and free up resources for other activities.

The asynchronous server buffering also acts as a clue to aid in performance optimization for the server. For example, if the buffer of messages to the visualizer becomes extensive, the simulation server can infer that the visualizer will take a lengthy time to empty the buffer. This allows the server to context switch out and pause simulation execution while concurrently permitting other server processes to run. Pausing the simulation execution also allows the visualizer to pace the execution speed of the server, thus keeping the simulation times of both the visualizer and server nearly synchronized. Synchronizing simulation times on visualizer and server is important when the interactivity of the simulation engine comes into consideration. For example, the visualizer calls for a change in the simulation parameters at time *t*. With a simulation server that simply executes until complete, the simulation server may have already completed execution (at time *t* + 5000). Thus, the server is required to reinitialize and backup in the simulation to time *t* in order to alter the simulation and recreate the changed visualization. This is opposed to the situation where the simulation server pauses at time *t* + 300 when the visualization buffer is nearly full, and waits for the visualizer to catch up. A simulation server that pauses is better able to move backward in the simulation execution than the simulation that does not pause.

Analyzing the visualizer execution time in combination with other factors, it can be seen that the asynchronous server has the ability to keep better pace with the visualizer. Hence, a more susceptible environment to user interactivity is made available. An asynchronous protocol provides benefits to moving towards an optimistic simulation server without relying on visualizer resources. The asynchronous server, using hints from the visualizer, has the ability to better control its execution than a synchronous server. When considering interactivity and performance in relation to the visualizer, the asynchronous communication protocol provides greater rewards for the client system, the server system, and the interactivity possibilities of the simulation.

5.3.5 Conclusions on the Comparative Evaluation

Though the synchronous protocol provides a faster server execution time, much of this execution time reduction is due to the eager communication inherent in the protocol. The model protocol is constructed with the assumption that the visualizer will perform no communication requests back to the server after the server has completed operation. Bearing in mind that the asynchronous protocol utilizes fewer network resources, is more adaptable to interactive simulation, follows web-standards based protocols, and maintains processing requirements similar to that of synchronous protocols, the conclusion can be made that an asynchronous protocol is a suitable replacement for current synchronous protocol systems. Following the conclusions made from the comparative evaluation, the WebQS3 system will use an asynchronous communication protocol to communicate between the visualization and simulation execution components.

Chapter 6: WebQS3 Design and Implementation

Several previous attempts to create a web-based simulation (WBS) environment have approached extending existing simulation systems, such as Arena, with a simple web-based front-end for increased availability to end users [Wiedemann 2001]. The simulation environment serves as the simulator component with a web server component attached to the front of the system. This allows users to provide simulation inputs, and it in turn delivers simulation output to web users. The two components traditionally communicate through the use of CGI scripts and APIs defined on the simulation system. Creating a system in this manner unnecessarily couples the front-end display to the back-end server, so that when one component is upgraded or replaced, the other components must be altered to match. A tightly coupled system reduces the maintainability and extensibility of the overall system. Special care has been taken to implement the WebQS3 system to avoid un-necessary coupling and create a simulation platform that is both efficient and suited to web-enabled execution.

The WebQS3 system has been engineered utilizing pure web-standards implementation following a component based development and a tiered architecture. Every component of the system is decoupled from every other component of the system, relying on only a given XML-based communication protocol for passing data into the component. This type of architecture creates an environment conducive to distribution across many computers and high scalability as far as concurrent users are concerned. Each piece of the architecture, including design decisions, is discussed in detail in the following sections. The sections are ordered following a trace through the system from start of model development to model execution. The progression of a model specification through the system follows a set path when executing the two major functional flows of the system.

6.1 Hardware and Software Environment

The WebQS3 server-system is a Java-based enterprise application implemented using the J2EE 1.3 specification. WebQS3 utilizes WebSphere application server (WAS) version 5.1 as an Enterprise JavaBean (EJB) and web container, providing application management services as well as web-communication facilities over HTTP/HTTPS. The model and user information are stored in a server-based DB2 Universal Database which is a Relational Database Management System (RDBMS). DB2 oversees the persistence of WebQS3 information in memory and to the file system. WebSphere and DB2 run on a 2-processor IBM xSeries 225 Server using dual 2.4 GHz Intel Xeon processors. The server contains 1.5GB of memory and 145 GB of disk space running Windows Server 2000.

The WebQS3 client-system is available to any SVG 1.0 enabled web browser (Microsoft Internet Explorer, Netscape Navigator, or Apple Safari). The preferred client browser is Microsoft Internet Explorer (MS-IE) version 6.0 [Microsoft 2004b] with the Adobe SVG Viewer (ASV) Plug-in version 3.0 [Adobe 2004] included. The client code was developed using standard SVG and DOM 2 [DOM2 2004] scripting methods, but there were not sufficient resources to test the implementation on browsers besides MS-IE 6. The test configuration of the client-system varied as many different systems participated in the system testing.

6.2 WebQS3 Architectural Design

The WebQS3 architecture, as shown in Figure 16, encompasses a family of nine separate components. Each component pairs with a partner component, creating a tier in the architecture. Each tier contains two components, one to handle the input and one to handle the output communication of the tier; each tier represents a piece of the J2EE architecture. Table 9 details the breakdown of the various tiers and components. The tiers are numbered in the diagram and correspond to a tier number in the table. As the table shows, no WebQS3 tier corresponds directly to the data-mapping tier, as the architecture does not strictly require data mapping components. WebQS3 uses the database mainly to store user information and XML model representations across sessions, not for simulation execution. Extending the architecture to use the database during simulation execution may necessitate extensive use of the database, thus the architecture allows for the extension by inserting a data-mapping tier where appropriate.

Table 9. Architecture Component Tier Assignment

Tier Number	J2EE Tier	WebQS3 Tier	Component
1	Data Tier	Data Tier	Database
2	Data Mapping Tier		
3	Business Tier	Simulator Tier	Model Generator, Simulation Manager
		XML Handler Tier	XML Schema Validator, Message Manager
4	Controller Tier	Communication Tier	Interaction Controller, Message Buffer
5	Client Tier	Client Tier	Model Composer, Visualization Manager

The WebQS3 system decomposes into two basic operations: composing a simulation model and executing a simulation model. Each operation represents a data flow through a series of components into the database tier (system input) or data flow from the database tier (system output). During each data flow, a single architecture component in each tier takes responsibility for handling the communication to and from the tier. Figure 16 presents a diagram of the overall architecture. The placement of the individual components in the diagram represents the how the communication is handled by each tier. Components located on the upper portion of the diagram represent input communication handlers while components on the bottom portion of the diagram represent output communication handlers. Discussion of the data flow handling details follows.

Composing a simulation represents a system input flow as the simulation model is composed and stored in the database. The user composes a simulation model using the *Model Composer*. The composer submits the model to the *Interaction Controller*, which decides where to direct the uploaded simulation model. Complete simulation models are forwarded to the *XML Schema validator*, ensuring the correct format of uploaded models. The *Model Generator* accepts valid XML model representations and converts the XML model representation to an executable simulation instance. Either the XML simulation representation or the executable instance may be stored in the database for retrieval and execution at a later time.

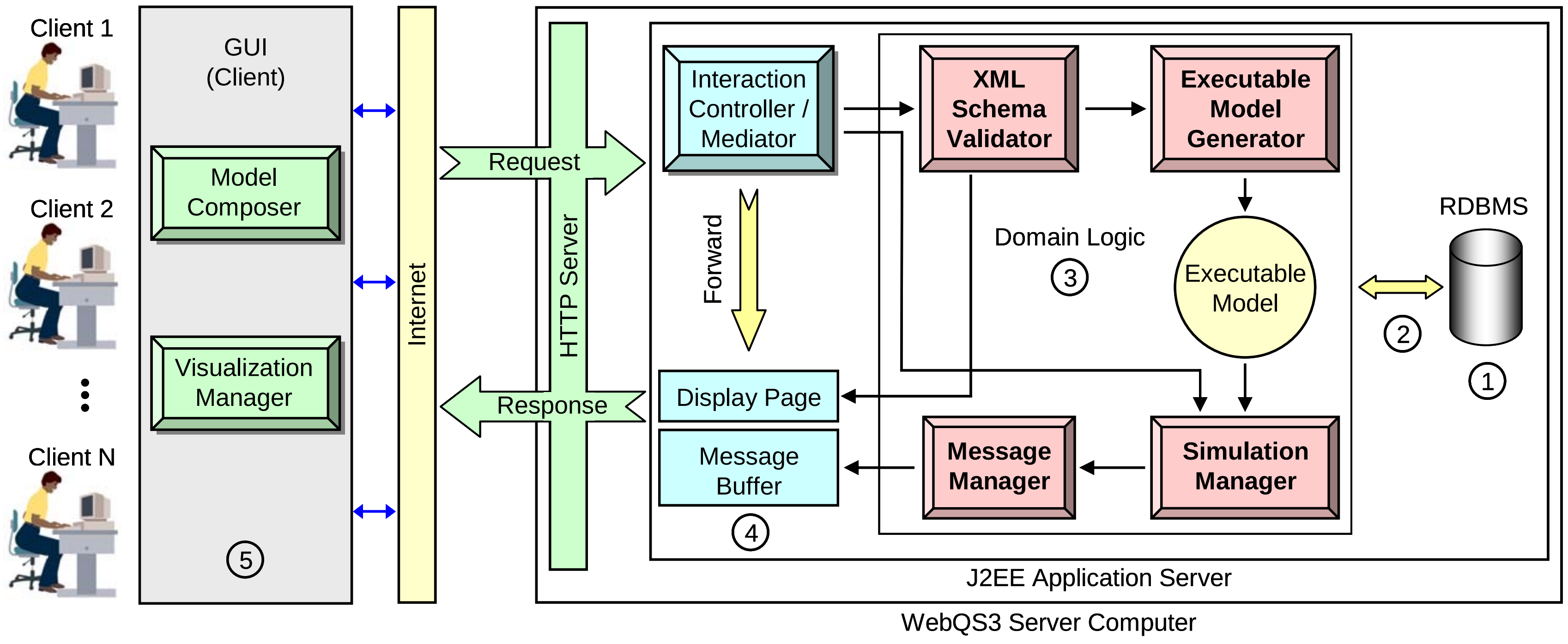


Figure 16. WebQS3 Architecture

Executing a simulation model entails a system output data flow. Firstly, the *Simulation Manger* locates a simulation model in the model database, or possibly a model that has been uploaded and stored in memory, and prepares the simulation for execution. The manager is also responsible for managing the execution of the simulation over the simulation's lifetime. During execution of the simulation, the manager emits XML formatted simulation messages to listeners in the XML Handling tier of the system. The *Message Manager* collects the XML messages emitted by the Simulation Manager and performs a series of operations on the messages as needed. Once the message manager handles the individual simulation messages, the messages are forwarded to the *Message Buffer,* where the messages are queued up to await retrieval by the client tier. The *Visualization Manager* queries the Message Buffer looking for available simulation messages. When messages become available in the buffer, the Visualization Manager downloads the messages to render a visualization for the user.

The two system data flows described encompass the major functionality of the WebQS3 system. The system assigns each component a specific purpose and responsibility, forming a complete system when all of the components are combined. The sections that follow detail the specific design responsibilities and design issues that arise in the design and implementation of the individual components.

6.3 WebQS3 Model Composer

The model composer, or simply the *composer*, allows a user to visually construct a simulation model in a What-you-see-is-what-you-get (WYSIWYG) manner using an SVG enabled web-browser. At the present time the composer only offers queuing system components for simulation model composition, but the composer may be extended to offer addition simulation components using extensibility mechanisms built into the system. WYSIWYG composition of simulation models, or simply *models*, depends on the concept of simulation composability, a developing idea that states that simulation components can be connected to each other using simple component-based engineering techniques. Model development does not have to be an arduous process of developing and maintaining packages of specialized simulation components. Instead, model developers can use a series of readily available, extensible, and customizable components when needed. Davis et al. [Davis 2000] provides a more in depth discussion of composability.

The use of a web-browser for model creation increases the accessibility of the simulation system by allowing any user with a browser access to model creation facilities. Traditional simulation environments, of the type mentioned in the introduction, do not allow visual composition of models. Instead, the environments rely on simulation programmers to construct and connect the needed simulation objects in for simulation execution. Simulation users cannot change model execution parameters. The users are simply allowed to run the simulation and receive results. If any changes are required, the model users must contact the programmers to make the change, or rely on the programmers to include a means to change model execution parameters. This reliance adds unnecessary complexity to the simulation development process, especially when creating simple models where component developers need not participate. Allowing users access to simple model composition facilities makes the application more accessible to users who do not have access to, or the resources to hire, simulation programmers. It also facilitates use of simulation models to a group of users who may not have considered simulation a viable and cost efficient option. Simulation programmers can focus on the development of executable simulation components and component descriptions for use by the simulation composers instead of splitting time between simulation development and component development. The sections that follow describe the implementation and design issues encountered in developing the composer tool. Also included is a description of the XML model format and creation process followed by the composer.

6.3.1 Composer Implementation Technologies

The composer was implemented using web-standard technology, available on any fully functional SVG enabled browser. If a user attempts to access the composer using a browser that is not fully functional then some of the composer functionality will not be available to the user. The composer provides a simple WYSIWYG interface allowing direct manipulation and construction of models on the client. Following the Model-View-Controller (MVC) paradigm of interface development [Sun 2004b, Alur, Crupi, and Malks 2001], the composer uses SVG to display a visual representation of the simulation model to a model developer (the view). When a user interacts with the model using a mouse or keyboard an EMCAScript event is triggered (the controller). The events are sent to the appropriate business logic handlers and update an XML representation of the simulation model to reflect the changes made by the user interaction (the MVC model). Every interaction by a user updates the XML simulation model representation as well as the SVG view displayed to the user. The XML simulation model representation may be uploaded to the server for storage or execution when the model composition is complete.

6.3.2 Provided Simulation Components

By default the composer provides a module of queue components to use in composition of queuing systems simulation models. The queuing module holds four reusable simulation model components: a generator, a First-In First-Out queue, a server, and an exit point. The components provided can be individually configured and reused by a simulation model developer and composed into a queuing system simulation.

When a component is selected the *component toolbar* appears. As shown in Figure 17, the toolbar contains three buttons to perform operations on the selected component. The first button, the “X” button, removes the selected component from the simulation model. The “P” button expands the component *properties panel*. The component properties panel displays the set of configurable properties available on the selected component. Many of the properties represent random variate streams that are configurable to a given variate generator type such as exponential or normal random variate distributions. Each variate generator contains a different set of configuration properties. For example, an exponential variate generator requires only a mean value while a uniform variate generator requires an upper and lower bound value. The configurable properties of individual component types vary. The “S” button expands the component *statistics panel*, which sets the statistics to be collected for the component during simulation execution. The statistics available for collection also vary by component type.

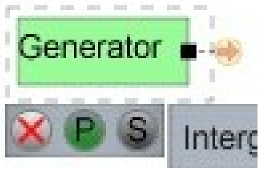


Figure 17. Component Toolbar

The final configurable property of a component is the *next place*. The next place represents the next component that queued items in the queuing system model should move to when the current component has completed processing the item. Configuring the next place involves holding down the left mouse button on top of the arrow icon to the right of the component. The arrow can then be dragged on top of the next component in the system and the left mouse button released. When a connection between components has been established a line is formed between the components with the arrow pointing towards the next place in the direction of movement of the queue system items. An example of the process is shown in Figure 18. Multiple next place components may be configured by dragging lines between separate components multiple times. A next place component may not be pointed to twice. Individual component types may place restrictions on the type of next place that may be set. The components may also provide *probabilistic branching to next place*. The probabilistic branching simply allows the model developer to provide a probability as a parameter in determining what the next place should be as Figure 19 shows. The probabilities must add up to a value of 1.0 when all the next place component paths are considered. Probabilistic branching overrides the default behavior of the next place decision making process. A connection between components may be removed by grabbing an arrow at the end on a connected line and dragging the arrow to white space in the background of the simulation model display. The line between components disappears when the next place connection is broken. The following sections details the individual component capabilities, limitations, and configurable properties and statistics.

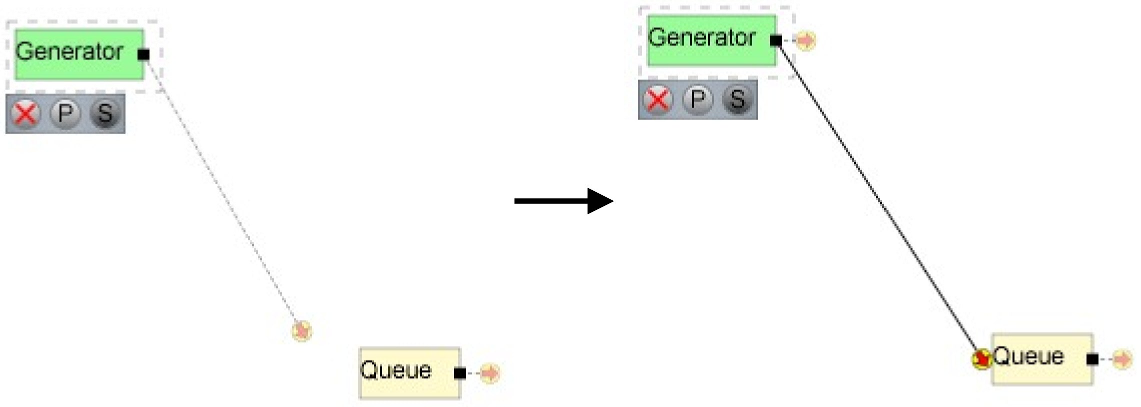


Figure 18. Next Place Component Configuration

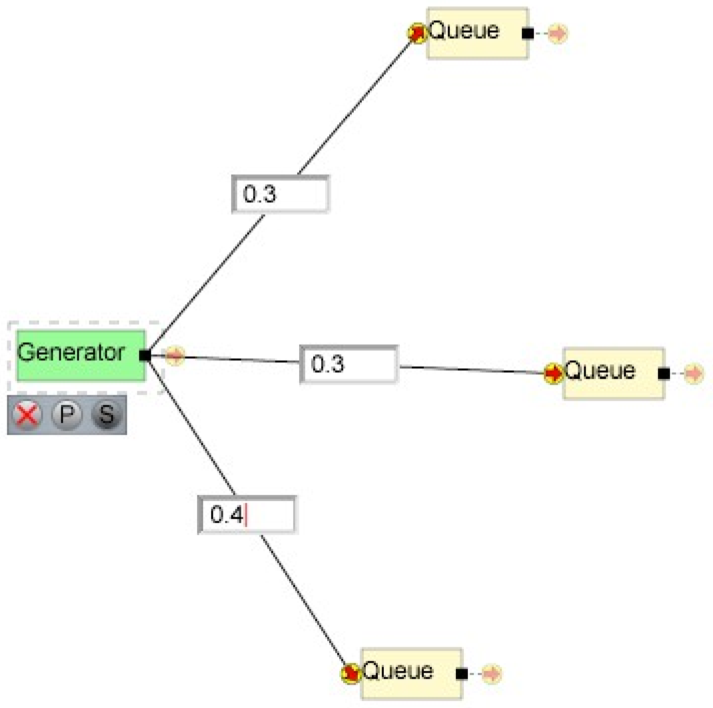
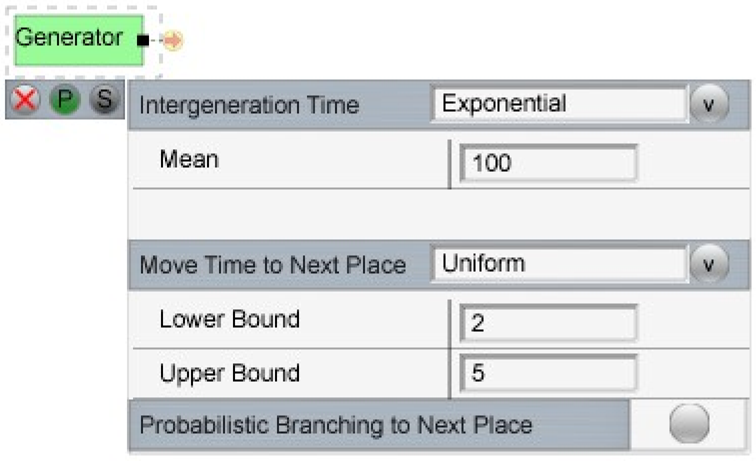


Figure 19. Probabilistic Branching Configuration

6.3.2.1 Generator Component

The *generator* component represents a component that generates items to be queued in the queuing system. Shown in Figure 20, the generator creates items based on the configurable *intergeneration time* random variate generator parameter. The intergeneration time variate generator generates a random variate that represents the number of time units that should elapse before an item is generated and sent to the next place in the queuing system. The default algorithm for determining the next place component is *shortest-queue-first*. When an item is generated by the generator the next place queue with the smallest number of items in the queue is chosen as the next place. The item is then sent to the next place using a generated travel time. The *Move Time to Next Place* random variate stream parameter determines the travel time between the generator and the next component in the system. The generator can point to *n* next places but is restricted to pointing only to queue components. It is possible to configure probabilistic branching to next place with the generator component. The generator component provides no statistics for collection during simulation model execution.

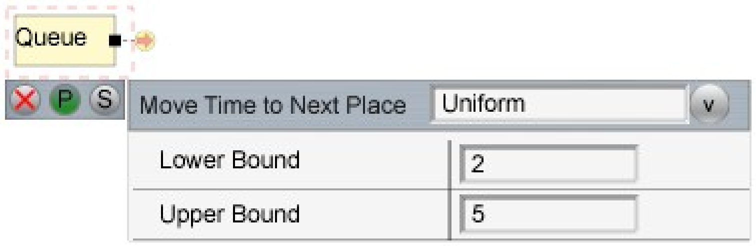


Intergeneration Time	Exponential
Mean	100
Move Time to Next Place	Uniform
Lower Bound	2
Upper Bound	5
Probabilistic Branching to Next Place	

Figure 20. Generator Component Configurable Properties

6.3.2.2 First-In First-Out Queue Component

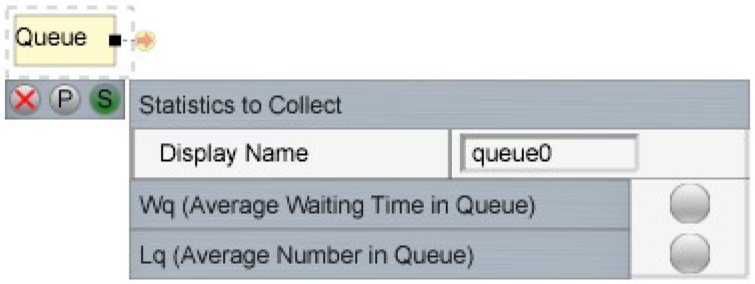
The *queue* component represents a component that queues items in a first-in first-out manner in the queuing system. Shown in Figure 21, the queue component only allows configuration of the *Move Time to Next Place* random variate generator. The Move Time to Next Place random variate generator parameter determines the travel time between the queue and the next component in the system. The queue can point to *n* next places but is restricted to pointing only to server components. The default algorithm for determining the next place is *idle-server*. When a server is put into idle state the server send out a message to waiting queues. A queue then attempts to reserve the server. If the reservation is successful, then a queue item is sent to the server using the generated travel time. If the reservation is unsuccessful then the queue continues to wait until notified of an idle server. A queue component may not use probabilistic branching to a next place as the probability may dictate that a queue item be sent to a working server, thus putting the system into an illegal state.



Move Time to Next Place	Uniform
Lower Bound	2
Upper Bound	5

Figure 21. Queue Component Configurable Properties

As Figure 22 shows, the queue component allows collection of 2 different statistics by clicking the check box next to the statistic identifier. First, the simulation system can collect information on the *average waiting time in the queue* (Wq). The system tracks the time queue items enter the queue and when the items exit the queue and determine the average time the queue item spent in the queue component. Secondly, the system can collect information on the *average number of items in the queue (L_q)*. The system monitors when items enter and exit the queue and tracks the number of items present in the queue component at any given time. The final element with regards to statistics collection is the *display name*. The display name provides a means to identify the component easily when the statistics collection report is presented at the completion of simulation execution.

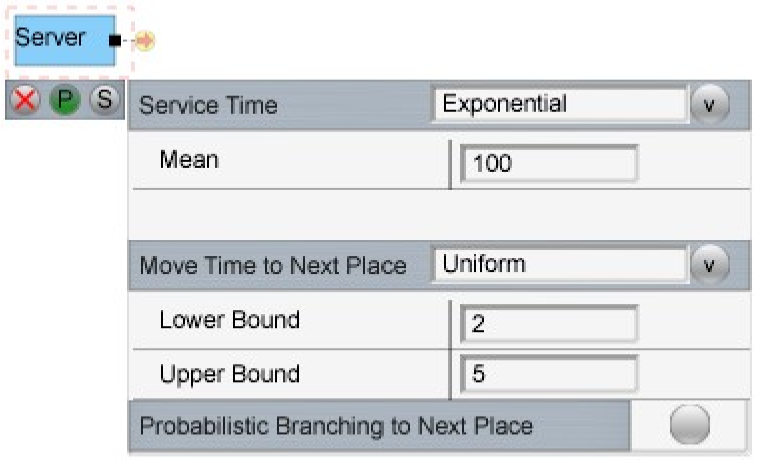

Queue

Statistics to Collect	
Display Name	queue0
Wq (Average Waiting Time in Queue)	
Lq (Average Number in Queue)	

Figure 22. Queue Component Configurable Statistics Collection

6.3.2.3 Server Component

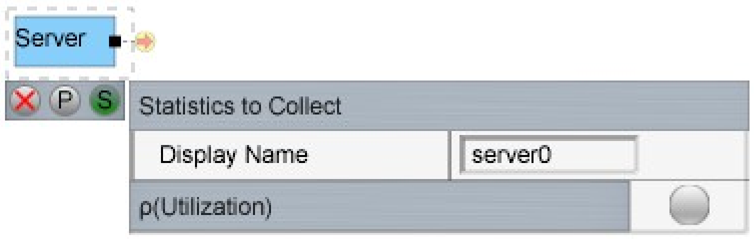
The *server* component represents a component that provides a processing element in the queuing system. Shown in Figure 23, the server processes queue item based on the configurable *service time* random variate generator parameter. The service time variate generator generates a random variate that represents the number of time units that should elapse while a queue item is being "processed" by the server. When the service time has elapsed, the item sent to the next place in the queuing system and the server moves into an idle state. The item is then sent to the next place using the travel time generated by the *Move Time to Next Place* random variate generator. The server can point to *n* next places but is restricted to pointing to queue components or exit points. If pointing to both an exit point and a queue component probabilistic branching **must** be used.


Server

Service Time	Exponential
Mean	100
Move Time to Next Place	Uniform
Lower Bound	2
Upper Bound	5
Probabilistic Branching to Next Place	

Figure 23. Server Component Configurable Properties

As Figure 27 shows, the queue component allows collection of a single statistic. The simulation system can collect information on the *utilization* (ρ). The system tracks the percentage of time that the server is in the "working" state and determines the percentage of overall system time spent working. The server also allows setting a *display name* to provide a means to identify the component easily when the statistics collection report is presented at the completion of simulation execution.



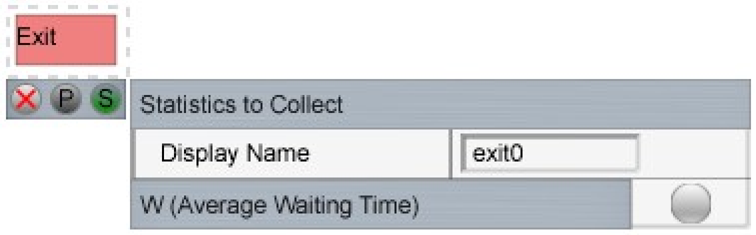
Statistics to Collect

Display Name	server0
p(Utilization)	●

Figure 24. Server Component Configurable Statistics Collection

6.3.2.4 Exit Point Component

The final component type is the *exit point*. The exit point represents an exit from the queuing system. When a queue item reaches the exit point, the item is removed from the system and statistic collection on the item ceases. The exit point provides no configurable properties but provides statistic collection options as shown in Figure 25. The system can collect the *average waiting time in the system* (W) for objects that pass through the individual exit points. Say that a simulation model has two exit points and a complicated set of queuing conditions. One exit points exists for an early exit from the queuing system when a system is congested. The other exit point exists at the end of the complex set of queues. Statistics can be collected on the exit point at the end of the queuing system as opposed to the entire simulation system. Collecting statistics on the single exit point allows better understanding of the time spent moving through the complex set of queues. Collecting statistics on the entire system would sway the statistics toward the items that left the system early driving the statistic report towards a lower time than actual. The exit point also allows setting a *display name* to provide a means to identify the component easily when the statistics collection report is presented at the completion of simulation execution.



Statistics to Collect

Display Name	exit0
W (Average Waiting Time)	●

Figure 25. Exit Point Component Configurable Statistics Collection

6.3.3 SVG Interface Development Issues

The composer makes extensive use of SVG 1.0. SVG controls all aspects of display, from the images of simulation components to the rendering of textbox input elements. The original intension of SVG 1.0 was for use as a simple vector graphics creation language, not as an interface design tool. Several facilities were added to the SVG specification during the specification development, such as user interactivity, that made SVG an attractive option for interface development. However, creation of complex interfaces remains difficult using the available specifications.

The use of other technologies, such as XForms, would considerably aid in the implementation and performance of the composer interface, yet XForms is an extremely new technology from the W3C (specification released October 2003) and many SVG tools do not include an XForms implementation. Additionally, SVG 1.0 lacks mechanisms to include XForms content in SVG interfaces, so any XForms inclusion in an application would be a non-standard extension to the SVG tool. Because XForms implementation is not available on the SVG 1.0 platform, the decision was made to use a custom SVG widget toolkit to render user input elements (textboxes, numeric inputs, checkboxes, buttons, etc). Using a custom SVG widget toolkit brings about a series of problems when dealing with SVG 1.0 systems. Many of the problems are addressed in the SVG 1.2 [SVG 2004b] specification, but the specification is still in development and few tools implement the draft specification. The tools that do implement the SVG 1.2 specification are extremely buggy and not suitable to producing commercial systems. Discussion of the major SVG interface development issues follow.

6.3.3.1 SVG as an Interface Description Language

SVG 1.0 facilitates the creation of beautiful user interfaces, since the technology creates vector graphics for display to users. SVG 1.0 can render almost any graphical design imaginable through a combination of animation events, graphic filters, and geometric shape rendering. The SVG 1.0 interactivity module allows users to interact with the graphical elements of the interface and to change the interface content based on user events. While SVG 1.0 offers all of the facilities required for interface creation, the process of creating the interfaces is extremely complex and time consuming, especially when adding complex functionality that requires interaction between elements. The SVG language implements interface design at an extremely low level, making it difficult for inexperienced developers to easily create SVG interfaces. A few examples are included below to illustrate the difficulty of interface development.

Every rendered SVG element must be positioned exactly on the screen. While absolute positioning of content is fine for static graphical display, the addition of interactivity and dynamic inclusion of new content becomes more cumbersome. For example, altering and re-rendering a vertical list of text elements requires the interface designer to implement the rendering logic in script. When the new text element is added to the list the vertical position of every element in the list must be recalculated. The absolute position of the new content and every piece of existing content in the list and on the screen has to be determined by the designers' script, which must be thoroughly tested. A common facility such as recalculating list element position would be simpler and more efficient to implement in the SVG tool than in relying on the interface designer to implement the logic. XHTML has relied on the CSS box model [CSS 2004] to facilitate easy development of interface content positioning. The CSS box model provides a consistent view of relating element content position in XHTML documents via CSS properties and a display type attribute attached to every XHTML element. The display type of the element tells the box model exactly how to render the XHTML element in relation to other content in the document, encouraging a quick interface development cycle. CSS box model positioning allows trouble-free addition of dynamic interface content and styling in contrast to the SVG method of pixel perfect placement of every element.

A second problem with SVG as related to interface creation is the concept of z-index. When SVG content visually overlaps the SVG tool must determine what content to display on top, as both pieces of content cannot be displayed at the same time. Currently, SVG determines the top content by overwriting content encountered early in the XML tree with content later in the XML tree. This means that the only way of showing content located early in the XML tree is to dynamically move the content to a location later in the tree and re-render the interface. While dynamically altering the tree is not always a problem, there are cases when the order of the XML tree is important and the tree content cannot be altered. Including a z-index property on SVG content would allow the interface designer to specify a simple property that the SVG system could check to determine which element to show. For example, specifying a z-index value between 0 and 100, where elements with higher z-index values are shown on top of elements with lower z-index values. Including a z-index property relieves the interface designer from working directly with the SVG tree and unnecessarily altering tree order and content.

A third issue is the event specification and handling of SVG 1.0. When SVG 1.0 was originally formulated, the specification writers decided to rely on other W3C specifications to assist in writing SVG. One of the specifications cited to be included in SVG 1.0 is DOM 2.0-Events, a sub-specification of DOM 2.0. DOM 2.0-Events provide an event model and series of default events, encouraging consistent implementations across all types of web-applications. A problem with the DOM 2.0-Events specification is that it is incomplete; this is so noted in the specification document. The Events specification allows no consideration for keyboard input, as the group working on the specification could not agree on how the input events should be handled. Note that keyboard input events are subsequently added in DOM 3.0-Events. Thus, SVG has no standard facilities to handle keyboard input, a must in some applications. Many SVG tools follow XHTML's lead and implement an older version of keyboard events, to appease developers until the DOM 3.0-Events specification is eventually released. A difficulty in using the old keyboard events is that the event model is inconsistent and non-uniform. Different vendors implement the keyboard detection events in different ways, do not offer access to series of keystroke combinations, or implement detection methods differently in different tool versions. For example, ASV 3.0 does not fire an event for the arrow keys (up, down, left, and right) while ASV 6.0 does fire events on those keyboard combinations, but only when detecting the events on certain content. SVG 1.2 and DOM 3.0-Events takes steps to correct the deficiencies of the SVG 1.0 event model, but current developers using SVG 1.0 must find appropriate workarounds.

While it is possible to use SVG 1.0 as an interface description language, the technology is not specifically intended to encourage interface development. Many of the requirements of the SVG language elements are realized at a very low level, making interface development complex and prone to error. Also, many facilities designed to encourage interface development are not available in current tools. Couple the low-level nature of the language with the incompleteness of many of the interface development facilities, and the development of more complex and interactive interface applications becomes extremely difficult.

SVG 1.2 seeks to rectify many of SVG 1.0's problems and move SVG to a true interface description language. The SVG 1.2 specification draft includes mechanisms for including outside XML content such as XHTML and XForms directly in SVG documents in a more extensible way, allowing developers to use facilities available in other specifications that are hard to implement directly in SVG. For instance, writing a table of values is simple in XHTML but quite complex in SVG. This is because pixel specific positioning is required for each table cell. SVG 1.0 offers the ability to insert outside content in SVG through the "foreign-object" element, which is not implemented in many mainstream SVG viewers. SVG 1.2 expands on the **`<foreign-object/>`** tag, allowing element content to be placed in various other places in the SVG document. SVG 1.2 also includes a technology for creating custom content in SVG documents. Designers can also specify and create their own set of XML tags to include in any SVG document. The SVG working group is considering the inclusion of a technology called the eXtensible Binding Language (XBL), where custom tags not only render as content but include an object-level programming interface with methods and events. Such technology would allow script level interaction with the custom content in an object-oriented manner. The inclusion of these two technologies creates an environment in which interface creation is much easier than in earlier versions of SVG.

6.3.3.2 Custom Widget Toolkit

Using a custom widget toolkit as opposed to an operating system implemented toolkit creates several problems, namely performance and usability considerations. The performance of the system lags whenever an input component is rendered because that the toolkit has to be implemented using ECMAScript and available SVG elements. As an example, a custom toolkit drop-down box with 3 options has to create and render over 60 SVG elements and set over 320 element attributes, not including the referencing of scripts to setup event handling. The time required to render the widgets is extremely noticeable in slower systems, especially when creating a large quantity of input elements.

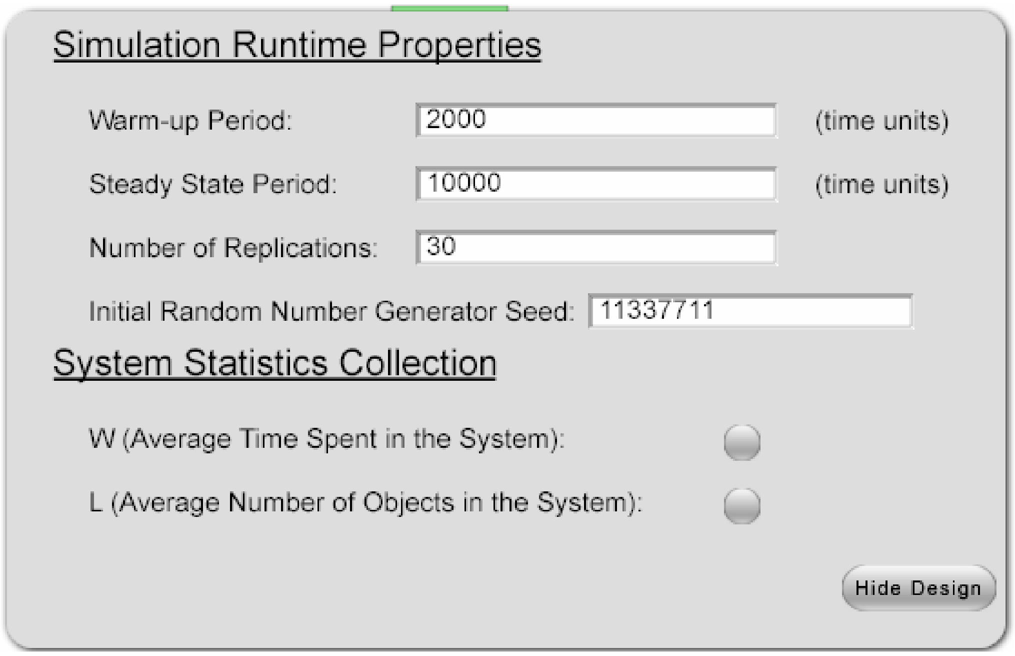
As utilizing a custom widget toolkit is the lone option for SVG 1.0 systems needing to accept user input, there are few other options to explore. The interface designer may rework the interface replacing complex input elements with other options, or work around the rendering time of the input widgets. The composer takes the second approach to working with the toolkit by implementing a simple threading system in ECMAScript to allow the widgets to render in the background while interface updates are displayed to the user. ECMAScript does not have direct threading libraries available for use, but does include a method for deferred execution and queuing the execution requests. The simple threading system breaks the threaded execution operation down into a series of short-lived functions. The library then schedules the short-lived functions to execute in the near future, allowing the threaded operation to complete over several function executions. Breaking up a long running process into smaller operations hides the lag of the input component rendering from the user, letting the composer appear functional. However, it delays the rendering of the new widget on the screen. The use of the custom widget toolkit aids in allowing quick design time based on the interface design specification, but requires workarounds for several performance issues as the interface design relies heavily on the use of input widgets for altering XML simulation model content.

6.3.4 XML Simulation Model Composition

The XML simulation model is divided into four sections, with different functional areas of the tool accessing different parts of the XML model representation. Each of the four sections of the XML document configures a different component of the WebQS3 system during the two data flows of the overall system. The *header* section contains meta-information related to the document and simulation runtime parameters (e.g., how long a simulation should run). The *model components* section contains the description of the simulation model and the properties set on the individual simulation model components. The *views* section of the document contains the SVG elements associated with each model component. Finally, the s*tatistics collection* section holds information regarding which components statistics to collect. An XSD description of the model representation format is provided in Appendix A with a sample model described using the format included in Appendix B.

6.3.4.1 Header Section

The header section of the XML model representation appears first in the document order contained under the **<head>** XML element. The header holds meta-information related to the model, as well as simulation runtime parameters. To illustrate, the header holds a title element that is used to set the title of the simulation when the SVG version of the model is displayed. The header also contains runtime properties settable via the *Simulation Runtime Panel*, shown in Figure 26. The properties set on the panel dictate how long a simulation will run and various properties regarding the system level statistic collection. When an input box on this panel is altered, the change is propagated down to the underlying XML model representation. If any of the properties are set to blank inputs then the composer removes the XML element representation of the property from the underlying model, relying on the default system value to be set instead of a value from the XML model.


Simulation Runtime Properties

Warm-up Period: 2000 (time units)

Steady State Period: 10000 (time units)

Number of Replications: 30

Initial Random Number Generator Seed: 11337711

System Statistics Collection

W (Average Time Spent in the System):

L (Average Number of Objects in the System):

Figure 26. Simulation Runtime Panel

The other set of header properties are available under the *Model Accessibility Panel*, shown in Figure 27. The model accessibility parameters dictate how users of the system are allowed to access the simulation in the database. The simulation title doubles as the title of the simulation page and the name displayed when users search for available simulations. Setting a model as "public" allows any user who visits the system to view and run the simulation model using the simulation visualizer, regardless if the user is a registered user or not. Setting the model as publicly editable allows any registered user to access and edit the simulation. The model only allows editing; the original creator of the simulation still maintains ownership of the model.

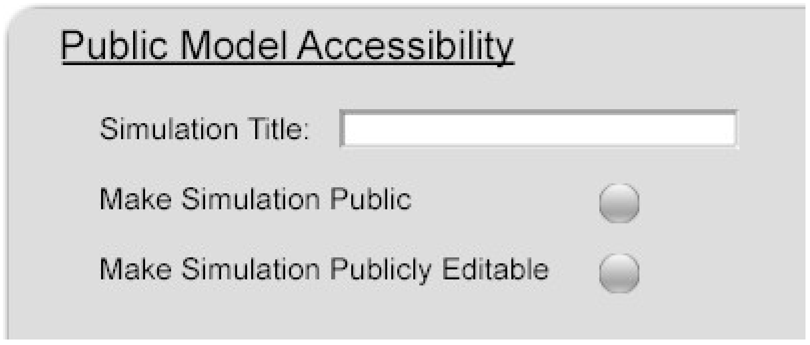
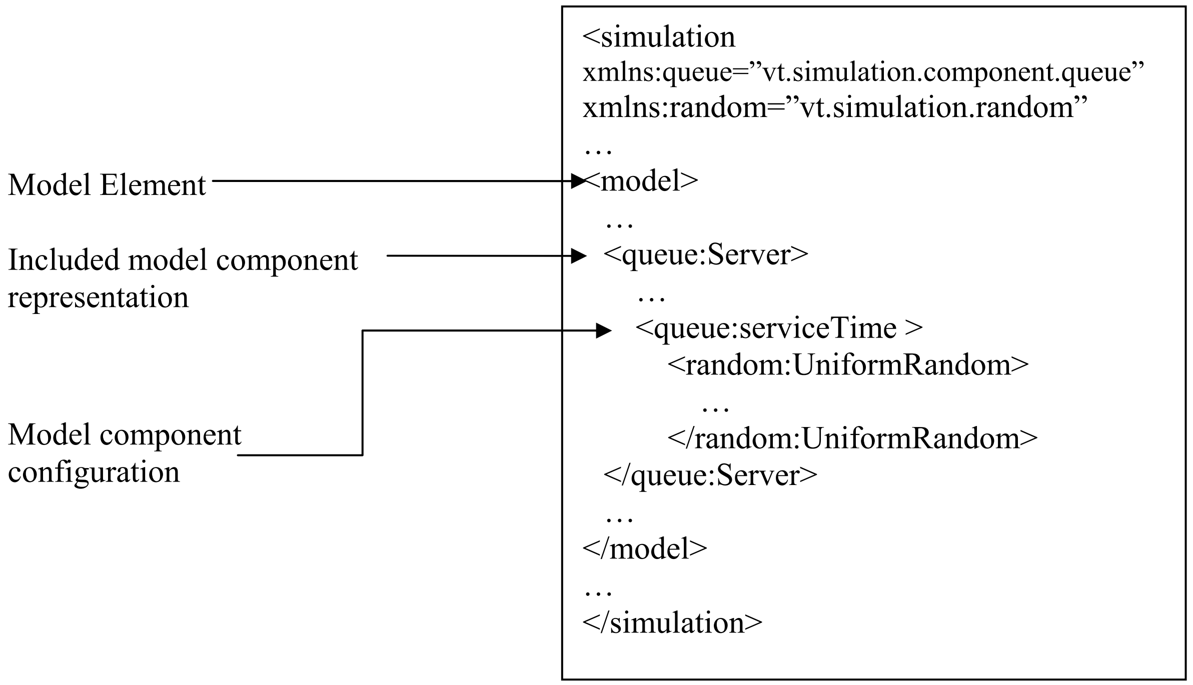


Figure 27. Simulation Availability Panel

Extending the header section of the XML document is a trivial task. The XML schema describing the content of the header section allows any new XML element to be added to the header section, as long as the new element is not in the same namespace as the rest of the header section. For instance, a user would like to add a timestamp to the document to track when the create date of the model and the last modification time. Adding a **`<timestamp />`** element with any content or attributes to the header is allowed by default and maintained by the composer tool whenever changes are made to the XML.

6.3.4.2 Model Components Section

The model components section of the XML model representation holds the description of the individual simulation components that compose the simulation model under the **`<model>`** XML element. Every model component included and configured in the simulation model has the model component configuration stored as a set of XML elements in the XML model representation. The direct child elements of the **`<model>`** XML element represent the component and component type of the simulation model elements. Children of the component XML representations declare the property values for various component configuration properties. Figure 28 illustrates the basic structure of the model components section of the XML model representation. The structure of the XML model components is described in the XML Model Generator section later.


Model Element
Included model component representation
Model component configuration

```
<simulation
xmlns:queue=”vt.simulation.component.queue”
xmlns:random=”vt.simulation.random”
…
<model>
  …
  <queue:Server>
     …
     <queue:serviceTime >
        <random:UniformRandom>
           …
        </random:UniformRandom>
  </queue:Server>
  …
</model>
…
</simulation>
```

Figure 28. XML Model Representation Format Example

Every XML model component included and configured in the simulation model is assigned a unique `id` attribute when the composer creates the simulation component. The `id` attribute represents a means of uniquely identifying every simulation component in the simulation and the `id` is used when setting up connections between simulation elements. When a connection is established, a `<basic:nextPlace/>` element is added to the simulation component with a `ref` attribute whose value is equal to the `id` of the connected component. For example, if a `Generator` component establishes a connection with a `Queue` component with unique `id` “queue1”, then an element equivalent to `<basic:nextPlace ref=”queue1”/>` would be added to the `Generator` component XML component representation element to represent the connection formed.

All the model component configuration properties available under the properties dialog are also reflected in the XML representation of each component. When the values of the configuration properties are altered in the composer interface, the values of the properties in the XML model representation are changed to match. Changing the XML immediately makes submitting the model to the database later a simple operation. The XML representation already exists and is simply validated to ensure that no errors exist before sending the XML model to the server for execution or storage.

The XML model representation should also facilitate reloading into the composer so that stored models can be changed and updated as needed. When the composer loads an existing XML model representation, the composer walks the component structure of the XML `<model>` element tree to extract needed information to re-create the visualization of the simulation model. For instance, while walking the XML tree the composer encounters a `<queue:Server/>` element. A `<queue:Server/>` element represents a `Server` component in the composer, which is known to have the properties: moveTimeToNextPlace, nextPlaceSpecification, and serviceTime. The composer walks the children of the **`<queue:Server/>`** element and finds the information associated with those properties (in XML elements). The composer then populates the display properly to facilitate editing by the user. Any missing values are either marked as an error or filled with the default value, as determined by the individual missing property. The composer also investigates the next place connections included in the XML component representation and forms visual connections between the component elements in the composer interface.

The model component section of the XML model representation is extensible through the use of the XSD *substitutionGroup* property. The substitutionGroup property designates that a particular element can be replaced with any element that is a part of the substitutionGroup that is named at the point. Thus, by setting the substitutionGroup property to an abstract base class, any element extending that base class can be plugged in at the component level. The model section defines a **`component`** base class (requiring an **`id`** attribute), which can then be extended allowing new components to be added to the model at any time.

6.3.4.3 Views Section

The views section of the XML model specification format holds the visual representations of the simulation model components. The **`<views>`** element acts as a container for the section in the XML model representation. The visualization of the simulation components is completely disparate from the actual component information to facilitate reuse of the model components and the view information independently. Individual views can be associated with any model component and vice-versa. The views section contains a series of **`view`** child elements. Each view element has a **`for`** attribute that references a simulation component **`id`**, setting up the mapping between view and representation.

The view element is extensible to representations besides XML. For example, a new series of view elements could be created in XHTML, MATHML, or any other XML dialect as a representation. The view element has no restrictions on its content other than being XML, allowing new view types to be plugged in as the functionality of the system is enhanced beyond SVG.

6.3.4.4 Statistics Collection Section

The statistics collection section is the final section of the XML model specification. The statistics collection information is contained within the **`<stat-collection>`** XML element of the model representation. As can be inferred, the section acts as a configuration setup for the statistics collection module. When configuring the statistics collector, the entire statistics collection section of the XML model specification is extracted and is handed to the collector component. Encapsulating the statistics configuration inside a single element instead of spread throughout the model XML components eases the configuration extraction, but adds logic to the XML representation validation process. Each simulation component has to specify what valid statistics are for the individual components.

The **`<stat-collection>`** element has two types of child. The first child is a special element type named **`<system-collect/>`** specifying system level statistic collection configuration. The other type of child is a **`<collect/>`** element, representing a request by the XML model to collect statistics for a particular element. The element has a **`for`** attribute, akin to the **`for`** attribute in the view section, referencing a particular simulation component to collect statistics for by **`id`**. The **`<collect/>`** and **`<system-collect/>`** elements contain **`<displayName/>`** and **`<statRequest/>`** children. The **`<displayName/>`** represents a display string to display to the user when statistics are displayed. Each **`<statRequest/>`** notifies the statistics collector to collect a particular set of strings and report the result at the end of simulation execution.

6.4 WebQS3 Communication Tier

The *Interaction Controller* and *Message Buffer* make up the *Communication Tier* of the WebQS3 system. Implemented using the Apache Struts [Apache 2004] web framework as a foundation, the Communication tier enforces the functional flow of user interaction through the system. Each user interaction with the server executes an **`Action`** class instance configured in the Struts toolkit. The Struts toolkit determines which **`Action`** class to run and execute a functional flow based on the particular Uniform Resource Locator (URL) the user calls on the web server. The Struts framework configures the **`Action`**-URL associations at runtime using mappings provided in an XML configuration file. Using an XML configuration file to set up interaction flow makes it easy to change the overall flow of the system, or add new functionality as needed without disturbing the existing system flow. The *Communication Tier* of the system uses many of the tools provided by the Struts toolkit, such as error messaging and user input parsing, to provide the functionality necessary for controlling user interaction flow with the system.

The *Interaction Controller* enforces certain event sequences or state conditions to occur before allowing users to perform certain operations. For example, some system functionality, such as saving a simulation model to the database, requires a user to login. The Interaction Controller may force the user to login to assign the user a valid user **`id`** before saving the model to the database. The Interaction Controller also parses and validates uploaded user input is complete. Incomplete information uploads return to the user, requiring the user to fill in missing information until the user provides all necessary information. The Interaction Controller ensures the completeness of information uploaded to the system from the client while directing interaction flow between client and server systems.

The *Message Buffer* represents a simple queue used to hold simulation messages before sending the messages to the simulation visualization manager. There are several possible implementations of the buffer, from a simple in-memory queue of string data to a Java Message Service (JMS) [JMS 2004] managed queue where the system takes great care to manage the queue length in relation to system resources. A managed queue ensures that the in-memory resource requirements do not exceed the memory availability of the system. In a long-running simulation, the capacity of the buffer may quickly overflow the memory availability of the server especially when running many concurrent simulations. The need to persist some of the messages to disk to conserve server memory for other tasks takes precedence over the fast access time of an in-memory queue. Persisting messages to memory frees memory resources for other applications, but increases the access time of time queue. Utilizing an existing managed queue system, such as JMS, or creating an adaptable persistence algorithm to monitor the queue access could aid in optimizing the queue performance. Using JMS or an adaptable persistence algorithm creates an environment where the visualization manager only accesses messages available in memory. Messages further back in the queue save to disk/move into memory during idle system time, such as when the visualization manger holds a large quantity of buffered messages on the client or the execution engine is idle.

6.5 WebQS3 XML Schema Validator

The XML schema validator (XSV) provides XML representation validation services for simulation models attempting to enter the simulation system. This component acts as a line of defense before incoming models are stored or converted to executable code by the system. XSV ensures that the model representation is valid and the components used in the model representation supported by the simulation manager. As Figure 29 shows, the validator accepts a complete XML formatted model representation, validates the model representation according to the appropriate schema specification, and either forwards the validated XML onto the next component, or returns a list of errors to the calling component. The XSV can be configured either at deployment time or execution time depending on the need of the system. The XSV component is also extendable; new simulation modules can be plugged in at will as long as a valid schema description is provided.

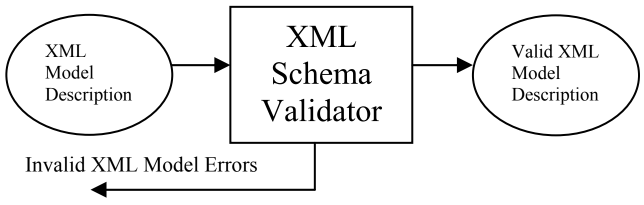


Figure 29. Basic Validator Component

6.5.1 Selecting Appropriate Validation Schemas

At either deployment time or execution time, the runtime system utilizing the XSV must provide a list of schema description files that fully describe valid simulation component representations. The validator conforms to the JARV [2004] validation framework: a vendor-neutral, implementation-independent interface for XML validators that allows for pluggable validator modules to be discovered at runtime. When the schema validator is handed a new schema to load, the schema can be formatted in any schema language (currently supporting DTDs, XSD, Relax NG [Relax NG 2004], and Schematron [Schematron 2004] schema descriptions). JARV allows the parser for the appropriate schema to be loaded and an executable version of a verifier for the schema description to be created (the schema description is transformed into an executable validation module). Providing an extensible system such as this allows component designers to describe valid XML representations of their components in the most efficient manner. When an XSD description will not describe a simulation component completely, the component designer can enhance the schema specification with Schematron, or use a Relax NG description, to describe conditions that cannot be easily described using the main schema language. For Example, the XSD schema description of the `Basic` (simulation component) module allows the XML model representation to describe the specification of a probabilistic branching to a next place. The XML model representation format calls for the declaration of a probability on the individual branches, which must sum to 1.0 when all the next place branches are taken into consideration. Providing cross-element validation (i.e., checking that sibling elements have probabilities that sum to 1.0) is impossible using an XSD Schema description. Therefore, the XSD Schema is enhanced with Schematron (which allows cross-element validation quite easily) in order to validate the model completely. Allowing the XML model representation to be described by any combination of schema description languages frees component developers from begin restricted to a single description language and allows for new schema description techniques to be easily plugged-in in the future.

The XSV loads the individual schemas provided by the runtime system and associates the loaded validation schema with a unique Uniform Resource Identifier (URI). The URI is very important to both the XML model specification and the schema validator, as the URI is how individual XML elements are matched to an appropriate schema or simulation module. Every XML element is associated with an XML-NS, and the XML-NS is associated with a unique URI for the XML document instance, quite simply associating the XML element to a particular unique URI. When the schema validator accepts a new XML model representation file for validation, it must first determine if the XML file is compatible with any of the loaded validation schemas. If the URI of the individual model representation elements match any of the schemas loaded into the validator, the validator knows that the particular element is supposed to be validated by that schema and performs the validation.

6.5.2 Performing the XML Model Representation Validation

XSV utilizes the SAX [SAX 2004] parsing technique to perform the parsing of the XML instance document for validation. SAX is a *push-parsing* technique where a XML element is consumed (in this case validated) as soon as the element is parsed. This method offers better performance than traditional parsers because the code awaiting the parser does not have to wait for then entire XML document to be parsed before examining the individual XML elements. Another advantage of utilizing SAX is that multiple *handlers* (element consumers) can be set up in a pipelined manner, as shown in Figure 30, which passes information about the element being parsed from handler to handler (filter to filter) and allows multiple handlers to be invoked during a single parsing session without having to re-parse the entire XML instance, which may be incredibly large.

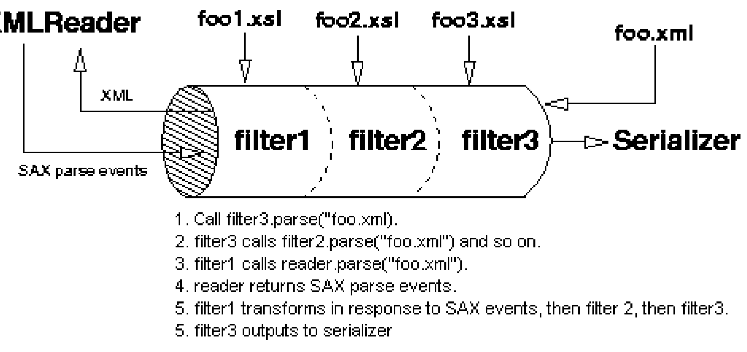

XMLReader
foo1.xsl
foo2.xsl
foo3.xsl
foo.xml
XML
SAX parse events
filter1
filter2
filter3
Serializer
1. Call filter3.parse("foo.xml").
2. filter3 calls filter2.parse("foo.xml") and so on.
3. filter1 calls reader.parse("foo.xml").
4. reader returns SAX parse events.
5. filter1 transforms in response to SAX events, then filter 2, then filter3.
5. filter3 outputs to serializer

Figure 30. Pipelined SAX Processing Example [Xalan 2004]

When the XSV accepts a XML model representation for validation, it first sets up a series of validation pipelines to handle the different available XML-NS types. XSV uses a pipelined validation technique because it does not know which individual schemas are needed until the individual elements of the XML model are consumed. Due to this fact, XSV must create all of the available schema description handlers and add them to the pipeline. When an element from the XML model is sent to the XSV, a special handler filter called the *delegator* intercepts the element as soon as XML validation starts. The delegator examines the XML-NS of the incoming element and determines the appropriate validation pipeline to utilize based on available schemas. Validation of element is then performed as seen fit by the individual pipeline. The entire process can be seen in Figure 31. As an example, the available `Queue` module performs checks against both an XSD description and a Schematron schema. When XSV is invoked, the XSD and Schematron schemas are loaded and formed into a queue validation pipeline. A validation pipeline encompasses a set of 2 handlers invoked one after another, since the validation needs to reference more than one schema description. For instance, an element arrives from the parser referencing the queue namespace, so the delegator sends the XML element to the queue validation pipeline. The element can be checked against both the XSD and Schematron without performing separate parses of the XML model representation (to validate elements against the XSD and Schematron schemas one at a time). As parsing the XML model representation is the most time consuming process associated with XML processing, performing a single parse for multiple element validation steps improves performance by orders of magnitude.

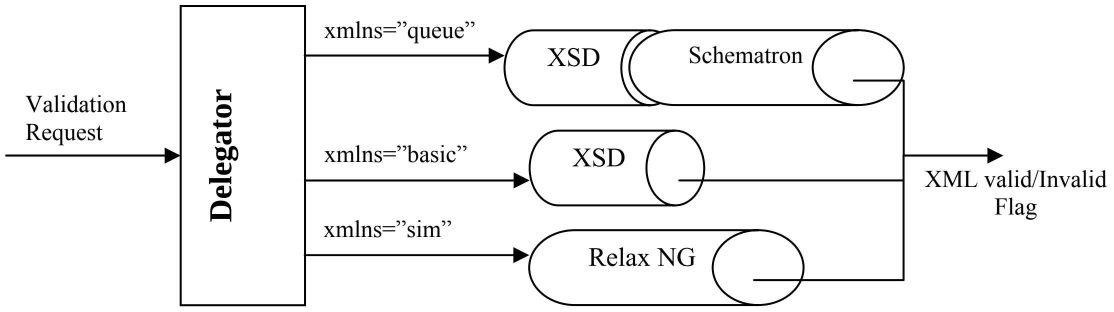


Figure 31. Pipelined XML Validation Processing

The individual validation handlers return a value of *valid* or *not valid,* indicating the validity of the XML element being parsed. If any of the elements in the XML model fail validation, the entire XML model fails validation. Currently, the individual validation schemas simply return the valid/invalid flag. The validator can be extended to return element specific validation error messages by utilizing technologies such as the XML-Schema infoset [XML Infoset 2004] or creating an appropriate Schematron meta-stylesheet to return usable error messages when validation fails. Performing the validation using a simple flag relies on the XML model creator to provide a valid XML model, or to understand where validation may possibly go wrong, whereupon it relies on the validator as a second check. Most commercially available validators are *fail-fast*, or stop validation on the first instance of an error. If there are many errors in a document, the validation needs to be performed numerous times before all the errors are detected. In an effort to streamline the process, the XSV is designed to return a simple flag instead of an error report. In the WebQS3, system errors can be detected much more efficiently in the model composer and caught before submitting the model to the server, thusly removing the need to perform a server submission while conserving network resources.

6.6 WebQS3 Executable Model Generator

Shown in Figure 32, the executable model generator, or simply the *generator*, performs transformations of valid XML model representation files into the executable object code understandable by the simulation manager. At this point, the validity of the XML model representation is important, as an invalid model representation file cannot successfully be transformed into valid executable code. Also, it is more configurable to validate the state of a model representation using the schema tools available in XML than performing binary validation of the generated model code.

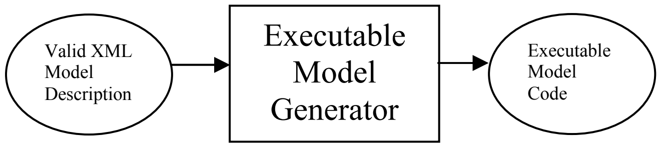


Figure 32. Basic Model Generator Component

The generator may take the XML model and either compiles the model into an executable form (similar to building and linking an executable in C++), or by populating an interpreted runtime framework with simulation objects conforming to the XML model. WebQS3 takes the second approach in creating the executable code. All simulation model objects are instantiated as default objects and the needed object properties are set to determine the behavior of the component. For example, a Server object is instantiated as a default component and then the "service time" variate generator, "move time to next place" variate generator, and the available next places are set based on the provided XML model. Generating the executable model in this fashion allows for models to be constructed more easily than assembling a compiled source version. As this is a small system, there is no reason to create a complex system of compiled executables.

6.6.1 Approaches to Model Generation

There are three basic approaches to transforming XML into executable code. All of the approaches need to provide a mapping between the individual XML elements and the executable representation of the element. Differences arise depending on who provides the mapping and generates the executable objects. The first approach involves the use of specialty binding tools that ties a specific XML format to a set of executable objects the tool creates. The second approach relies on the use of a configuration file to describe the mapping between the XML model and the executable objects. The last approach embeds the XML mapping inside the executable object. The various approaches are described below to describe the design issues surrounding the various generation methods and provide a background for further discussion.

6.6.1.1 XML Binding Tools

Several companies have provided binding tools such as Quick [2002], Castor [2004], or the Java Architecture for XML Binding (JAXB) [2004]. These tools allow a developer to provide an

XML file to the tool and generate a set of executable objects at *compile* time that will directly represent the XML instance. This is a beneficial approach to use when determining a set XML format in advance of the development effort. Furthermore the developer should be comfortable with the creation of executable objects by an outside tool. In this type of XML interpretation, the executable objects created by the parser are usually simple data holder objects. The simulation object developer must then interpret the data objects to determine the behavior of the individual simulation objects. An example of an XML binding parser (JAXB) is demonstrated in Figure 33. As the figure shows, the developer provides an XSD description that is sent through a *binding compiler* to create a series of classes and interfaces based on the XSD description. The classes and interfaces then work in conjunction with the XML document and XML parser to create *content objects*, data objects that represent the data encapsulated in the XML document. Next, the developers code references the content objects to extract the needed information, just as if the developer referenced the XML directly. The tech article "Java Architecture for XML Binding (JAXB)" [2003] provides in-depth discussion of the process.

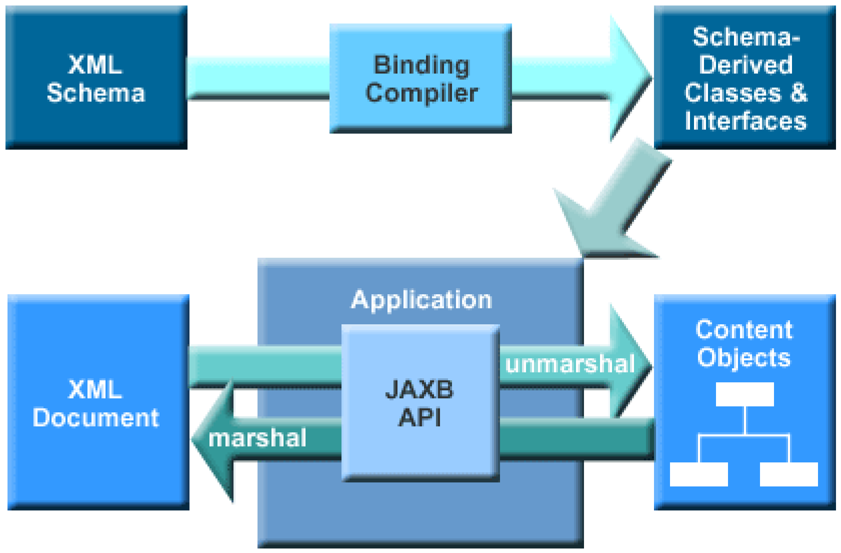


Figure 33. Java Architecture for XML Binding [JAXB 2003]
© 1994-2004. Sun Microsystems, Inc. Used with permission.

Benefits of using binding tools for XML interpretation and model generation include the simplicity of the use for the developer and the increased performance of the parse. Binding tools are extremely simple for developers to use; the developer has only to provide a sample XML file or schema and the tool will generate the appropriate set of parsing classes and executable data representations of the XML. The binding tool handles all of the "plumbing" involved in the XML processing, allowing the developer to concentrate on the business logic (simulation object functionality) of the system. Tying the XML instance document to a generated set of classes allows the binding tool to provide an optimized parser pipeline for the particular XML document, creating a very efficient parsing process. Also, if the XML document handed to the generator is invalid for any reason, binding tools based on an XML schema usually include extra object checking to provide validation incase invalid XML documents make it past the validator component. This could serve as a backup to the XSV module that appears before this component in the architecture.

The negative aspects of this approach are the modifiability of the XML description, the use of a single schema language, and number of objects created. Every time the XML representation format of the model changes, the binding tool will need to be re-invoked to create a new set of data representation objects. This can cause problems if the XML model representation schemas change often, as is the case since the simulation executable builder will need to be recompiled and linked with every schema change. The second disadvantage is the reliance of most tools on a single schema description language. Many of the tools use a variant of XSD Schema or a custom description in determining the set of executable objects to create for data representation. As noted before in the description of the XSV, some XML Model conditions cannot be described using a single schema description and must be enhanced with other schema types. Utilizing binding tools that rely on a single schema description may not be possible for some XML Model structures. Finally, giving up control of the XML binding to the binding tool can cause massive object creation bloat. Some binding tools generate as many as a dozen classes to handle the parsing of a single XML element. Managing that many objects may cause problems when trying to keep the footprint of an executable small.

While this type of XML parsing and configuration was investigated, it was not used for the WebQS3 model generator, as the XML model representation schema was still in flux at the time the generator was created. Utilizing a binding tool simplifies the XML parsing process for the developer, but unnecessarily limits the development of the XML model schema descriptions for simulation elements.

6.6.1.2 Configurable XML Mappings

Configurable XML mappings allow developers to specify an exact mapping between an XML instance file and a set of executable objects. The process of XML parsing using configurable mappings is similar to the services provided by binding tools, except that configurable mapping tools allow more flexibility. Configurable mappings tools, such as the Apache Commons Digester [Digester 2004], provide greater flexibility by parsing the mapping at *runtime* as opposed to compile time. This allows the mapping file to be easily changed to correspond to changes in the XML schema description or underlying executable simulation objects.

Configurable mapping tools expect the simulation object developer to provide two things: a set of executable objects to be populated with data and a mapping file indicating which XML elements are mapped to which executable object. The mapping file specifies that when a particular XML element is parsed, certain data fields on the mapped executable simulation object should be populated with the results of the parsed XML content. The mapping file stipulates that XML elements may not simply be mapped to a data field but also possibly to a set of method calls on the executable object. For example, when the configurable mapping parser finds an element **`<place ref="queue1"/>`**, the mapping file can indicate that the parser should look up an element with an id of “queue1” as opposed to creating a property with name “ref” and value “queue1.” The use of configurable mapping parsers has the benefit of allowing simulation object designers to have their objects populated directly from the XML file data as opposed to using an intermediary data object, as provided by the binding tools. The mapping parsers also allow more configuration options than typically offered by binding tools, allowing complex operations to be programmed into the parser at runtime. The downside of using mapping parsers occurs because of its configuration. The configuration file is sometimes difficult to understand for less experienced developers, and also must be maintained to reflect all the changes made to the incoming XML documents. There may also be problems where mapping must directly correspond to a valid document. If the document handed to the generator is for some reason invalid, a mapping parser relies on the underlying executable objects to detect any problems. If an extra validation is required, then the set of executable objects must be extended.

Configurable mapping generators are useful for larger systems where the available simulation object base can be extended. The mapping generator allows simulation objects and modules to be added to the system by extending the mapping configuration. The WebQS3 system does not use a pure configurable mapping generator; WebQS3 is a small system and the cost of implementing and maintaining the mapping file outweighed the extensibility benefits. The idea of a mapping editor inspired the system generator and the XML representation format, but the system generator implements a hybrid-embedded mappings technique rather than a configurable mapping file.

6.6.1.3 Embedded Mappings

Embedded mapping generators are the least configurable and maintainable approach of the three presented, but is the easiest to utilize and implement. Using an embedded mappings approach, the developer directly embeds an XML parser into the model generator component and inspects the XML tree generated for specific element instances. For example, the developer knows that the XML document should start with a **`<simulation>`** element with a **`<model>`** element as a child. The parser searches the XML tree for a **`<simulation>`** element at the root. Then the parser iterates over the list of children to find the **`<model>`** element holding the data to construct the executable model component. The embedded mapping generator walks the XML tree in an attempt to locate the data needed, ignoring the non-essential parts of the document.

In this approach, the mapping file described in configurable mapping section is no longer interpreted at runtime, but is instead embedded in the actual compiled code in the form of executable commands created by the developer. The mapping is directly tied to a specific XML instance format and cannot be changed without recompiling the parser. Benefits of embedded mapping generators include simplicity, developer control, and increased performance. The generation process under this approach is simplistic; the developer can extract needed elements from the XML document and dictate how individual elements are handled. The developer is also in direct control of the parsing process; this allows operations such as out of order handling of elements or threaded execution. By focusing only on necessary data elements, the generation process can be streamlined to improve performance by processing only the needed elements. If there is a large quantity of elements present in the XML that are not needed for the model generation, the unessential elements can be ignored during the processing stage, improving generator performance. The weakness of embedded mapping generators is the maintainability. In order to modify the generator process, the entire generator component needs to be recompiled. This can become an issue if the XML format of the model changes frequently during the system lifetime. Using an embedded mapping generator does not allow many configuration options, but developing and testing are quick when the XML instance document format is known and the executable simulation objects are provided beforehand.

6.6.2 WebQS3 Generation Process

The WebQS3 generator is based on a combination of an embedded mapping generator and a configurable mapping generator. The configuration mapping is partially embedded in the generator, and partially embedded in the XML instance document itself. Every XML Model instance is created to reflect an object graph. The XML Model is formatted where the top-level element of every component is the executable type of the element (e.g., a **`<basic:Generator>`** element is represented by a **`Generator`** executable Java element). The XML model representation also takes advantage of XML-NS to provide a unique namespace to every element type (see section 4.4.2), which corresponds to the Java element package in the executable system (e.g., elements in the **`basic`** namespace correspond to the **`vt.simulation.component.basic`** java package to uniquely identify every executable element). The java package is identified using the XML-NS URI association. Finally, the child elements of the top-level arguments represent the properties of the individual executable model elements. If the child property has further children, it is known that the element represents a child object, whereas if the element contains no child the element must represent a primitive value or an external element reference. An example of the component property mapping is shown in Figure 34.

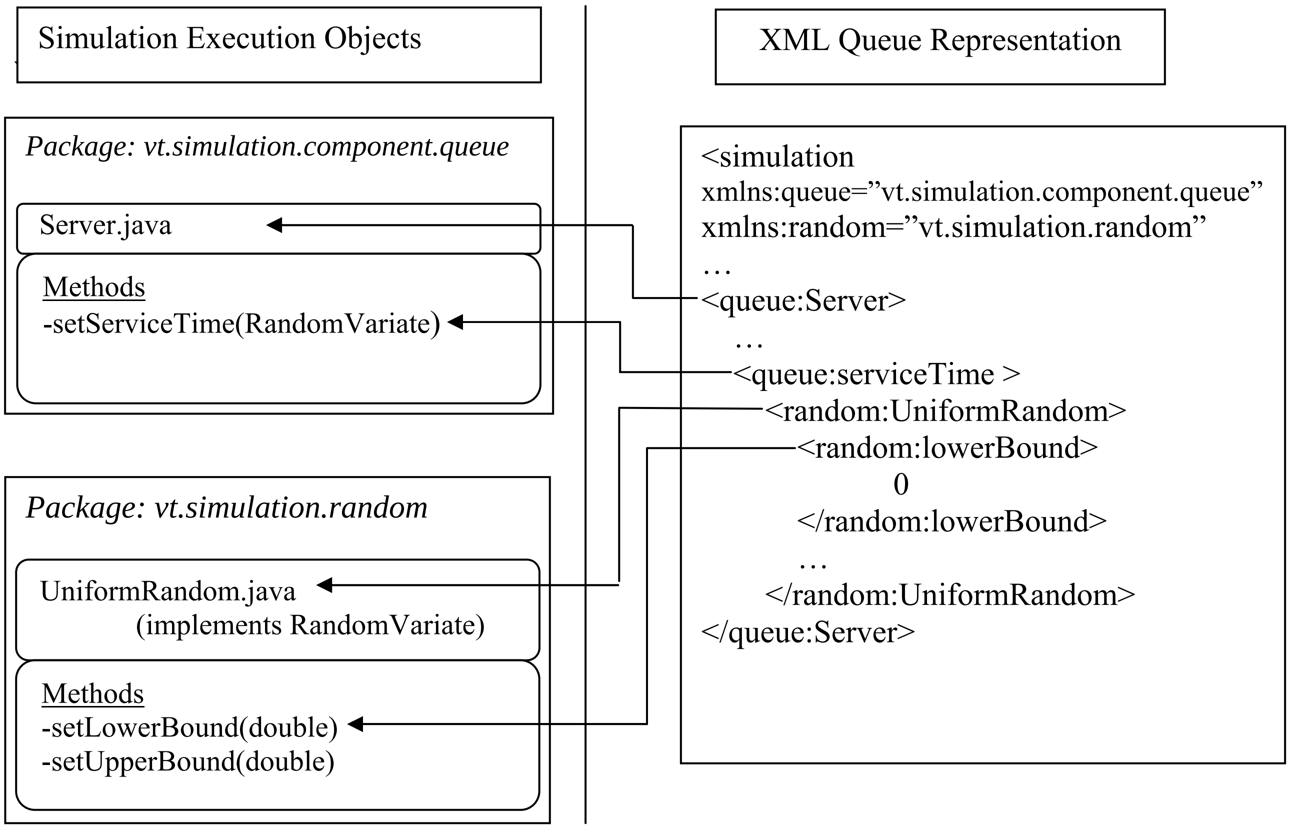


Figure 34. WebQS3 Generator Example

Formatting the XML Model in this manner virtually replaces the mapping configuration file with the XML Model instance document itself. The parser can still be written in an extensible manner by investigating the properties and children of every element encountered in a recursive manner. The XML Model document can be easily modified and extended without changing the model generation component. Unfortunately, this strength is also a drawback. Requiring the XML Model to be formatted as an object graph may limit how the various XML elements can be described in the schema, forcing them to reflect the object structure of the underlying executable elements. For developers unwilling to expose the executable object model to simulation model creators, the use of this type of generator is a serious security concern. Security conscious developers may do better to use a configurable mapping generator. Taking the requirements for WebQS3 into consideration, this type of generator provided the greatest flexibility with the easiest implementation and so it was chosen as the approach to take in generating executable models.

6.7 WebQS3 Model Database

The WebQS3 model database provides a data persistence mechanism for WebQS3. The simulation execution scheme does not necessitate much persistence in running simulations, so the system utilizes the database mainly for infrastructure purposes. The database stores user information and the XML form of the simulation models across user sessions. Users can save complete models on a server and come back at a later date to edit the properties of the model, or execute a simulation based on the model. Storing the XML model representation in the database not only allows the models to be stored across user sessions, but also allows model sharing between system users. By marking a model as publicly accessible in the database, multiple users are allowed to access and run stored models created by other users. Creating publicly accessible models easily enhances collaboration and communication between users working on the same simulation. Users can access, edit, and run the simulation while at different work sites and discuss the outcome of the simulation over email or Internet chat. The WebQS3 system provides simple collaboration features by allowing models to be set as publicly accessible or publicly editable. In a more robust simulation system, the model database should assume a much larger role in allowing persistent storage in case of system failure or long running simulations, to improve overall system performance, or to provide easy multi-user access to executing simulations.

Storing executing simulation messages in a persistent storage medium allows the system to account for user disconnect or system failures. As a result of persistent storage, this simulation system can recover from failure because the state of a currently executing simulation is stored in the database at all times. The simulation system recovers by returning the system to the state stored in the database. Maintaining the simulation state in the database also allows simulations to be paused across user sessions. A user can pause an executing simulation and restart the execution minutes, hours, or possibly days later without requiring system execution resources to maintain the running simulation. Utilizing the database to store long running simulation state eases load on the memory and computation resources of the execution server. Easing the memory and computation load becomes an important factor when an executing simulation generates several hundred thousand messages. Using a RDBMS to manage data makes development simpler for system developers as the RDBMS decides when to store information to the file system or leave information in memory.

The model database can store not only the XML representation of a simulation model, but also the generated executable form of the simulation model. Storing the executable model enhances performance of the system by only performing model generation once when the model is saved, as opposed to every time the simulation is executed. Caching the generated executable especially helps in cases where the generation process takes a long time, such as for larger models or complex simulation components. The simulation components provided in the system are small and quickly generated, so there was no need to store the executable binary in the simulation database.

Storing simulation execution information in the database helps to maintain system consistency across failures and allows multiple users to access executing simulation information using standard data mapping components, such as Enterprise Java Beans (EJB). Using data mapping components permits a simulation identifier to be assigned to every piece of information stored in the database. With multiple users retrieving simulation information, the user can present the simulation id to the simulation execution system and retrieve information and simulation state specific to an active simulation. While possible using simple memory structures, allowing a RDBMS or application server to handle the memory management shifts the responsibility to a component that has been specifically developed for the management task.

6.8 WebQS3 Simulation Manager

The heart of the simulation system architecture is the *simulation manager*; it executes a provided simulation model and sends simulation data messages to waiting users. As executing a simulation is the main function of the system, designing the simulation manager to provide excellent performance and high maintainability is an important issue. The simulation manager provides a wrapper around any available simulation system, managing the execution of individual simulation instances and ensuring that the simulations run in a consistent manner without drastically impacting overall server performance. As Figure 35 shows, the simulation manager accepts a thread-safe instance of an executable simulation model that is executed by the manager. The manager then emits multiple XML-formatted simulation messages declaring changes in the simulation execution as the changes occur. When the simulation manager accepts an executable model to execute, the manager spawns a new simulation runner that allocates a set of resources to the simulation execution. The simulation runner then runs to completion of the simulation execution unless the user explicitly stops or pauses the simulation.

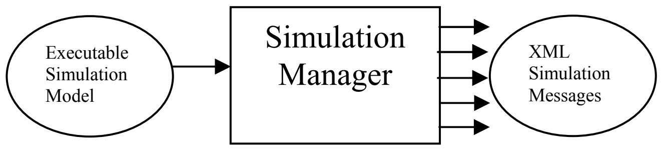


Figure 35. Simulation Manager Component

6.8.1 Design Issues

The manager component requires special consideration in the development. Once the manager accepts a model to execute, it must execute the simulation and emit messages concurrently. The simulation manager requires a single input product, but the output from the component is not a single work product. Instead, the output is a set of messages of indeterminate count that can be emitted at any time. Given the nature of the component, a special threading model must be created that allows for the asynchronous execution of the simulation, while at the same time handling output messages. If a special threading model is not implemented, then a possibility exists for message starvation at the user end, especially in the case of long running simulations. If the visualization manager blocks waiting for simulation messages to arrive, the visualization manager will appear unresponsive and inhibit visualization from occurring. This will significantly degrade the user experience without creating a mechanism for concurrent execution.

In creating the manager, it is also imperative that a system be created that supports multiple users executing simulations at the same time. The simulation runner must create all objects at the process level and pay special consideration to shared resources and pooled objects. Reusing resources between processes working at the same time may induce errors in the simulation. For example, the simulation manager cannot utilize a static simulation execution manager to track simulation level variables. Referring to static variables makes the implementation of individual executable simulation objects simpler, but causes havoc when multiple simulation runners are executing concurrently. A static method call lets any object in the system query the simulation execution manager for simulation properties, such as the current simulation time. The difficult situations occur when individual processes attempt to alter the shared variable. Allowing multiple processes access to the shared variable requires synchronization code between the processes. If not implemented correctly, both processes could block and create a deadlock situation in the simulation manager. Also consider that a static simulation manager has no mechanism of determining which objects belong to which simulation execution when multiple simulations run concurrently. Execution management is much easier if the simulation manager is created to work under a single threaded model, thus ignoring synchronization and multiple-simulation situations.

Implementing the manager on a single-processor server and considering multiple users requires the use of a preemptive time-shared thread scheduler to avoid thread starvation. When a simulation runner executes a long running simulation, the simulation must be stopped periodically to allow other users' simulation processes to run. If the thread scheduler is not pre-emptive, then thread starvation may occur by not allowing other users simulation runners to execute, creating a situation where some users will not receive simulation messages.

The final design consideration pertains to long-running simulations in which the emitting of simulation messages outpaces the acceptance of the messages by the user. The simulation system is forced to buffer the simulation messages until the user accepts the messages for visualization. If the simulation runner continues to execute undeterred, then the possibility exists of message buffer overflow, or possibly an out-of-memory error. The simulation manager can handle allocated resources more efficiently by providing logic that will monitor the length of the message buffer. To illustrate: when the simulation manager detects that the message buffer for the currently executing simulation is near overflowing, the process managing the simulation runner can be placed into a waiting state. The simulation manager waits for the message buffer to decrease in length before reactivating the runner process. The simulation manager can switch out processes that do not need to be run immediately in order to allow other processes seeking computation time to execute, and thus maintaining a steady stream of simulation output to all users.

6.8.2 Design Attempts

The initial design utilized the toolset available in the J2EE 1.3 specification. While the design proved initially feasible, several problems were encountered as the implementation progressed. The decision was made to create a custom simulation manager by extending the J2EE environment. Extending the application server to handle the simulation manager design yielded more control over the system, but also created many new execution and resource handling responsibilities for the designer to consider.

6.8.2.1 J2EE Component Design

J2EE components achieve asynchronous execution using JMS message queues and message driven beans (MDB). MDBs represent a specialized type of stateless session EJBs designed to accept enterprise messages asynchronously from a JMS messaging system. System components send a message to the MDB component and return control to the calling component without waiting for a return message. When an MDB receives an incoming message the application server spawns a new thread to process the message. It must also be noted that MDBs are stateless, meaning that it cannot be guaranteed that individual MDB instances will maintain state between executions, or even that the same MDB instance will handle consecutive messages.

Creating the simulation execution manager as an MDB instance would be advantageous for several reasons. First of all, the application server handles thread scheduling and control. Relying on the application server to handle threads eliminates a source of many headaches in system development and debugging, making the overall system development much easier and creating a user-scalable system by default. The application server automatically handles preemptive scheduling of threads and creates high user scalability by relying on the dependable J2EE architecture components. Secondly, utilizing the JMS system for the underlying messaging architecture delegates the responsibility of managing simulation messages between components to a dedicated system that has been developed to handle enterprise level messaging. The messaging system automatically handles problems such as buffer overflow, message persistence for long-lived messages, and message delivery to multiple subscribers. Finally, utilizing standard J2EE components ensures that the system can be easily ported between application servers and different hardware configurations if needed. Figure 36 shows the architecture of the initial J2EE MDB/JMS design. The individual threaded components, represented by rectangles, are implemented as MDBs with managed JMS queues acting as asynchronous messages buffers in-between the components. The components listen for incoming messages; when a message is received a new thread is spawned to handle the message. The simulation manager creates the message handler components, represented by hexagons, for every instance of a simulation execution. The message handler components asynchronously listen for messages from the simulation runner and handle the messages as needed. Constructing the system using JMS and MDB creates a concurrent and scalable system of the type required for the simulation manager.

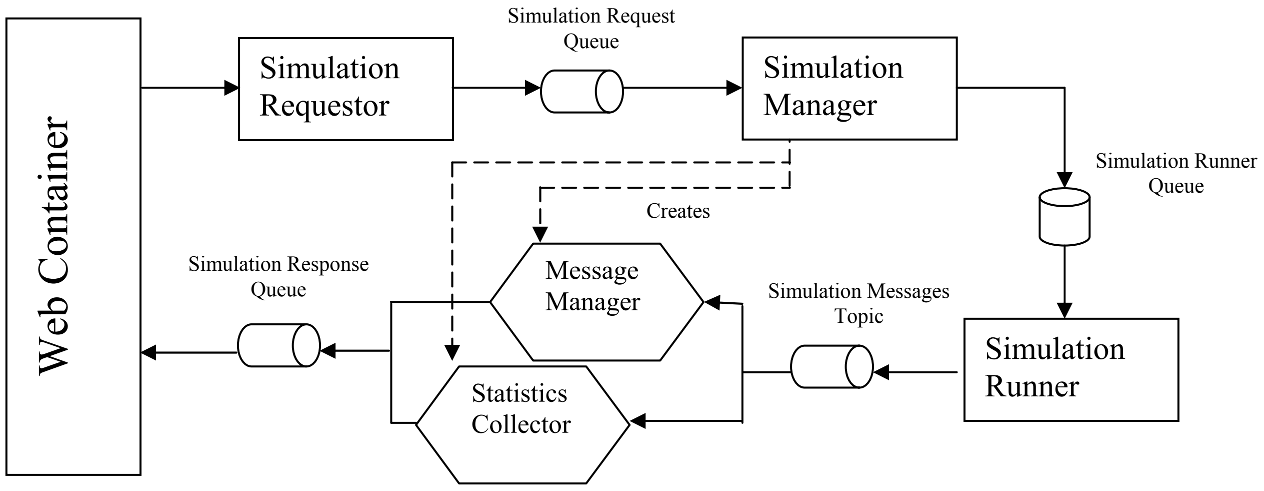


Figure 36. MBD/JMS Simulation System Architecture

Several configurations utilizing standard J2EE MDBs were implemented in an attempt to forge the desired simulation manager environment. While the designs initially appeared successful, problems were encountered as more of the design was implemented. There are two specific examples of why the J2EE based design was not feasible. Problems originally arose when utilizing the JMS messaging system to handle messaging between components. The J2EE specification does not allow any component to create an asynchronous message listener inside the EJB or web application container. The restriction proved difficult to skirt, as it meant that only MDBs or synchronous message listeners could be instantiated. Utilizing a blocking message listener was not functionally possible since J2EE works under a single threaded model. The J2EE system is specified to run in a single thread, waiting for each operation to complete before the other asynchronous operations initiated from the thread can start. Creating a blocking message listener within an executing MDB thread causes the thread to wait, emitting none of the messages the MDB marked for sending, in turn, causing a deadlock situation. A more concrete example is the process of the simulation manager creating the statistics collector. At the outset, the simulation manager sends a message to the simulation runner indicating that the runner should start a new execution. The thread queues the start simulation message to wait for sending when the manager startup method finishes. Next the manager creates the collector, which causes the manager thread to block waiting for simulation messages from the simulation runner. The simulation runner waits for a message to signal execution, yet the start-execution message is blocked while waiting to be sent from the manager thread. A deadlock situation has been created causing the appearance of system failure, while messages are only queued while waiting to be sent. While it was possible to create several different simulation manager instances, the individual simulation executions block waiting for message input to occur. The system was highly scalable to many users, but block waiting for messages creating an unusable system where no work is performed.

While it is possible to get around the problem by spawning a new "listener" thread to block and wait for messages, it is strongly discouraged in the J2EE specification. The application server environment has been finely tuned to manage the threaded environment; spawning new threads outside of application server control can cause difficulties in managing the system, especially when attempting to control threads in the EJB container. In fact, newer J2EE specifications are disallowing new threads to be spawned, so if the system spawned new listener threads the system may not work after future system upgrades.

Since using message listeners is impractical, the only remaining option is to use an MDB to handle the messaging. Using MDBs leads to problems because the MDB instances are stateless. In some situations, such as statistic collection, state needs to be maintained between message handling. It is possible to maintain state between messages using database calls, but the system performance may be degraded as the MDBs handle several hundred thousand messages per simulation. The J2EE standard design settled on using MDBs as simulation message listeners, as MDBs provided the only viable option. While the message listener problem was easily solved, more serious problems were encountered with the J2EE threading model.

The J2EE threading model specifies that the developer of J2EE components should assume a single threaded environment. Initially, a single threaded environment appeared suitable for the simulation execution manager; in the end, the J2EE model did not match the threading model required by the design. For example, when a simulation runner MDB accepted an "execute simulation" message, the runner loaded the execution environment and began generating simulation messages as output. The messages were sent directly to JMS queues where upon they were sent to waiting listeners to perform further computation (e.g., visualization or stat collection). Problems arose with this approach in that the simulation manager thread never gave up control to actually send the messages to the listeners. The messages were simply queued up to wait for sending at a later time. As noted previously, the application server did not allow the messages to be sent from the simulation runner MDB thread until the MDB completed processing. Thus, when a long-running simulation executes, no messages are sent to listeners until the end of the simulation execution. The system appears to be unresponsive creating a degraded experience for users. A concurrent simulation manager based on MDB does not appear possible because the application server assumes a single threaded execution profile, even in the context of handling and sending asynchronous messages. A means of creating system-managed threads in the EJB container is needed. Such a facility is not available in standard J2EE, but may be available through custom extensions on higher end application servers. Using a high-end application server was deemed infeasible, especially due to inexperience with the platform, so it was decided to give up on the standard J2EE mechanisms and attempt a custom solution.

6.8.2.2 Custom J2EE Extension Design

The custom J2EE simulation implementation creates a managed-thread context outside the scope of the application server. This allows the system to handle its own thread management, which is more difficult than letting the application server to handle thread management but allows much more control and flexibility. The extension consists of a set of two threads along with a custom messaging system created for every simulation execution request. The manager design follows the architecture explained in the J2EE-standard component design, but uses synchronous Java objects instead of asynchronous MDBs.

The use of independent threads allows the simulation execution to run concurrently. The first thread, the *execution* thread, wraps the simulation execution engine, allowing the simulation execution to run in tandem with other simulation execution instances. The default Java system thread scheduler manages the execution threads in a time-sliced manner, allowing multiple simulations equal access to processor resources. The manager also controls access to processor and memory resources for individual simulation, requiring the execution engine to stop or pause given the runtime conditions of the system. The second, or *monitor,* thread watches the output simulation message buffer and ensures that buffer overflow does not occur. If the message buffer nears overflow, the simulation execution thread is told to pause and wait until the message buffer length decreases. The execution thread is reactivated once the message buffer reaches an acceptable level. The monitor thread is assigned a higher priority than the execution thread, ensuring that the monitor thread always runs ahead of the execution thread. The monitor executes every few seconds, ensuring that the message buffer does not overflow. The monitor thread has the added responsibility of detecting user inactivity. If the simulation message buffer is not accessed for ten minutes, it is assumed that the user is no longer running the simulation. The monitor tears down the execution engine, de-allocating the resources assigned to the execution instance.

The custom message system is a memory-based messaging system that was created to allow messages to be sent between system components. The custom messaging system is not as robust as a JMS system, but it does allow increased control over the message delivery and storage. Under the custom system, messages are queued and delivered immediately to components waiting for simulation messages in the order the messages were emitted. All of the messages are stored in memory until a user requests the message for delivery. While storing messages in memory requires greater system resources, the monitor thread limits any resource overuse by individual processes.

J2EE Extensions usually connect with the application server using the J2EE Connection (J2C) Mechanism. J2C lets the application server manage the connection resources between the server and the extension creating a more efficient system. WebQS3 does not utilize the J2C facilities; instead, the simulation execution manager extension integrates into the web container. The web container allows much more flexible use of threads than the EJB container in the current J2EE specification. Future research may want to examine the possibility of creating a J2C connection to a separate process running the simulation execution manager.

6.8.3 XML Message Generation

Internally, the simulation manager employs a set of message callbacks to generate XML messages. When the simulation is constructed, the simulation runtime registers as a listener for notification of all state changes on the components included in the simulation. The individual components then notify the simulation runtime of a change in the object state by sending a message to waiting listeners. The message contains information regarding the state change. For example, when a **`Generator`** creates new dynamic object, the dynamic object sends a **`create`**

message. This message contains the name of the object created, the simulation time of the creation, and the component that created the object. The simulation runtime accepts the messages generated by the components, converts the messages to XML, and sends the messages to the message manager component.
Table 10 presents a list of the XML messages that the runtime sends to the message manager and the available properties.

Table 10. XML Message Types, Content, and Purpose

XML Message Name	XML Message Attributes		Sent when…
	Attribute Name	Attribute Value	
Create	simObject	Created object name	Dynamic object created
	location	Object creation location	
	Time	Object creation time	
destroy	simObject	Destroyed object name	Dynamic object destroyed
	location	Object destruction location	
	Time	Object destruction time	
migrate	simObject	Migrating object name	Dynamic object moves between components taking no time
	From	Migrating from component	
	To	Migrating to component	
	Time	Migration start time	
move	simObject	Moving object name	Dynamic object moves between two components taking a move time.
	From	Moving from component	
	To	Moving to component	
	duration	Time taking to move	
	Time	Move start time	
state-change	simObject	Name of object changing state	Simulation components changes state
	propertyName	Property changing state	
	oldValue	Old state value	
	newValue	New state value	
	Time	Time of state change	
simulation-started	No attributes		Simulation startup
simulation-ended	No attributes		Simulation complete
replication-started	No attributes		Replication startup
replication-ended	Time	Time of replication end	Replication complete
error	Contains error message as text		System error

The individual simulation components are responsible for notifying the runtime of the appropriate state change information via Java callback function. If the component does not notify the runtime, the runtime is unaware of when an XML message should be sent. Thus, the runtime only reports simulation state changes that the simulation components wish to make public, allowing the individual components to encapsulate internal logic mechanisms. Also, the runtime is not responsible for the content of the messages, as the runtime simply acts as a message converter and sender. The individual components decide what the message content should be. The runtime relies on an accurate report of component state when sending XML messages to the message manager.

6.9 WebQS3 Message Manager

As Figure 37 shows, the message manager accepts simulation messages and applies any pre-visualization processing required. The implemented system message manager consists of two components handling simulation messages: a statistics collector and a message filter. The statistic collector interprets incoming messages to determine a set of given statistics based on a configuration file. The message filter accepts incoming messages and determines if the individual messages need to be forwarded to the output message buffer.

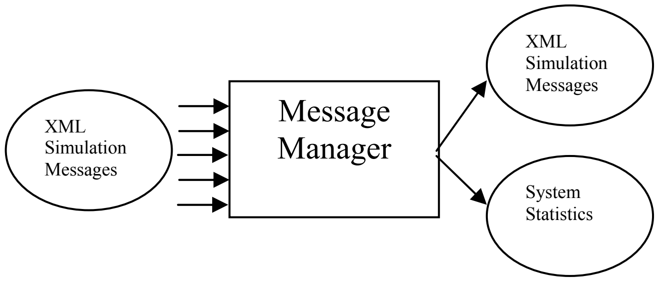


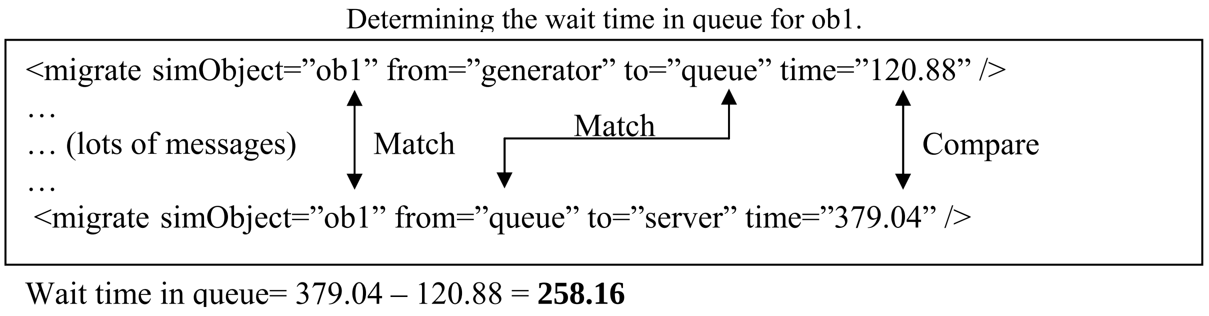
Figure 37. Message Manager Component

6.9.1 Statistics Collection Component

Typically, the statistics collection process is embedded into the individual simulation execution objects. For example, a server computing the overall server utilization statistic embeds the statistic tracking code directly in the simulation object and updates the tracking variables on every object method execution. Extracting the statistic collection outside of the executable simulation object creates a more reusable and customizable simulation component. The simulation components can concentrate on performing the needed simulation logic instead of statistic collection. Also, when the statistics collection is separated from the component execution the two components can be implemented as two separate processes. Creating separate processes allows the components to run concurrently, possibly increasing overall execution performance of the system.

The WebQS3 system implements a separate statistic collection module that listens for incoming XML formatted simulation messages. At initialization time, the statistics collector requires an XML configuration file declaring what statistics to collect for which simulation objects. The statistics can be specified at a system level or for individual components depending on the system need. The collector parses the configuration file and sets up a series of message listeners, listening for a particular message or sequence of messages that allows the message listener to extract the appropriate statistic from the information embedded in the XML message. Once the configuration is parsed, the collector listens for XML simulation messages. If the XML message matches the pattern registered with a message listener, the listener accepts the message and records the information from the XML needed to collect the statistic. Finally, when an **`end-of-replication`** message is detected, the statistics for each message listener stores the current statistic and the statistics collection process resets to begin collection for the next replication.

As an example, the average waiting time of an object in a queue component can be determined by listening for a pair of **`migrate`** messages, as shown in Figure 38. First, the collector listens for a migrate message with a "to" attribute equal to the unique identifier of the queue component and record the identifier of the dynamic simulation object associated with the message and the "in" simulation time of the message. Second, the collector listens for a migrate message with a “from” attribute equal to the unique identifier of the queue. The collector then searches for an existing record with the identifier of the dynamic simulation object associated with the message. The second message represents the “out” simulation time of the simulation object. The wait time for the simulation object is the difference of the “in” time and the “out” time. The set of all wait times can be averaged to find the overall average service time for the component.


Determining the wait time in queue for ob1.

```
<migrate simObject=”ob1” from=”generator” to=”queue” time=”120.88” />
…
… (lots of messages)
…
<migrate simObject=”ob1” from=”queue” to=”server” time=”379.04” />
```

Wait time in queue= 379.04 – 120.88 = **258.16**

Figure 38. Calculating Simulation Statistics

An added benefit of extracting the statistic collection process is the maintainability and extensibility of the collection system. The collection system includes new statistic collection variables by adding components to listen for a new set of simulation messages. Also, the collection modules need not be tied to a specific executable simulation object, but instead be tied to a set of simulation messages. Thus, the collection module is decoupled from the executable object and can be easily reused for other simulation objects desiring to collect the given statistic.

6.9.2 Message Filter

The message filter acts as an optimization mechanism to assist the visualization manager. The filter listens to each incoming simulation message and determines if the message is needed to perform the visualization. If the visualization requires the incoming message, the filter stores the message in the output queue where the message awaits processing by the visualization manager. If the message is not needed, the message is discarded. The filter logic is directly tied to the capabilities of the visualization manager, creating a coupling between the components and decreasing system maintainability. The maintainability decrease is offset by the fact that the filter may eliminate a large quantity of un-needed simulation messages, thus saving network bandwidth resources and visualization processing time. In one example, the performance of the visualization was increased by 125% due to the filter eliminating over half the simulation messages generated by the simulation manager. For future developments, it may be advantageous to create a set of filters tied to individual client types instead of one generic filter, adding a visualization communication optimization layer in the architecture (e.g., a specific message filter for an SVG-based client as well as a specific filter for a Flash-based client).

6.10 WebQS3 Visualization Manager

The visualization manager provides a means of viewing the state of an executing simulation at any time during the simulation execution. Providing an animated visualization of the simulation aids users as means of understanding the simulation through a visual interaction. The visual interaction especially helps users unfamiliar with the details of the simulation. The use of animation and other visual displays opens the usability and availability of the application to users outside the sphere of simulation experts. Due to time constraints and issues raised during informal usability testing only the base functionality of the visualizer was implemented. The simulation visualizer runs a simulation and displays simulation runtime conditions. Conducting further usability testing of the design features is needed to determine if users feel the features lead to a more usable system.

The visualization manager, or simply the *visualizer*, resides on the client computer. The visualizer interacts over a network connection with the message buffer component to request available simulation messages, interpreting the simulation messages retrieved from the network to create the visualization. The visualization manager does not interpret the messages immediately in all cases, but sometimes buffers the messages for interpretation at a later time depending on the state of the visualization. This allows the visualizer to control the resources allocated to the visualization.

This section comprises two sub-sections. To begin, the overall communication strategy is developed explaining how the visualizer starts the simulation execution and communicates with the server throughout the visualization process. A detailed discussion on the visualization strategy follows, describing how the initial visualization is created and how the visualization rendered and updated as time passes.

6.10.1 Server Communication Strategy

The visualizer communicates with the server through a set of Java Servlets accessed via HTTP and a web URL. Calling a particular URL on WebQS3 performs an operation on the server and returns data to the visualizer for interpretation. The list of available simulation operation URLs and return content are provided in Table 11. The client-server communication strategy relies on the asynchronous communication protocol detailed in Chapter 5: , communicating over the HTTP protocol and using standard get and post requests available on any web-enabled computer. Communicating with the server asynchronously through standard HTTP allows the visualizer to retrieve information from the server when needed, without extending the execution environment of the client computer. At the same time, the visualizer controls the flow of information from the server. The visualizer communicates with the server via URL. The visualizer starts a simulation execution and retrieves simulation messages from the message buffer until the simulation signals a completion. At completion, resources devoted to the visualization for the simulation are released. The communication operations are explained in detail in the following sections.

Table 11. Available simulation operation URLs and return content

Simulation Operation	System URL Extension (after /webQS3)	URL Parameters	Return Content
Start Simulation	/StartSimulation	Post operation containing the XML model instance to execute	**`<simUUId>`** XML document containing the simulation execution identifier of the running simulation
Stop Simulation	/StopSimulation	SimId Simulation Execution Identifier	None
Retrieve Messages	/GetSimulationMessages	SimId Simulation Execution Identifier messageCount (Maximum) Number of messages requested	**`<messages>`** XML document containing the next set of simulation messages (since the last access of the message buffer). The XML document is encoded using the "application/gzip" format.
Get Simulation Statistics	/getSimulationStatistics.do	simId Simulation Execution Identifier	XHTML formatted simulation statistical output

6.10.1.1 Start Simulation Operation

When the visualizer wishes to execute a visualization the visualizer calls the **`/StartSimulation`** URL. The **`/StartSimulation`** requires that an XML model document accompany the URL request, so that the simulation manager will have a simulation model to execute. Calling the URL instantiates a new simulation manager instance on the server to run the provided simulation model. The URL call returns an XML formatted document starting with a **`<simUUID>`** element and containing a 16-character text string identifying the executing simulation on the server. The text represents the "universally unique identifier" (UUID) of the simulation manager instance executing the simulation on the server. The WebQS3 server generates the UUID and guarantees that the UUID will be unique to a single simulation execution instance using the UUID for EJB design pattern [Marinescu 2002]. The visualizer stores the execution UUID and uses the key to refer to the particular execution instance to retrieve information from in subsequent server requests. Receiving a UUID signals the visualizer that the simulation manager is ready to execute and that the visualization should begin by starting the process of retrieving simulation messages.

6.10.1.2 Get Simulation Messages Operation

The process of retrieving messages from the server is under the complete control of the visualizer. If the visualizer needs messages to execute the visualization, the visualizer requests messages from the server. Under this paradigm, it is important that the server always have messages available for the visualizer. Thus, the assumption is made that the simulation execution runs faster than the visualization. Putting the client in control of the message retrieval allows the visualizer to manage resources, devoting resources to the visualization when needed and not overtaxing the client system (e.g., retrieving too many messages and filling the available memory of the system).

After the simulation execution initializes, the visualizer undergoes a warm-up period. The visualizer requests messages from the server until an acceptable number of messages are sent. The visualizer buffers the messages received from the server until the warm-up period is marked as over, subsequently starting the normal message retrieval process. Buffering the messages on the client allows the visualization engine to create visualization ahead of the time the visualization is shown to the user to increase performance. The message retrieval process undergoes a warm-up period in hopes of having an adequate supply of messages if any subsequent message requests fail or take longer than expected, while still allowing message processing to occur. The WebQS3 system visualizer uses a warm-up period of 8000 messages.

A robust asynchronous communication protocol requires implementing an adaptable retrieval algorithm. The algorithm monitors the visualization process and determines the number of simulation messages to retrieve. WebQS3 implements a simplistic retrieval algorithm leaving the adaptable algorithm development for future research. The visualizer monitors the number of simulation messages left to process on the client and requests more messages when the level reaches a minimum value. The visualizer currently requests more messages when the buffer falls below 3000 messages. The retrieval algorithm also notes to stop retrieval operations when a **`<simulation-ended/>`** or **`<error/>`** message are detected. The retrieval algorithm takes care to inspect the end of all incoming message documents to see if a **`<simulation-ended/>`** message is present. The message streams must be inspected, otherwise the retrieval algorithm would have to wait for the message to be processed. During message processing time the retrieval algorithm would continue making message requests for non-existent simulation messages and waste network resources. While a simple retrieval algorithm may function poorly for more complex simulations, the retrieval process works adequately for the simulation models targeted by the WebQS3 system.

When messages are required, the visualizer calls the **`/GetSimulationMessages`** URL passing the stored execution UUID as a parameter and the number of messages to return. The number of messages requested may not be the number of messages returned by the server. The server message buffer may not contain the requested number of messages and only return the messages available. For instance: The simulation is running slowly and generating only a few messages or the simulation has ended generating no more messages after the end-of-simulation message. The content returned by the **`/GetSimulationMessages`** URL contains an XML document of the messages retrieved encoded using the application/gzip format. By gziping the XML document, the size of the document can be reduced substantially to reduce network transfer time, but requiring a small time for decompression on the client. As the client is requesting small chunks message content (e.g., 1200 to 2400 messages at a time encompassing about 100 – 200 KB) the performance overhead of the decompression should be minimal. After decompression, the visualizer parses the messages XML document to for later interpretation.

When the visualizer receives messages, the messages are placed into a buffer until retrieved by the message interpreter. The buffer needs to provide two high-performance operations, as the buffer implementation should not devote processor resources away from the visualization. The first operation involves quickly retrieving a single message from the head of the buffer (i.e., retrieving the next simulation message for interpretation). The second operation involves quickly adding a large number of messages to the end of the buffer (i.e., storing newly incoming messages). The data structure chosen for the buffer is a special linked list of shallow trees that allows message retrieval in O (1) time and large numbers of messages to be added in O (1) time. The data structure is dependent on the format of the XML document sent from the server to stay in the current format of a shallow tree having a single root node and N leaf child nodes with no subsequent children. When the buffer receives a new XML tree of messages to store, the buffer simply adds the tree of messages to the end of buffer using a tail pointer in O (1) time. The retrieval process is a bit more complicated. When a message is requested the buffer locates the tree at the beginning of the buffer. Next, the first child node of the tree is removed and stored for return. At this point, if the tree is empty then the tree is removed from the buffer using a head pointer in O (1) time and the stored XML message is returned. The buffer has highly efficient add and remove operations, depending on the structure of the XML message document to aid in the performance.

6.10.1.3 Stop Simulation Operation

When the either the user or the visualizer signals a stop simulation action, a request is made to the **`/GetSimulationMessages`** URL passing the stored execution UUID as a parameter. The request returns no information, but instead signals the server that resources devoted to executing the simulation should be de-allocated. The visualizer also stops any processes devoted to running the visualization and releases messages stored in the message buffer.

6.10.2 Visualization Strategy

The visualization strategy is comprises two parts: creating the initial visualization and updating the visualization based on the simulation messages retrieved from the server. The initial model visualization is displayed when the visualizer firsts launches, displaying the layout of the components that are a part of the model. Once the user starts the visualization process, the visualization is updated by creating a series of animations. The animations display movement of objects between components or state changes in the individual objects. All of the animations are created using SVG (relying on the Synchronized Multimedia Integration Language (SMIL) [SMIL 2004]) and updated using ECMAScript event handlers. The process is detailed below.

6.10.2.1 Initial Model Visualization

The initial model visualization is displayed to the user when the user launches the visualizer tool. The model the user wishes to run is handed to the visualizer by embedding the XML form of the model in the visualizer XML content. The XML simulation model goes through a simple eXtensible Stylesheet Language Transform (XSLT) [XSLT 2004] transformation, used to extract the view components of the model. The visualization components are directly embedded in the visualizer XML display.

Simply extracting the view components of the model is not sufficient to create a complete visualization as the XML simulation. The simulation model only holds view information for the individual components, not the connections between components. Thus, the XSLT transformation must extract the connection information from the XML model by inspecting the connections formed between elements in the component model section of the document. The connection information is extracted, placing the connection line element into the visualizer XML, and assigned a unique **`id`** attribute based on a naming convention of "from place"-MoveTo-"to place." To illustrate, a connection between queue1 and server1 would yield a connection named "queue1-MoveTo-server1." The connection lines are then setup (i.e., moved to the proper display location) at initialization time.

If any other objects or animations need to be setup during the XSLT processing, it is simple to add the appropriate handler by updating the XSLT. For example, state change animations for a working server are created for every **`Server`** component in the model. The XSLT inspects the component type of the visualization being rendered and places a "server working" animation next to the visualization of every server component. Noting that extending the XSLT is as simple as adding a few lines of text, it is easy to extend the simulation visualization components to display new information or include extra animations.

6.10.2.2 Updating the Visualization

The visualization is updated to display the current state of the executing simulation by creating SVG animations to change the state of the visualization. The visualization processor interprets incoming simulation messages and schedules either an animation or a function execution based on the timestamp of the message. Animation executions perform a change to the visualization by showing movement on the screen, such as moving an object between two components. Function executions update simulation state information, such as incrementing the replication number. The particular scheduled operation depends upon the simulation message type. Table 12 shows the list of messages that the visualizer interprets and the operations that are scheduled based on the message type. The visualizer declares an operation to perform for each type of message; when a message is retrieved from the message buffer, the needed operation is scheduled to occur at the appropriate time in the visualizer.

Table 12. Simulation Message Interpretation Operations

Simulation Message	Scheduled Operation Type	Operation Performs
Create	Animation & Function	Creates a new dynamic object visualization and schedules an animation to make the object appear on screen. Also, schedules function execution to increment the count of created objects.
Destroy	Animation & Function	Schedules an animation to make the referenced object disappear from the screen. Also, schedules function to increment the count of destroyed objects.
State-Change	Animation	Schedules an animation to show the state change.
Move	Animation	Schedules an animation to show movement between simulation components.

Replication-Started	Function	Schedules a function to devote resources to new replication execution.
Replication-Ended	Function	Schedules a function to de-allocate resources from ending replication.
Simulation-Started	Function	Resets simulation environment
Simulation-Ended	Function	Schedules a function to stop simulation execution and display statistics.

The scheduling of animations and function executions encompass the main functionality of the visualizer. The visualizer animations rely on a real-time execution time assignment in order to begin at the appropriate time. Also, the real-time clock that the animations are referencing is out of developer control, save for resetting the clock to a particular value (e.g., resetting the clock to 0 for a new replication). Since the real-time clock referenced by animations to determine start time is out of developer hands, the execution timeline for the simulation must be mapped onto the existing real-time clock. The mapping cannot be performed directly, as the mapping function must contend with animation speed and disabled animation variables. The displayed animation speed must reflect the change in the animation speed variable. Therefore, the mapping function is not a constant, but changes over time. In addition, when animation is disabled then no animation should display and the clock update as fast as possible. Figure 39 displays a sample of the time-event mapping as the speed of the animation changes while timed events occur. The circles represent timed events and the vertical dashes represent time units. As the diagram shows, there exists a disparity between the simulation timeline and the visualization timeline when the speed of the animation changes dynamically. The time units in the visualization timeline are not constant, but change dynamically to represent the animation speed.

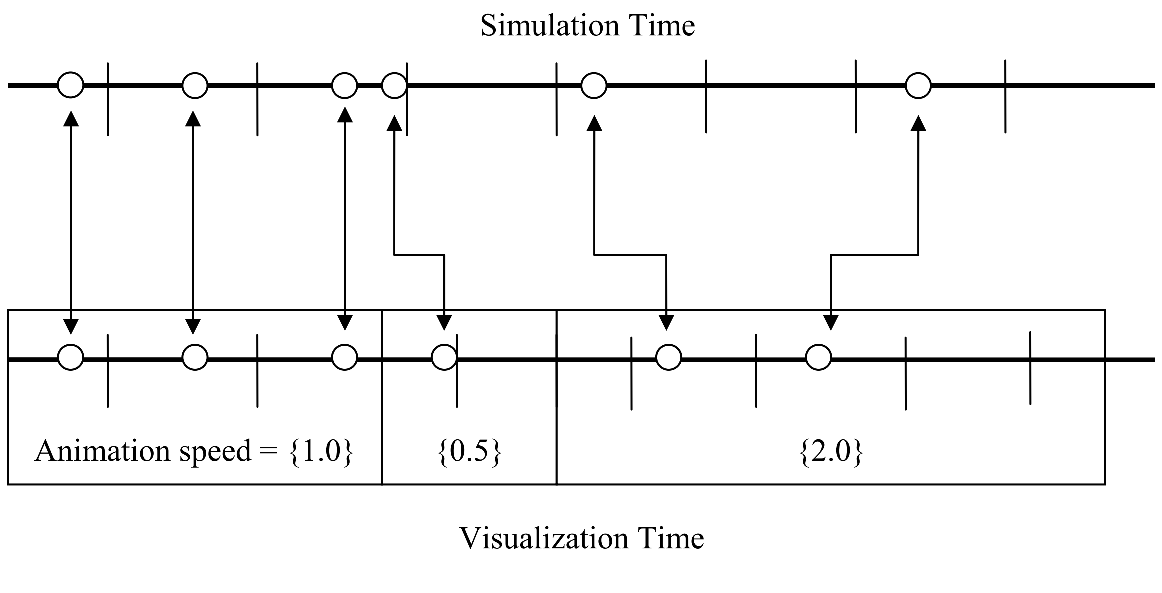


Figure 39. Timed Event Mapping Sample

The visualizer employs a simple formula to map the simulation execution time onto the visualization timeline. The visualization timeline is also referred to as the real-time timeline because the SVG animation relies on a real-time value in milliseconds to determine when to start and stop animations. Initially, the visualizer tracks the time of the last scheduled animation times using two variables: **lastSimulationTime** and `lastRealTime`. The variables track the time of the last scheduled event in both the simulation timeline and the visualization timeline, providing a starting point for the next animation. Next, a calculation is performed to determine the next occurrence of an animation. Taking the simulation time timestamp of the current XML message and subtracting the last simulation time yields the simulation time passed between the messages (`nextSimTime`). `nextSimTime` is divided by the current animation speed to allow acceleration or deceleration of the simulation timeline as necessary. This produces the visualization-time taken between the last visualization event and the next visualization event (`nextRealTime`). Adding `nextRealTime` to `lastRealTime` gives the real-time of the event to schedule with the SVG event scheduler.

The visualizer buffers the simulation messages received so that messages are always available to execute. Buffering the simulation messages ahead of the visualization creation allows the visualizations to be created ahead of the time the animations are displayed, helping the system appear more responsive. By creating the animations ahead of time in the background the system can prepare the animations and resources the animation needs when there is spare time on the processor. A simple change from invisible to visible is all that is required to see the animation. Buffering of the animations allows complex animations requiring many resources to appear quickly on the screen.

Quickly processing the messages and generating animations creates three problems: message buffer underflows, resource allocation, and overlapping timelines. Message buffer underflows can easily occur when processing messages in a greedy manner. The message processor simply retrieves messages at a rate faster than the message retriever can fill the buffer with messages from the server. Resource allocation creates problems during long running processes. Attempting to interpret all the simulation messages at once causes the visualization to slow down considerably or run out of memory by creating too many animation objects. Also, while processing simulation messages, the visualizer cannot devote processor resources to changing the animations, making the system appear unresponsive or jumpy. Overlapping timelines causes problems for short timeline simulations with multiple replications. If simulation messages are processed too fast then the visualizer may schedule animation events for a second replication before events for the first replication have ended. Thus, events for both of the replications may show up at once.

To solve the problems outlined above, the visualizer system implements three preventative measures to limit the number of messages processed at a time. The visualizer provides a processor "choke" that limits the flow of messages into the message processor. If the message processor scheduled messages more than ten seconds out from the current real-time clock, the message processor is bypassed and the processor is made to wait five seconds to allow the animation to sync with the processor. This has the added effect of allowing the simulation manager on the server to keep pace with the visualization, as messages are not requested until needed by the visualizer. Also, the visualizer is only allowed to process a few messages at a time. It then pauses the processing to allow the animation system to update the animations on the screen. The visualizer currently processes 50 messages at a time before it stops execution and waits for the next batch of messages from the buffer. Finally, the processor stops execution at the end of each replication so that the timelines will not overlap. When a **`<replication-ended/>`** message is received, the message processor stops until it is known that overlapping timelines will not occur. At that point, the message processor is restarted so that new simulation messages can be created.

The last responsibility of the visualizer is the display of simulation statistics. Since the visualizer has a limited viewport area and the length of the simulation statistics content is unknown, the visualizer allows the browser to display the statistics. When the simulation visualization ends, a dialog box is displayed, allowing the user to show statistics if they wish. If the user requests the simulation statistics, the visualizer calls the **`/GetSimulationStatistics`** URL, which opens a new browser window. The server retrieves the statistical output from the statistics collection module and processes the output into XML using an XSLT transformation. The results are subsequently shown to the user for inspection.

Chapter 7: Verification and Validation

This chapter presents the verification and validation activities conducted to assess the WebQS3 accuracy. *WebQS3 Verification* deals with the assessment of transformational accuracy of WebQS3 and addresses the question of “Are we creating the WebQS3 right?” [Balci 2003]. *WebQS3 Validation* deals with the assessment of behavioral or representational accuracy of WebQS3 and addresses the question of “Are we creating the right WebQS3?” [Balci 2003].

7.1 WebQS3 Verification

The following verification techniques throughout the WebQS3 development life cycle in assessing the transformational accuracy of WebQS3:

- Desk checking
- Documentation checking
- Inspections
- Structured walkthroughs

Balci [1998] describes of the above techniques in detail.

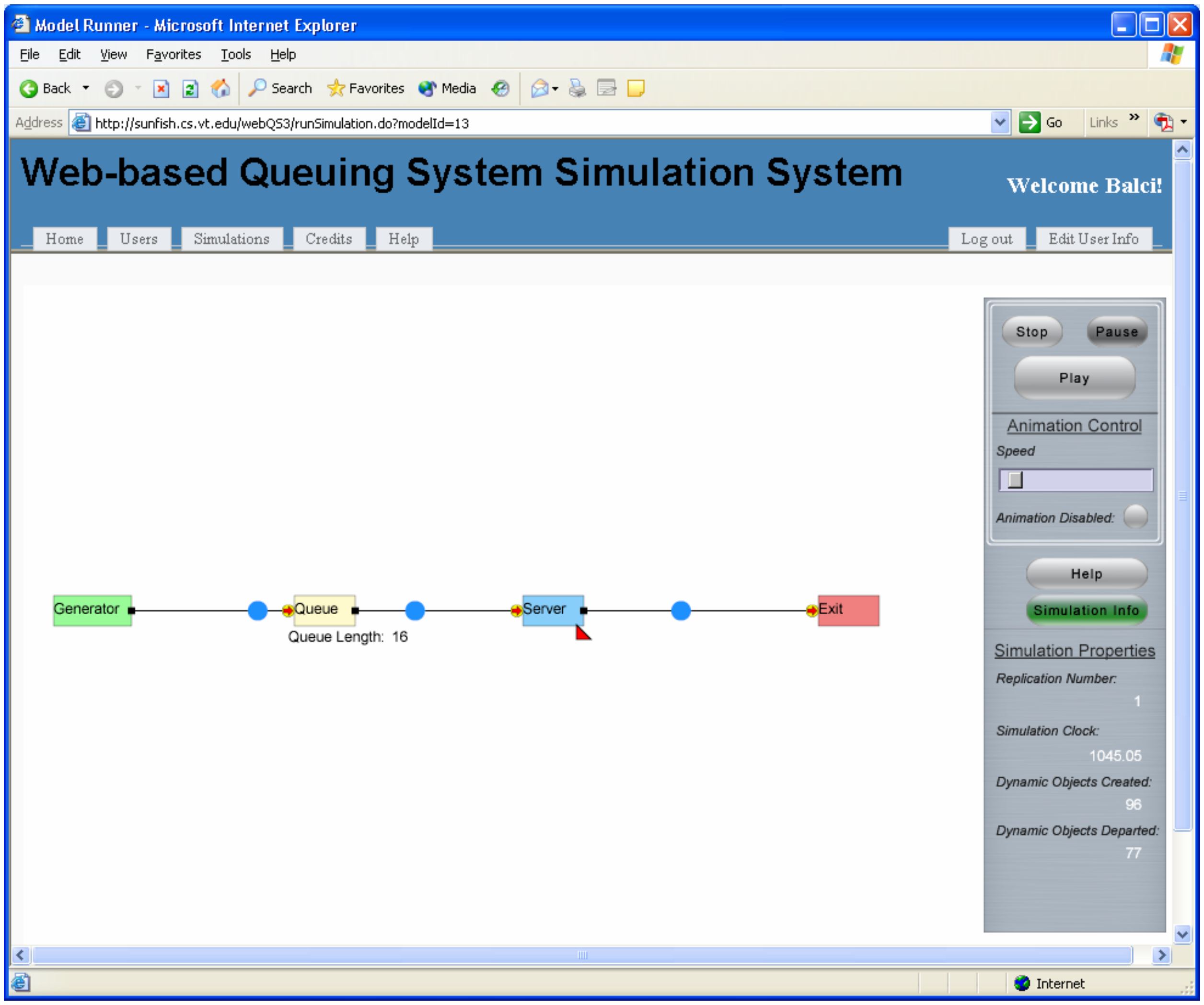
7.2 WebQS3 Validation

In assessing the behavioral or representational accuracy of WebQS3, we used the *Comparison Testing* technique [Balci 1998]. In the case studies described below, we identified three queuing systems that can be modeled by using Queuing Theory or Analytic Modeling. We developed a simulation model of each queuing system using WebQS3 and estimated a set of predefined performance measures. We compared the performance measure values estimated by WebQS3 simulations with the respective performance measure values computed by using Analytic Modeling. The comparison results provided in the case studies below increase our confidence that WebQS3 possesses sufficient behavioral or representational accuracy.

7.2.1 Case Study 1

In this case study, we consider a single line single server queuing system as depicted in Figure 40. It is assumed that the random variable, inter-arrival times of dynamic objects originating from the Generator component, has an Exponential probability distribution with a mean of 22 seconds. The waiting line in front of the server follows the First-Come First-Served (FCFS) queue discipline. The server processes the dynamic objects with a service time that is exponentially distributed with a mean of 20 seconds.

The queuing system described above is designated with the M/M/1 notation in Queuing Theory and has an analytical solution [Gross and Harris 1974]. The first **M** indicates that the arrival process follows a **M**arkovian or Poisson process with an arrival rate of λ, which implies that the inter-arrival times are exponentially distributed with a mean of $1/\lambda$. The second **M** indicates that the service provided by the server also follows a **M**arkovian or Poisson process with a service rate of μ, which implies that the service times are exponentially distributed with a mean of 1/μ. The number 1 in the notation indicates that there is only one server.


Model Runner - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://sunfish.cs.vt.edu/webQS3/runSimulation.do?modelId=13
Go Links
Web-based Queuing System Simulation System
Welcome Balci!
Home Users Simulations Credits Help Log out Edit User Info
Stop Pause Play
Animation Control
Speed
Animation Disabled:
Help
Simulation Info
Generator
Queue
Server
Exit
Internet

Queue Length: 16

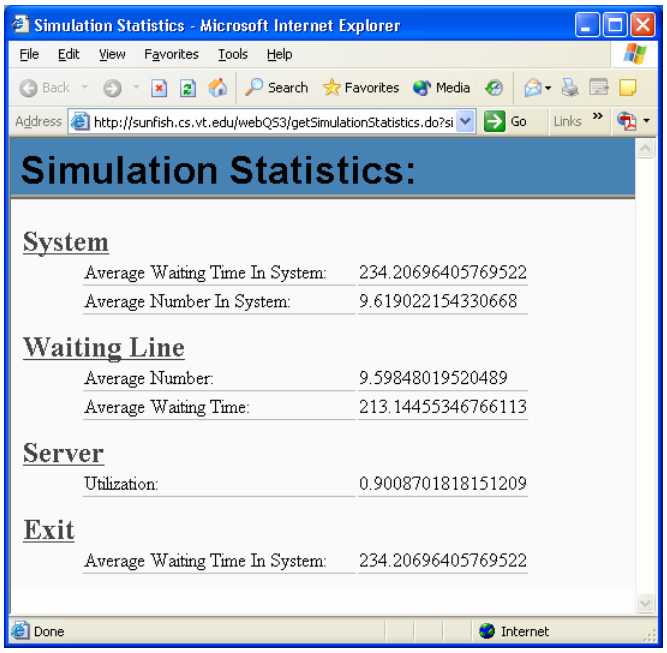
Simulation Properties
Replication Number: 1
Simulation Clock: 1045.05
Dynamic Objects Created: 96
Dynamic Objects Departed: 77

Figure 40. M/M/1 Single Line Single Server Queuing System Simulation using WebQS3

As depicted in Figure 40, we quickly composed a model of the M/M/1 queuing system by using the reusable model components provided by WebQS3. Move times are assigned to the dynamic objects to be able to view a visualization or animation of the simulation. After the model is verified by viewing the visualization, the move times are set to zero, and experiments are performed to estimate the following performance measures:

- ρ = Server Utilization
- W_q = Average Waiting Time of Dynamic Objects in the Waiting Line (Queue)
- L_q = Average Number of Dynamic Objects in the Waiting Line (Queue)
- W = Average Waiting Time of Dynamic Objects in the System
- L = Average Number of Dynamic Objects in the System

The *Method of Replications* [Banks et al. 2001; Law and Kelton 2000] is used to perform the simulation experiments. In a simulation run, the model is warmed up by running the simulation for 1000 dynamic objects departing the system, after which the model is run for 10,000 dynamic objects departing the system in steady state. The average of the 10,000 correlated observations in steady state for a performance measure is taken as an independent observation. The simulation run is replicated 20 times using different random number seeds to obtain 20 independent observations for a performance measure. The user-supplied random number seed is multiplied by the replication number to obtain a different seed for each replication. The average of the 20 independent observations is taken as the point estimate of a performance measure. The WebQS3 simulation experimentation results are shown in Figure 41.


Simulation Statistics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://sunfish.cs.vt.edu/webQS3/getSimulationStatistics.do?si
Go Links
Done
Internet

Simulation Statistics:

System

Average Waiting Time In System:	234.20696405769522
Average Number In System:	9.619022154330668

Waiting Line

Average Number:	9.59848019520489
Average Waiting Time:	213.14455346766113

Server

Utilization:	0.9008701818151209

Exit

Average Waiting Time In System:	234.20696405769522

Figure 41. WebQS3 Simulation Experimentation Results for the M/M/1 Queuing System

Table 13 shows the WebQS3 comparison testing results for the M/M/1 queuing system.

Table 13. WebQS3 Comparison Testing Results for the M/M/1 Queuing System

Performance Measure	Analytical Solution	Estimated Value by WebQS3 Simulation	% Deviation
ρ	0.90910000	0.90087018	0.905271
W_q	200.02200220	213.14455347	6.560554
L_q	9.09200011	9.598480195	5.570612
W	220.02200220	234.2069641	6.447065
L	10.00110011	9.619022154	3.820359

The percent deviations shown in Table 13 are all reasonable given the variance of each performance measure. Hence, our confidence is increased in WebQS3 having sufficient accuracy.

7.2.2 Case Study 2

In this case study, we consider a single line multiple server queuing system as depicted in Figure 42. It is assumed that the random variable, inter-arrival times of dynamic objects originating from the Generator component, has an Exponential probability distribution with a mean of 7.1 seconds. The waiting line in front of the server follows the FCFS queue discipline. The three servers are identical and each processes the dynamic objects with a service time that is exponentially distributed with a mean of 20 seconds.

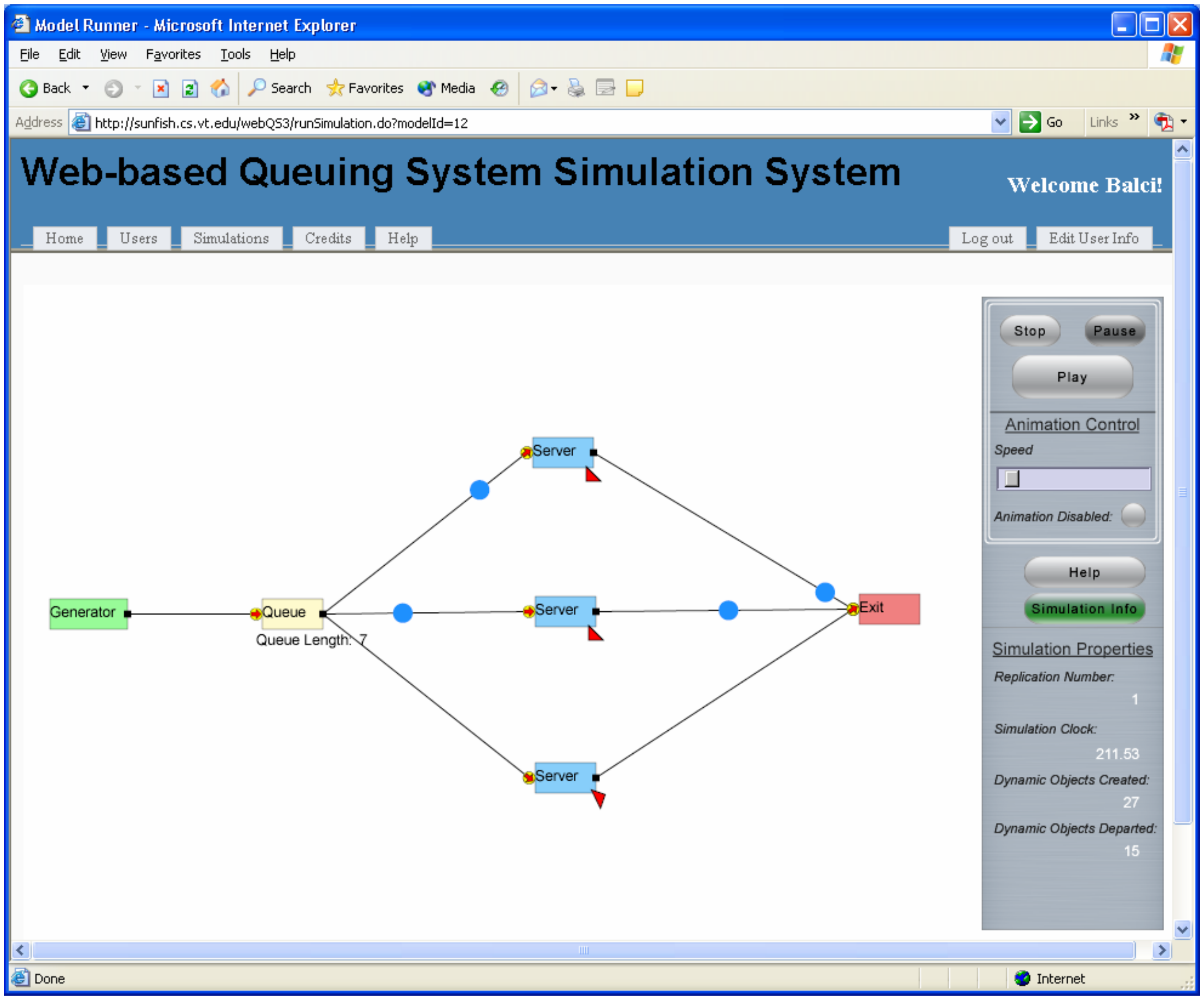

Model Runner - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://sunfish.cs.vt.edu/webQS3/runSimulation.do?modelId=12
Go Links
Web-based Queuing System Simulation System
Welcome Balci!
Home Users Simulations Credits Help Log out Edit User Info
Generator
Queue
Queue Length: 7
Server
Server
Server
Exit
Stop Pause Play
Animation Control
Speed
Animation Disabled:
Help
Simulation Info
Simulation Properties
Replication Number: 1
Simulation Clock: 211.53
Dynamic Objects Created: 27
Dynamic Objects Departed: 15
Done
Internet

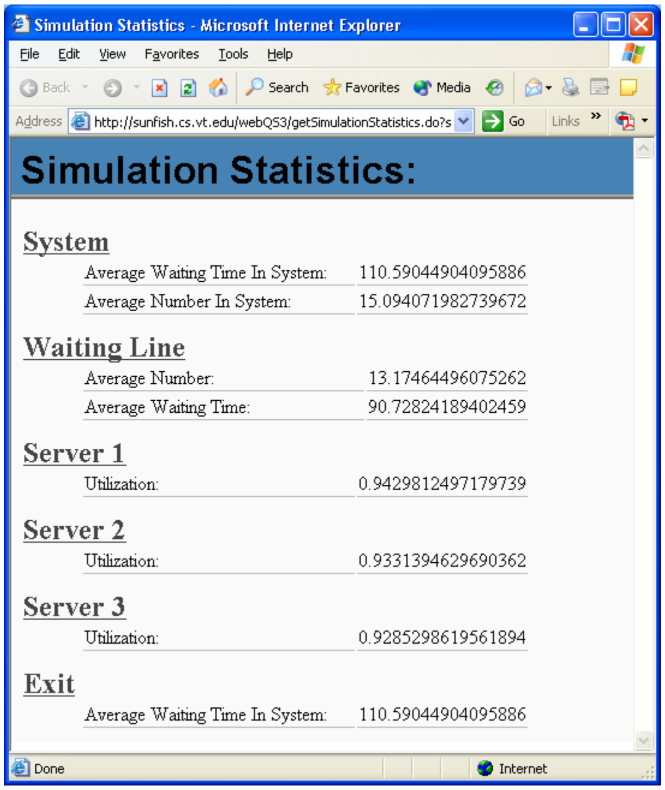
Figure 42. M/M/3 Single Line Multiple Server Queuing System Simulation using WebQS3

The queuing system described above is designated with the M/M/3 notation in Queuing Theory and has an analytical solution [Gross and Harris 1974]. The first **M** indicates that the arrival process follows a **M**arkovian or Poisson process with an arrival rate of λ, which implies that the inter-arrival times are exponentially distributed with a mean of $1/\lambda$. The second **M** indicates that the service provided by the server also follows a **M**arkovian or Poisson process with a service rate of μ, which implies that the service times are exponentially distributed with a mean of $1/\mu$. The number 3 in the notation indicates that there are three identical servers.

As depicted in Figure 42, we quickly composed a model of the M/M/3 queuing system by using the reusable model components provided by WebQS3. Move times are assigned to the dynamic objects to be able to view a visualization or animation of the simulation. After the model is verified by viewing the visualization, the move times are set to zero, and experiments are performed to estimate the following performance measures:

- ρ = Server Utilization
- W_q = Average Waiting Time of Dynamic Objects in the Waiting Line (Queue)
- L_q = Average Number of Dynamic Objects in the Waiting Line (Queue)
- W = Average Waiting Time of Dynamic Objects in the System
- L = Average Number of Dynamic Objects in the System

The *Method of Replications* [Banks et al. 2001; Law and Kelton 2000] is used to perform the simulation experiments. In a simulation run, the model is warmed up by running the simulation for 1000 dynamic objects departing the system, after which the model is run for 10,000 dynamic objects departing the system in steady state. The average of the 10,000 correlated observations in steady state for a performance measure is taken as an independent observation. The simulation run is replicated 20 times using different random number seeds to obtain 20 independent observations for a performance measure. The user-supplied random number seed is multiplied by the replication number to obtain a different seed for each replication. The average of the 20 independent observations is taken as the point estimate of a performance measure. The WebQS3 simulation experimentation results are shown in Figure 43.


Simulation Statistics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://sunfish.cs.vt.edu/webQS3/getSimulationStatistics.do?s Go Links
Done Internet

Simulation Statistics:

System

Average Waiting Time In System:	110.59044904095886
Average Number In System:	15.094071982739672

Waiting Line

Average Number:	13.17464496075262
Average Waiting Time:	90.72824189402459

Server 1

Utilization:	0.9429812497179739

Server 2

Utilization:	0.9331394629690362

Server 3

Utilization:	0.9285298619561894

Exit

Average Waiting Time In System:	110.59044904095886

Figure 43. WebQS3 Simulation Experimentation Results for the M/M/3 Queuing System

Table 14 shows the WebQS3 comparison testing results for the M/M/3 queuing system.

Table 14. WebQS3 Comparison Testing Results for the M/M/3 Queuing System

Performance Measure	Analytical Solution	Estimated Value by WebQS3 Simulation	% Deviation
$\rho_{server1}$	0.93896667	0.94298125	0.427553
$\rho_{server2}$	0.93896667	0.93313946	0.620598
$\rho_{server3}$	0.93896667	0.92852986	1.111520
W_q	96.87503580	90.72824189	6.345075
L_q	13.64436442	13.17464496	3.442590
W	116.87503580	110.59044904	5.377185
L	16.46126442	15.09407198	8.305513

The percent deviations shown in Table 14 are all reasonable given the variance of each performance measure. Hence, our confidence is increased in WebQS3 having sufficient accuracy.

7.2.3 Case Study 3

In this case study, we consider a computer system defined by Balci [1988] and is depicted in Figure 44.

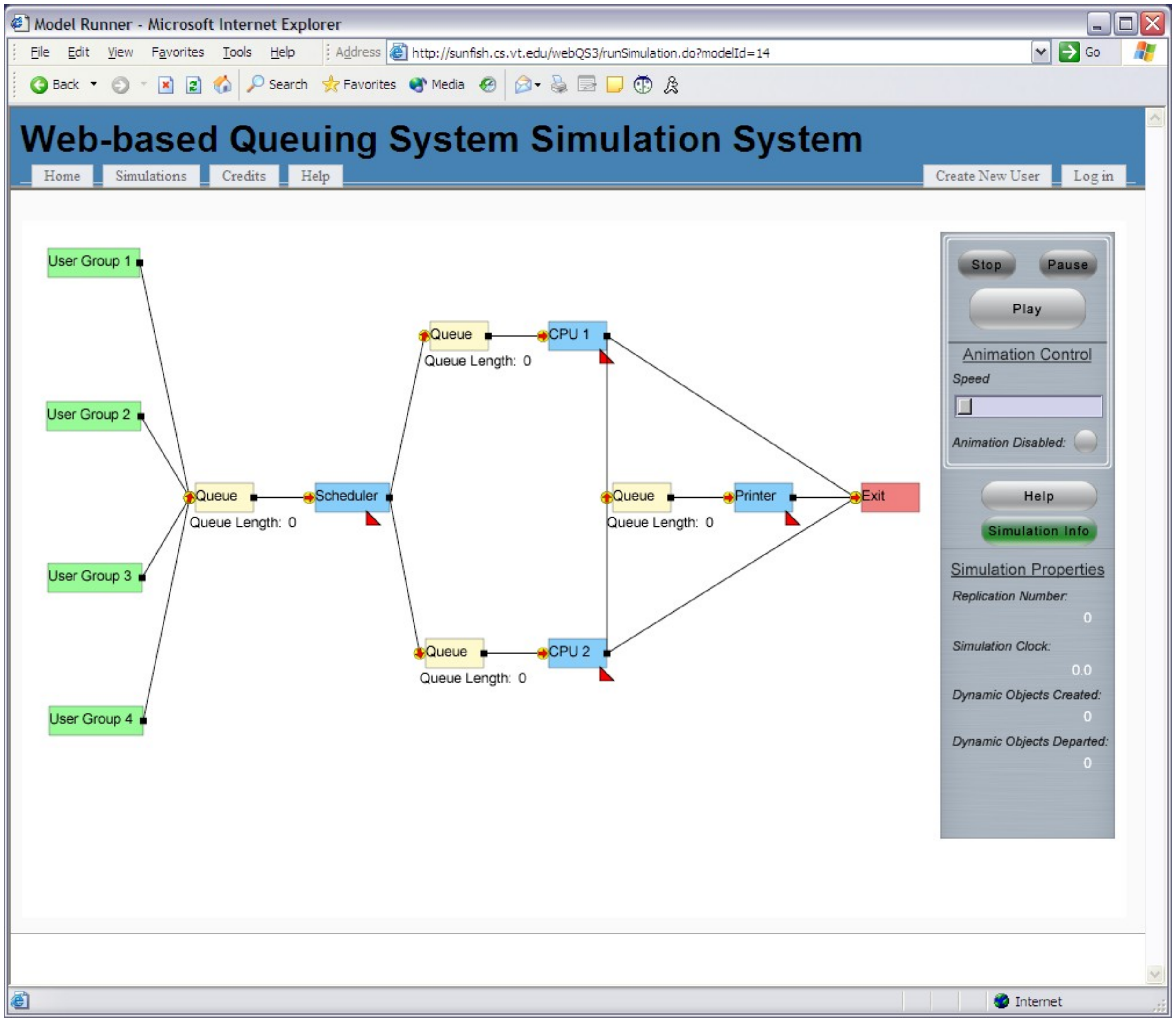


Figure 44. Computer System Simulation using WebQS3

In this computer system, there are four groups of users sending jobs to a computer system. The inter-arrival times of jobs submitted from a user group are exponentially distributed with the means presented in Table 15. An arriving job joins the FCFS queue in front of the Scheduler. The scheduler assigns the job to CPU1 with a probability of 0.6, or to CPU2 with a probability of 0.4. A CPU executes jobs one at a time until completion. At the completion of job execution on a CPU, the job is sent to the printer with a probability of 0.8, or exits the system with a probability of 0.2. It is assumed that all queues in the system follow the FCFS queue discipline.

Table 15. Probability Distribution of Inter-arrival Times of Jobs from each User Group

User Group	Probability Distribution of Inter-arrival Times of Jobs	Mean
1	Exponential	3200 Seconds
2	Exponential	640 Seconds
3	Exponential	1600 Seconds
4	Exponential	266.67Seconds

The probability distribution of job execution times is given in Table 16 for each service facility.

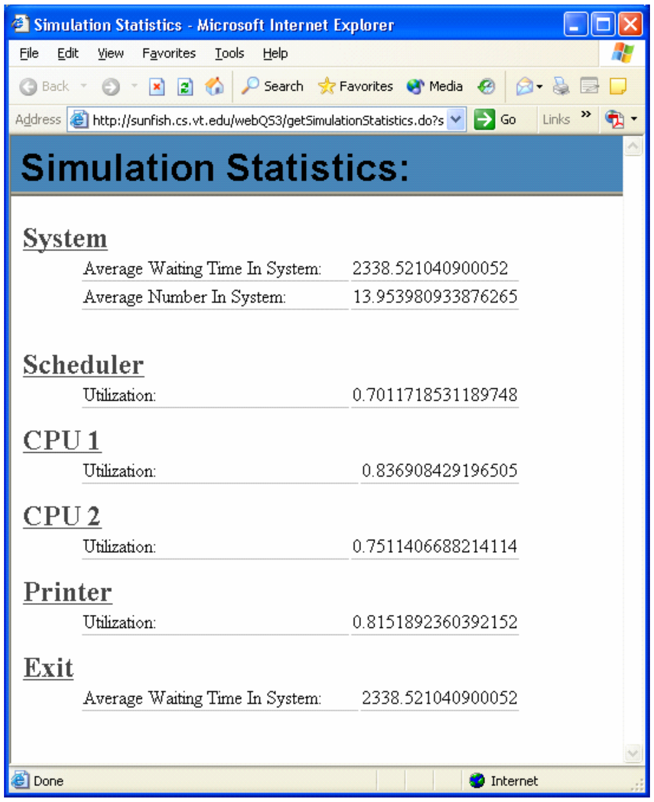
Table 16. Probability Distribution of Job Execution Times for Each Facility

Facility	Probability Distribution of Job Execution Times	Mean
Scheduler	Exponential	112 Seconds
CPU1	Exponential	226.67 Seconds
CPU2	Exponential	300 Seconds
Printer	Exponential	160 Seconds

As depicted in Figure 44, we quickly composed a model of the computer system by using the reusable model components provided by WebQS3. Move times are assigned to the dynamic objects representing the jobs to be able to view a visualization or animation of the simulation. After the model is verified by viewing the visualization, the move times are set to zero, and experiments are performed to estimate the following performance measures:

- $\rho_{scheduler}$ = Scheduler Utilization
- ρ_{CPU1} = CPU1 Utilization
- ρ_{CPU2} = CPU2 Utilization
- $\rho_{printer}$ = Printer Utilization
- W = Average Waiting Time of Jobs in the System
- L = Average Number of Jobs in the System

The *Method of Replications* [Banks et al. 2001; Law and Kelton 2000] is used to perform the simulation experiments. In a simulation run, the model is warmed up by running the simulation for 3000 jobs departing the system, after which the model is run for 15,000 jobs departing the system in steady state. The average of the 15,000 correlated observations in steady state for a performance measure is taken as an independent observation. The simulation run is replicated 20 times using different random number seeds to obtain 20 independent observations for a performance measure. The user-supplied random number seed is multiplied by the replication number to obtain a different seed for each replication. The average of the 20 independent observations is taken as the point estimate of a performance measure. The WebQS3 simulation experimentation results are shown in Figure 45.


Simulation Statistics - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://sunfish.cs.vt.edu/webQS3/getSimulationStatistics.do?s Go Links
Done
Internet

Simulation Statistics:

System

Average Waiting Time In System:	2338.521040900052
Average Number In System:	13.953980933876265

Scheduler

Utilization:	0.7011718531189748

CPU 1

Utilization:	0.836908429196505

CPU 2

Utilization:	0.7511406688214114

Printer

Utilization:	0.8151892360392152

Exit

Average Waiting Time In System:	2338.521040900052

Figure 45. WebQS3 Simulation Experimentation Results for the Computer System

Table 17 shows the WebQS3 comparison testing results for the computer system.

Table 17. WebQS3 Comparison Testing Results for the Computer System

Performance Measure	Analytical Solution	Estimated Value by WebQS3 Simulation	% Deviation
$\rho_{\text{scheduler}}$	0.70	0.70117185	0.167408
ρ_{CPU1}	0.85	0.83690843	1.540185
ρ_{CPU2}	0.75	0.75114067	0.152089
ρ_{printer}	0.80	0.81518924	1.898655
W	2400.00	2338.52104090	2.561623
L	15.00	13.95398093	6.973460

The percent deviations shown in Table 17 are all reasonable given the variance of each performance measure. Hence, our confidence is increased in WebQS3 having sufficient accuracy.

Chapter 8: Conclusions and Future Research

8.1 Conclusions

The development of the WebQS3 system documents that simulation visualization using web-standards and readily available Internet tools is possible, at least in the scope of queue simulations. The architecture is readily maintainable due to the decoupled nature of the components relying only on XML as a communication mechanism between the components. The system includes extension points at every architecture component, providing a means of extending the system at a later date to include new simulation objects or additional functionality. The development of a WYSIWYG simulation model composer using only web browser technology increases the availability of WBS system to a wider audience of users while also making the application easier to use for novice users. The design of an XML-based model representation format that can easily be converted into executable model code makes the execution environment platform independent by relying on readily available parsing tools. XML allows the users to work in a heterogeneous environment where the platform and implementation language of the client and the server are irrelevant. The use of an asynchronous communication protocol frees the client from relying on the Java Runtime Environment and allows more flexibility in the client implementation. The ideas demonstrated through the development of the WebQS3 system go against may of the development paradigms of current WBS systems demonstrating that there are alternative development methods to the standard Java-based approaches.

The creation of the WebQS3 system provides a good starting point for future research in the area of WBS. WebQS3 was built as a demonstration of the ideas developed in this thesis. Many of the ideas expressed have merit, but need more investigation and more robust implementations to allow distribution of the WebQS3 system on a wide scale. Much more usability testing needs to be performed on a cross-section of possible users allowing for the system interfaces to be refined for specific users. Also, many of the architecture components may be enhanced to provide better performance, functionality, and stability. A plethora of further work is available for those that wish to undertake the effort.

8.2 Contributions

This thesis demonstrates several contributions to the web-based simulation field. The contributions include:

- Development of an extensible component-based M&S environment
- Design of a standards-based architecture for a WBS system wherein simulation execution is performed on the server and visualization takes place on the client
- Development of WYSIWYG model composition tools using only an SVG web browser interface

- Creation of a web-based client-server concurrent visualization system using web-standards technologies such as ECMAScript and SVG, which are available in any modern web browser, as an implementation mechanism.
- Design of an XML-based simulation model representation format
- Development of XML conversion utilities to convert an XML simulation model representation to executable code
- Design of an asynchronous communication protocol to communicate simulation state between client and server utilizing standard mechanisms
- Design of a lightweight XML communication format for describing changes in simulation state

8.3 Future Research

Currently, very few web-based simulation servers offer services via an asynchronous protocols and web-standards development. With most simulation servers relying on a synchronous protocol, all the client visualization tools available are built using Java, and dependent on the JRE. The development of a new standards-based system utilizing asynchronous communication protocols is necessary. The creation of a new server-visualizer system will open access to a wider audience of potential users that do not have access to high-end client workstations.

8.3.1 Interactive Simulation Server

Once a simple asynchronous server has been developed, it is necessary to take the next step and make it interactive for the user. Many interactive simulation systems have been developed, such as Diablo II [Blizzard Entertainment, 2003], Ultima Online [Origin, 2003], or The Sims Online [EA Games, 2003]. The market is rich with interactive networked video games (interactive simulations) requiring specialized software and hardware. Much research has been focused on developing these specialized gaming systems, but few resources have been devoted to developing a generalized simulation environment for anyone to use. Developing a generalized interactive simulation environment will provide basis points for game developers as well as simulation professionals creating distributed simulations and providing an interactive web-front end to the simulation.

8.3.2 Asynchronous Visualizer Extensions

As stated earlier, many of the web-based simulation servers required the users to have a JRE available to run the simulation visualization. By eliminating the need for a custom communication protocol and rely on standard mechanisms a new browser-based visualizer can be created. The important aspect of an asynchronous visualizer is the message request algorithm. An adaptable algorithm needs to be created, which allows the visualizer to have access to the needed visualization messages while avoiding buffer under/overflows. An efficient algorithm allows better resource management to occur on both the client and the server portions of the system, while still providing a fluid experience for the user. Additionally, the visualizer needs to be extended to allow the use of images instead of a simple box and test display. The original prototypes for the system called for such image inclusion, but ultimately a means of allowing users to specify the images in an extensible and usable way could not be determined within the time constraints of the project.

Also, many of the tools available for implementing standards-based visualizer designs need to mature to become commercially feasible, especially in the area of SVG and integration with XForms. Loads of resources are devoted to creating custom visualization systems for very specific users and purposes. By creating a generic (standards-based) base engine for all the visualization engines development of specialized engines can be enhanced by relying on a standard and well tested foundation. Research should be devoted to creating a set of standard tools for creating WBS visualization engines in SVG and related web-standard technologies.

8.3.3 Visualizer Usability Testing and Enhancement

The current usability features built into WebQS3 provide a good stating point for future designers. Many of the features are helpful, but many also need further development. Usability testing beyond the initial pilot tests should be conducted to determine if the included features enhance the product, or hinder overall user experience. Also, as functionality is added to tool, usability consideration and design practices should be considered to create a user-centered product.

8.3.4 Simulation Component Creation Framework

The WebQS3 system provides extension mechanisms in every component of the architecture. Creating new components to enhance the functionality of the system requires the creation of new executable simulation components as well as XML meta-data descriptions of the component interfaces. Creating a framework that aids component developers develop new simulation objects would ease the burden on the developer. Developing a simple component logic description language (or family of description languages) and an accompanying set of tools would foster component development. Sending the developers' logic description through a code generator could create the necessary executable code as well as the meta-data descriptions of the components without the users' knowledge, insulating the developer from the backend implementation language and making components more portable.

Appendix A: XML Simulation Model Schema

A.1   Simulation Module

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by David Myers
(Virginia Tech) -->
<xsd:schema targetNamespace="http://websim.cs.vt.edu"
xmlns:simulation="http://websim.cs.vt.edu"
xmlns:basic="vt.simulation.component.basic"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:sch="http://www.ascc.net/xml/schematron"
elementFormDefault="qualified">
 <!-- Component Modules -->
 <xsd:import namespace="vt.simulation.random" schemaLocation="random.xsd"/>
 <xsd:import namespace="vt.simulation.component.basic"
  schemaLocation="basic.xsd"/>
  <xsd:import namespace="vt.simulation.component.queue"
  schemaLocation="queue.xsd"/>
 <!-- Simulation wrapper code -->
 <xsd:annotation>
  <xsd:appinfo>
   <!-- Define schematron namespaces -->
   <sch:ns uri="vt.simulation.component.basic" prefix="basic"/>
   <sch:ns uri="http://websim.cs.vt.edu" prefix="simulation"/>
   <sch:pattern name="Every Model element is required to have a view">
    <sch:rule context="simulation:views/simulation:view">
     <sch:key name="viewKey" path="@for" />
    </sch:rule>
    <sch:rule context="simulation:model/*">
     <sch:assert test="key('viewKey', @id)">A View is required for the
       simulation element</sch:assert>
    </sch:rule>
   </sch:pattern>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:element name="simulation">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="head">
     <xsd:annotation>
      <xsd:documentation>Meta data associated with the
         simulation</xsd:documentation>
     </xsd:annotation>
     <xsd:complexType>
      <xsd:all>
       <xsd:element name="title" type="xsd:string" />
       <xsd:element name="warm-up" type="xsd:double" minOccurs="0" />
       <xsd:element name="steady-state" type="xsd:double" minOccurs="0" />
       <xsd:element name="replications" type="xsd:integer" minOccurs="0" />
       <xsd:element name="initial-seed" type="xsd:long" minOccurs="0" />
       <xsd:any minOccurs="0" maxOccurs="unbounded"
          processContents="lax" />
      </xsd:all>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="model">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:annotation>
          <xsd:documentation>Allow model components to be added to the
              model</xsd:documentation>
        </xsd:annotation>
        <xsd:element ref="basic:Component" maxOccurs="unbounded"
             minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="views">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:element name="view">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:any maxOccurs="unbounded" processContents="lax"/>
            </xsd:sequence>
            <xsd:attribute name="for" type="xsd:IDREF" use="required"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:unique name="viewRef">
      <xsd:selector xpath="./simulation:view"/>
      <xsd:field xpath="@for"/>
    </xsd:unique>
  </xsd:element>
  <xsd:element name="stats-collection">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="system-collect" minOccurs="0"
              type="simulation:StatCollectionType">
          <xsd:annotation>
            <xsd:documentation>Statistic collection properties for the
                entire system</xsd:documentation>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="collect" minOccurs="0" maxOccurs="unbounded">
          <xsd:annotation>
            <xsd:documentation>Specifies which statisitics to collect for
                a given element</xsd:documentation>
          </xsd:annotation>
          <xsd:complexType>
            <xsd:complexContent>
              <xsd:extension base="simulation:StatCollectionType">
                <xsd:attribute name="for" type="xsd:IDREF"
                     use="required"/>
              </xsd:extension>
            </xsd:complexContent>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:unique name="statRef">
      <xsd:selector xpath="./simulation:collect"/>
```

```
        <xsd:field xpath="@for"/>
       </xsd:unique>
      </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="StatCollectionType">
    <xsd:sequence>
     <xsd:element name="displayName" type="xsd:string" />
     <xsd:element name="statRequest" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
       <xsd:simpleContent>
        <xsd:extension base="xsd:boolean">
         <xsd:attribute name="name">
          <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Utilization" />
            <xsd:enumeration value="Average Waiting Time" />
            <xsd:enumeration value="Average Number" />
            <xsd:enumeration value="Average Waiting Time In System" />
            <xsd:enumeration value="Average Number In System" />
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute>
        </xsd:extension>
       </xsd:simpleContent>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

A.2 Random Variate Module

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by David Myers
(Virginia Tech) -->
<xsd:schema targetNamespace="vt.simulation.random"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:random="vt.simulation.random"
xmlns:sch="http://www.ascc.net/xml/schematron"
elementFormDefault="qualified">
  <xsd:complexType name="RandomVariateStream">
    <xsd:annotation>
     <xsd:documentation>A random variate stream</xsd:documentation>
    </xsd:annotation>
    <xsd:choice>
      <xsd:element name="ExponentialRandom"
      type="random:ExponentialRandomType"/>
     <xsd:element name="NormalRandom" type="random:NormalRandomType"/>
     <xsd:element name="UniformRandom" type="random:UniformRandomType"/>
     <xsd:element name="ConstantRandom" type="random:ConstantRandomType" />
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="ExponentialRandomType">
```

```
    <xsd:annotation>
       <xsd:documentation>An Exponential Random variate stream
       representation</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
     <xsd:element name="mean" type="random:positiveDouble"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="NormalRandomType">
    <xsd:annotation>
       <xsd:documentation>A Normal Random variate stream
       representation</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
     <xsd:element name="mean" type="random:positiveDouble"/>
     <xsd:element name="stdDev" type="random:positiveDouble"/>
     <xsd:element name="truncated" type="xsd:boolean" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="UniformRandomType">
    <xsd:annotation>
       <xsd:documentation>A Uniform Random variate stream
       representation</xsd:documentation>
     <xsd:appinfo>
       <sch:ns uri="vt.simulation.random" prefix="random"/>
       <sch:pattern name="Upper bound must be greater than lower bound">
         <sch:rule context="random:UniformRandom">
             <sch:assert test="number(random:upperBound) &gt;
             number(random:lowerBound)">Upper bound must be less than the lower
             bound</sch:assert>
         </sch:rule>
       </sch:pattern>
     </xsd:appinfo>
    </xsd:annotation>
    <xsd:sequence>
     <xsd:element name="lowerBound" type="random:positiveDouble"/>
     <xsd:element name="upperBound" type="random:positiveDouble"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ConstantRandomType">
    <xsd:annotation>
       <xsd:documentation>A Constant Random variate stream
       representation</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
     <xsd:element name="value" type="random:positiveDouble" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="positiveDouble">
    <xsd:restriction base="xsd:double">
     <xsd:minInclusive value="0" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

A.3 Basic Component Module

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by David Myers
(Virginia Tech) -->
<xsd:schema targetNamespace="vt.simulation.component.basic"
xmlns:sch="http://www.ascc.net/xml/schematron"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:random="vt.simulation.random"
xmlns:basic="vt.simulation.component.basic" elementFormDefault="qualified">
 <!-- Module import -->
 <xsd:import schemaLocation="random.xsd" namespace="vt.simulation.random"/>
 <!-- Type Definition -->
 <xsd:complexType name="ReferenceType">
  <xsd:annotation>
   <xsd:appinfo>
    <sch:pattern name="ids must be unique">
     <sch:rule context="simulation:model/*">
      <sch:key name="id" path="@id" />
      <sch:assert test="count(key('id', @id)) = 1">The given id is
         already in use</sch:assert>
     </sch:rule>
    </sch:pattern>
   </xsd:appinfo>
   <xsd:documentation>A reference to another simulation
    object</xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name="ref" type="xsd:IDREF" use="required"/>
 </xsd:complexType>
 <xsd:complexType name="StaticObjectType" abstract="true">
  <xsd:annotation>
   <xsd:documentation>Simple simulation object that can be placed in the
    simulation</xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name="id" type="xsd:ID" use="required"/>
 </xsd:complexType>
 <xsd:complexType name="NextPlaceType">
  <xsd:annotation>
   <xsd:appinfo>
    <sch:ns uri="vt.simulation.component.basic" prefix="basic"/>
    <sch:pattern name="NextPlace element is weights sum to 1">
     <sch:rule context="*[basic:place]">
      <sch:assert test="not(@weighted) or @weighted = 'false' or
         (@weighted = 'true' and sum(basic:place/@weight) = 1)">Weighted
         next place specifications must sum to 1</sch:assert>
     </sch:rule>
    </sch:pattern>
   </xsd:appinfo>
   <xsd:documentation>Specification of next place
    elements</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
   <xsd:element name="place" maxOccurs="unbounded">
    <xsd:annotation>
     <xsd:appinfo>
      <sch:ns uri="vt.simulation.component.basic" prefix="basic"/>
```

```
        <sch:pattern name="Place elements are weighted?">
         <sch:rule context="basic:place">
          <sch:report test="(not(../@weighted) or ../@weighted = 'false')
              and  @weight">The weight attribute is unnecessary and will be
              ignored because the next place decision is not
              weighted</sch:report>
          <sch:assert test="not(../@weighted) or ../@weighted = 'false'
              or (../@weighted = 'true' and @weight)">If the next place is
              weighted then each place much have a weight</sch:assert>
         </sch:rule>
        </sch:pattern>
        <sch:pattern name="Every reference is only made once in a
          specification">
         <sch:rule context="basic:place">
          <sch:assert test="count(../basic:place[@ref = current()/@ref])
              = 1">Only a single instance of every reference is
              allowed</sch:assert>
         </sch:rule>
        </sch:pattern>
       </xsd:appinfo>
       <xsd:documentation>A place that the simulation element may send
         dynamic objects to</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
       <xsd:attribute name="weight" type="basic:weight" use="optional"/>
       <xsd:attribute name="ref" type="xsd:IDREF" use="required"/>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="weighted" type="xsd:boolean" use="optional"
      default="false">
     <xsd:annotation>
      <xsd:documentation>If set to true the next place elements are provided
        a weight in evaluating the next location to move
        to</xsd:documentation>
     </xsd:annotation>
    </xsd:attribute>
   </xsd:complexType>
   <xsd:complexType name="ContainerType">
    <xsd:annotation>
     <xsd:documentation>Simple simulation container
       element</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
     <xsd:extension base="basic:StaticObjectType">
      <xsd:sequence>
       <xsd:element name="moveTimeToNextPlaceVariate"
         type="random:RandomVariateStream"/>
       <xsd:element name="nextPlaceSpecification"
         type="basic:NextPlaceType"/>
      </xsd:sequence>
     </xsd:extension>
    </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ExitObjectType">
    <xsd:annotation>
     <xsd:documentation>Simple simulation container
```

```
      element</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="basic:StaticObjectType"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="weight">
    <xsd:restriction base="xsd:double">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="1"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="Component" type="basic:StaticObjectType"
   abstract="true"/>
  <xsd:element name="ExitObject" type="basic:ExitObjectType"
   substitutionGroup="basic:Component"/>
</xsd:schema>
```

A.4 Queue Component Module

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by David Myers
(Virginia Tech) -->
<xsd:schema targetNamespace="vt.simulation.component.queue"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:sch="http://www.ascc.net/xml/schematron"
xmlns:random="vt.simulation.random"
xmlns:basic="vt.simulation.component.basic"
xmlns:queue="vt.simulation.component.queue" elementFormDefault="qualified">
  <!-- Module Imports -->
  <xsd:import namespace="vt.simulation.random" schemaLocation="random.xsd"/>
  <xsd:import namespace="vt.simulation.component.basic"
   schemaLocation="basic.xsd"/>
  <xsd:annotation>
    <xsd:appinfo>
      <!-- Define schematron namespaces -->
      <sch:ns uri="vt.simulation.random" prefix="random"/>
      <sch:ns uri="vt.simulation.component.basic" prefix="basic"/>
      <sch:ns uri="vt.simulation.component.queue" prefix="queue"/>
      <sch:ns uri="http://websim.cs.vt.edu" prefix="sim"/>
      <sch:pattern name="A generator is required">
        <sch:rule context="sim:model">
          <sch:assert test="queue:Generator">A Generator is required in a
             queue simulation</sch:assert>
        </sch:rule>
      </sch:pattern>
      <sch:pattern name="An Exit is required">
        <sch:rule context="sim:model">
          <sch:assert test="basic:ExitObject">An Exit Point is required in a
             queue simulation</sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
```

```
<!-- Type Definition -->
<xsd:complexType name="GeneratorType">
 <xsd:complexContent>
  <xsd:extension base="basic:ContainerType">
   <xsd:sequence>
    <xsd:element name="intergenerationTime"
       type="random:RandomVariateStream"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ServerType">
 <xsd:complexContent>
  <xsd:extension base="basic:ContainerType">
   <xsd:sequence>
    <xsd:element name="serviceTime" type="random:RandomVariateStream"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="WaitingLineType">
 <xsd:complexContent>
  <xsd:extension base="basic:ContainerType"/>
 </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Generator" type="queue:GeneratorType"
   substitutionGroup="basic:Component">
 <xsd:annotation>
  <xsd:appinfo>
   <sch:pattern name="Check Generator References">
    <sch:rule
        context="queue:Generator/basic:nextPlaceSpecification/basic:place">
     <sch:assert test="name(key('id', @ref)) =
        'queue:WaitingLine'">Generator can only reference Waiting Lines as
        a next place.   The Generator is referencing a &apos;<sch:name
        path="key('id', @ref)" />&apos;</sch:assert>
    </sch:rule>
   </sch:pattern>
  </xsd:appinfo>
  <xsd:documentation>Queue System Generator element</xsd:documentation>
 </xsd:annotation>
</xsd:element>
<xsd:element name="Server" type="queue:ServerType"
   substitutionGroup="basic:Component">
 <xsd:annotation>
  <xsd:appinfo>
   <sch:pattern name="Check Server References">
    <sch:rule
        context="queue:Server/basic:nextPlaceSpecification/basic:place">
     <sch:assert test="(name(key('id', @ref)) = 'queue:WaitingLine') or
        (name(key('id', @ref)) = 'basic:ExitObject')">Servers can only
        reference Waiting Lines or Exit Objects as a next place.  The
        Server is referencing a &apos;<sch:name path="key('id',
        @ref)"/>&apos;</sch:assert>
    </sch:rule>
    <sch:rule context="queue:Server/basic:nextPlaceSpecification">
     <sch:assert test="@weighted = 'true' or
```

```
            not((count(basic:place[name(key('id', @ref)) =
            'queue:WaitingLine']) &gt; 0) and (count(basic:place[name(key('id',
            @ref)) = 'basic:ExitObject']) &gt; 0))">A server can only reference
            an Exit and a Queue if the determination is weighted.</sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
    <xsd:documentation>Queue System Server element</xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:element name="WaitingLine" type="queue:WaitingLineType"
   substitutionGroup="basic:Component">
  <xsd:annotation>
    <xsd:appinfo>
      <sch:pattern name="Check Waiting Line References">
        <sch:rule
         context="queue:WaitingLine/basic:nextPlaceSpecification/basic:place">
          <sch:assert test="(name(key('id', @ref)) = 'queue:Server')">Waiting
            Lines can only reference Servers as a next place.  The Waiting Line
            is referencing a &apos;<sch:name path="key('id', @ref)"
            />&apos;</sch:assert>
        </sch:rule>
        <sch:rule context="queue:WaitingLine/basic:nextPlaceSpecification">
          <sch:assert test="not(@weighted) or @weighted = 'false'">Waiting
            Line next place specification are not allowed to be
            weighted</sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
    <xsd:documentation>Queue System Waiting Line (Queue)
     element</xsd:documentation>
  </xsd:annotation>
</xsd:element>
</xsd:schema>
```

Appendix B: Sample XML Simulation Model

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by David Myers
(Virginia Tech) -->
<simulation:simulation xmlns:simulation="http://websim.cs.vt.edu"
xmlns:svg="http://www.w3.org/2000/svg"
xmlns:queue="vt.simulation.component.queue"
xmlns:basic="vt.simulation.component.basic"
xmlns:random="vt.simulation.random"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://websim.cs.vt.edu ../schema/simulation.xsd">
 <simulation:head>
  <simulation:title>Sample M/M/1 Queue System</simulation:title>
 </simulation:head>
 <simulation:model>
  <queue:Generator id="generator">
   <basic:moveTimeToNextPlaceVariate>
    <random:NormalRandom>
     <random:mean>5</random:mean>
     <random:stdDev>3</random:stdDev>
     <random:truncated>true</random:truncated>
    </random:NormalRandom>
   </basic:moveTimeToNextPlaceVariate>
   <basic:nextPlaceSpecification weighted="true">
    <basic:place weight="1.0" ref="queue" />
   </basic:nextPlaceSpecification>
   <queue:intergenerationTime>
    <random:UniformRandom>
     <random:lowerBound>0</random:lowerBound>
     <random:upperBound>10</random:upperBound>
    </random:UniformRandom>
   </queue:intergenerationTime>
  </queue:Generator>
  <queue:WaitingLine id="queue">
   <basic:moveTimeToNextPlaceVariate>
    <random:ExponentialRandom>
     <random:mean>10</random:mean>
    </random:ExponentialRandom>
   </basic:moveTimeToNextPlaceVariate>
   <basic:nextPlaceSpecification>
    <basic:place ref="server" />
   </basic:nextPlaceSpecification>
  </queue:WaitingLine>
  <queue:Server id="server">
   <basic:moveTimeToNextPlaceVariate>
    <random:UniformRandom>
     <random:lowerBound>0</random:lowerBound>
     <random:upperBound>10</random:upperBound>
    </random:UniformRandom>
   </basic:moveTimeToNextPlaceVariate>
   <basic:nextPlaceSpecification>
    <basic:place ref="exit" />
   </basic:nextPlaceSpecification>
   <queue:serviceTime>
```

```
      <random:ConstantRandom>
        <random:value>15</random:value>
      </random:ConstantRandom>
    </queue:serviceTime>
  </queue:Server>
  <basic:ExitObject id="exit"/>
 </simulation:model>
 <simulation:views>
  <simulation:view for="generator">
   <svg:svg height="28" width="75">
    <svg:rect class="selectable" x="0" y="0" width="70" height="25"
fill="yellow" stroke="black" stroke-width="0.5"/>
    <svg:text x="5" y="17" font-size="10pt">Generator</svg:text>
    <svg:rect class="startPoint" x="67" y="9.5" width="6" height="6"
fill="black" />
   </svg:svg>
  </simulation:view>
  <simulation:view for="server">
   <svg:svg height="28" width="75">
    <svg:rect class="selectable" x="0" y="0" width="70" height="25"
fill="yellow" stroke="black" stroke-width="0.5"/>
    <svg:text x="5" y="17" font-size="10pt">Server</svg:text>
    <svg:rect class="startPoint" x="67" y="9.5" width="6" height="6"
fill="black" />
   </svg:svg>
  </simulation:view>
  <simulation:view for="queue">
   <svg:svg height="28" width="75">
    <svg:rect class="selectable" x="0" y="0" width="70" height="25"
fill="yellow" stroke="black" stroke-width="0.5"/>
    <svg:text x="5" y="17" font-size="10pt">ssQueue</svg:text>
    <svg:rect class="startPoint" x="67" y="9.5" width="6" height="6"
fill="black" />
   </svg:svg>
  </simulation:view>
  <simulation:view for="exit">
   <svg:svg height="28" width="52">
    <svg:rect x="0" y="0" width="50" height="25" fill="yellow"
stroke="black" stroke-width="0.5" />
    <svg:text x="5" y="17" font-size="11pt" style="pointer-events:
none">Exit</svg:text>
   </svg:svg>
  </simulation:view>
 </simulation:views>
 <simulation:stats-collection xmlns:simulation="http://websim.cs.vt.edu">
  <simulation:system-collect>
   <simulation:displayName>Server system</simulation:displayName>
   <simulation:statRequest name="Average Number In
   System">true</simulation:statRequest>
  </simulation:system-collect>
  <simulation:collect for="server">
   <simulation:displayName>Server 1</simulation:displayName>
   <simulation:statRequest name="Average Number In
   System">true</simulation:statRequest>
  </simulation:collect>
  <simulation:collect for="queue">
   <simulation:displayName>Queue 2</simulation:displayName>
```

```
    <simulation:statRequest name="Average Waiting
    Time">1</simulation:statRequest>
    <simulation:statRequest name="Average
    Number">true</simulation:statRequest>
   </simulation:collect>
  </simulation:stats-collection>
</simulation:simulation>
```

Bibliography

Adobe (2004), "Web Center Features – SVG – Manual Download." Retrieved 10 January, 2004. Available: http://www.adobe.com/svg/viewer/install/main.html

Alfonseca, M, J. deLara, and H. Vangheluwe (2001), "Web-Based Simulation of Systems Described by Partial Differential Equations," In *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA, pp. 629 – 636.

Alur, D., J. Crupi, and D. Malks (2001), *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice-Hall, Palo Alto, CA.

Apache (2004), "Apache Struts project", Retrieved 18 January, 2004. Available: http://jakarta.apache.org/struts.

Arena (2003), "Arena Software," Retrieved 1 May, 2004. Available: http://www.arenasimulation.com/

Balci, O. (1988), "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages," In *Proceedings of the 1988 Winter Simulation Conference* (San Diego, CA, Dec. 12-14). IEEE, Piscataway, NJ, pp. 287-295.

Balci, O. (1998), "Verification, Validation, and Testing," In *The Handbook of Simulation*, J. Banks (ed.), John Wiley & Sons, New York, NY, Aug., pp. 335-393.

Balci, O. (2003), "Verification, Validation, and Certification of Modeling and Simulation Applications," In *Proceedings of the 2003 Winter Simulation Conference* (New Orleans, LA, Dec. 7-10). IEEE, Piscataway, NJ, pp. 150-158.

Banks, J., J.S. Carson, B.L. Nelson, and D.M. Nicol (2001), *Discrete-Event System Simulation*, Third Edition, Prentice-Hall, Upper Saddle River, NJ.

Berger, M. and U. Leiner. "Remote Visualisieren und Manipulieren von Simulationen im Internet." In Proceedings if Simulation and Animation 1997 Magdeburg, 1-11. Society for Computer Simulation Europe, Ghent, Belgium.

Blizzard Entertainment (2003), "Blizzard Entertainment - Diablo II", Retrieved 20 Nov, 2003. Available: http://www.blizard.com/diablo2/.

Castor (2004), "The Castor Project." Retrieved 29 March, 2004. Available: http://www.castor.com

Chandrasekaran, S., G. Silver, J. A. Miller, J. Cardoso, and A. P. Sheth (2002), "Web Service Technologies and Their Synergy with Simulation," In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA, pp. 606-615.

CSS (2004), "Cascading Style Sheets", Retrieved 29 January, 2004. Available: http://www.w3.org/Style/CSS

DataPower (2004), "DataPower: XS40 XML Security Gateway", Retrieved 18 April, 2004. Available: http://www.datapower.com/products/xs40.html

Davis, P., P. Fishwick, C. Overstreet, and C. Pegden (2000), "Model Composability as a Research Investment: Responses to the Featured Paper", In *Proceedings of the 2000 Winter Simulation Conference*, pp. 1585 – 1591.

De Lara, J. and M. Alfonseca. "Constructing Simulation-Based Web Documents." IEEE Multimedia, Volume 8, Issue 1. January – March. Pages: 42 – 49.

Digester (2004), "Digester – Commons." Retrieved 29 March, 2004. Available: http://jakarta.apache.org/commons/digester/

DOM (2004), "W3C Document Object Model." Retrieved 18 February, 2004. Available: http://www.w3.org/DOM

DOM2 (2000), "Document Object Model (DOM) Level 2 Core Specification." Retrieved 10 January, 2004. Available: http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/

EA Games (2003), "The Sims." Retrieved 20 Nov, 2003. Available: http://thesims.ea.com.

Fishwick, P. A. (2002), "Using XML for Simulation Modeling," In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA, pp. 616-622.

Fontana, J. (2004). "Vendors to Target XML Traffic Jam." *Network World*. March 2004.

Fujimoto, R. M. (2000), *Parallel and Distributed Simulation Systems*, Wiley Interscience.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1994), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

Gan, B. P., L. Liu, Z. Ji, S. J. Turner, and W. Cai (2001), "Managing Event Traces for a Web Front-End to a Parallel Simulation." In *Proceedings of the 2001 Winter Simulation Conference*, Volume 1, Arlington, VA, pp: 637 – 644.

Gross, D. and C. M. Harris (1974), *Fundamentals of Queueing Theory*, John Wiley & Sons, New York, NY.

Healy K. J. and Kilgore, R. A. (1997). "Silk: A Java Based Process Simulation Language." In *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA, pp. 475 – 482.

Howell, F. and R. McNab (1998). "Simjava: A Discrete Event Simulation Library for Java." In *Proceedings of the 1998 International Conference on Web-Based Modeling and Simulation*. San Diego, CA.

Howell, Fred and R. McNabb. (2003). "SimJava". Retrieved 12 Nov, 2003. Available: http://www.dcs.ed.ac.uk/home/hase/simjava/.

Huang J., C. Fang-Tsou, and J. Chang (1998), "A Multiuser 3D Web Browsing System", IEEE Computing, 2, 5, 70-79, 1998.

JARV (2004), "JARV." Retrieved 5 March, 2004. Available: http://iso-relax.sourceforge.net/JARV/.

JAXB (2003), "Java Architecture for XML Binding (JAXB)." Retrieved 29 March, 2004. Available: http://java.sun.com/developer/technicalArticles/WebServices/jaxb/

JAXB (2004), "Java Architecture for XML Binding (JAXB)." Retrieved 29 March, 2004. Available: http://java.sun.com/xml/jaxb/index.jsp

JMS (2004), "Java Message Service." Retrieved 1 March, 2004. Available: http://java.sun.com/products/jms

Kaptelinin, V., B. Nardi, and C. Macaulay (1999), "The Activity Checklist: A Tool for Representing the "Space" of Context." Interactions, July and August, 1999, 27-39.

Kilgore, R. A. (2001a). "Open-Source Simulation Modeling Language (SML)." In *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA, pp. 607 – 613.

Kilgore, R. A. (2001b). "Open-source SML and Silk for Java-Based, Object-Oriented Simulation." In *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA, pp. 262 – 268.

Kilgore, R.A. (2002), "Simulation Web Services with .Net technologies," In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA, pp. 841-846.

Kim, T., J. Lee, and P. Fishwick (2002), "A Two-Stage Modeling and Simulation Process for Web-Based Modeling and Simulation." ACM Transactions on Modeling and Computer Simulation, Vol. 12, No. 3, July 2002, pp. 230 – 248.

Klein, U., S. Strassburger, and J. Beikirch (1998), "Distributed Simulation with JavaGPSS Based on the High Level Architecture." In *Proceedings of the 1998 International Conference on Web-Based Modeling and Simulation*. San Diego, CA. pp. 85-90.

Kuljis, J. and R. Paul (2003), "Web-Based Discrete Event Simulation Models: Current States and Possible Futures." *Simulation and Gaming*, Vol. 34, No. 1, March 2003, pp. 39-53.

Law, A. M. and W. D. Kelton (2000), *Simulation Modeling and Analysis*, Third Edition, McGraw-Hill, New York, NY.

Lorenz, P. and K. Ritter (1997), "Skopeo – A Platform Independent System Animation for the W3." In Proceedings of the Simulation and Automation '97. Magdeburg. Pages 12-23.

Lorenz, P., H. Dorwarth, and K. C. Ritter (1997), "Towards a Web Based Simulation Environment." In *Proceedings of the 1997 Winter Simulation Conference*. Pages: 1338 – 1344.

Marinescu, F. (2002), *EJB Design Patterns*, John Wiley & Sons, New York, NY.

Microsoft (2004a), "The .NET Framework," Microsoft Corporation, http://www.microsoft.com/net/

Microsoft (2004b), "Internet Explorer Home Page." Retrieved 18 April, 2004. Available: http://www.microsoft.com/windows/ie/default.asp

Miller, J.A. (2003), "Jsim: A Java-Based Simulation and Animation Environment." Retrieved 12 Nov, 2003. Available: http://chief.cs.uga.edu/~jam/jsim/

Mullet, K. and D. Sano (1995), *Designing Visual Interfaces: Communication Oriented Techniques*, Englewood Cliffs, NJ: Sunsoft Press.

Nielsen, J. (1992), “Finding Usability Problems through Heuristic Evaluation.” CHI 1992

Nielsen, J. (1994), “Heuristic Evaluation,” In *Usability Inspection Methods*, eds. J. Nielsen and R.L. Mack. New York: John Wiley.

Newman, W. (1997), “The Use of Critical Parameters in the Design of Web-Based Interactive Systems.” Presented at *Time and the Web*, 19 June 1997.

Norman, D.A. (1986), “Cognative Engineering.” In *User Centered System Design*, eds. D.A. Norman and S.D. Draper, 31-61. Hillsdale, NJ: Lawrence Erlbaum Associates.

Origin (2003), “ORIGIN – Ultima Online,” Retrieved 20 Nov, 2003. Available: http://www.uo.com.

Page (1999), “A Survey of Web-Based Simulation,” Retrieved 10 February, 2004. Available: http://www.thesimguy.com/Projects/websim/survey/survey.html

Page, E., A. Buss, P. Fishwick, K. Healy, R. Nance, and R. Paul (2000), “Web-Based Simulation: Revolution or Evolution.” *ACM Transactions on Modeling and Computer Simulation*, Vol. 10, No. 1, 2000, pp. 3 – 17.

Pritsker (2003), “MAPICS,” Retrieved 1 May, 2004. Available: http://www.pritsker.com/awesim.asp.

Quick (2002), “Quickpage.” Retrieved 28 March, 2004. Available: http://quar.sourceforge.net/web/2001-12/products/quick/index.html

Relax NG (2004), “OASIS RELAX NG TC.” Retrieved 15 March, 2004. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=realx-ng

Ritter, Klaus-Christoph (1997), “Skopeo.” Retrieved 12 Nov, 2003. Available: http://simos2.cs.uni-magdeburg.de/Skopeo/Ani.html.

Rosson, M.B. and J. Carroll (2002), *Usability Engineering*, San Francisco, CA: Morgan Kaufmann Publishers, 2002.

SAX (2004), “SAX.” Retrieved 10 March, 2004. Available: http://sax.sourceforge.net

Schematron (2004), “Academia Sinica Computing Centre’s Schematron Home Page.” Retrieved 15 March, 2004. Available: http://xml.ascc.net/resource/schematron/schematron.html

Schumann, Marco (2003), “Visualization of Simulation Results on the Internet.” 1997. Online. 12 Nov, 2003. Available: http://www.cs.uni-magdeburg.de/~maschuma/GraphIt/GraphIt.html.

Scriven, M. (1967), “The Methodology of Evaluation,” In *Perspectives of Curriculum Evaluation*, eds. R. Tyler, R. Gagne, and M. Scriven, 39 – 83. Chicago: Rand McNally.

SMIL (2004), “W3C Synchronized Multimedia Home Page.” Retrieved 10 March, 2004. Available: http://www.w3.org/AudioVideo/

SOAP (2003), "SOAP Specification (Primer)", http://www.w3.org/TR/soap12-part0/

St. Amant, R (1998), "Planning and User Interface Affordances," Proceedings of the 4th International Conference on Intelligent User Interfaces, p. 135-142, 1998, Los Angeles, California.

Strassburger, S., T. Schulze, U. Klein, and J. O. Henriksen (1998), "Internet-Based Simulation Using Off-The-Shelf Simulation Tools and HLA. In *Proceedings of the 1998 Winter Simulation Conference*, Volume: 2. Pages: 1669 -1676.

Sun (2004a), "Java 2 Platform, Enterprise Edition (J2EE)." Sun Microsystems, Inc., http://java.sun.com/j2ee/

Sun (2004b), "Java Blueprints Website," Retrieved 28 January, 2004. Available: http://java.sun.com/j2ee/blueprints/

SVG (2004a), "Scalable Vector Graphics (SVG)." Retrieved 16 February, 2004. Available: http://www.w3.org/Graphics/SVG

SVG (2004b), "Scalable Vector Graphics Version 1.2 W3C Working Draft Specification", Retrieved 18 March, 2004. Available: http://www.w3.org/TR/2004/WD-SVG12-20040318/

Tao, L. (2001), "Shifting Paradigms with the Application Service Provider Model," *IEEE Computer*, Vol. 34, No. 10 (Oct.), pp. 32-39.

Tarari (2004a), "Tarari: Products: XML Content Processor." Retrieved 12 April, 2004. Available: http://www.tarari.com/xml/index.html

Tarari (2004b), "Tarari: Products: Random Access XML (RAX) Content Processor." Retrieved 12 April, 2004. Available: http://www.tarari.com/rax/index.html

Vanderdonckt, J., X. Gillo (1994), "Visual Techniques for Traditional and Multimedia Layouts." Proceedings of the Workshop on Advanced Visual Interfaces, p. 95-104, June 01-04, 1994, Bari, Italy.

Veith, T. (1997), "Netsim: A Java(TM)-Based WWW Simulation Package." Thesis. Virginia Tech, 1997.

WAI (2004), "Web Accessibility Initiative (WAI) Home Page." Retrieved 29 March, 2004. Available: http://www.w3.org/WAI

Watzmann, S. (2002), "Visual Design Principles for Usable Interfaces." pp. 263-285.

Whitman, L, . Huff, and S. Palaniswamy (1998), "Commercial Simulation over the Web." In *Proceedings of the 1998 Winter Simulation Conference*, pp. 335 – 339.

Wiedemann, T. (2001), "Simulation Application Service Providing (SIM-ASP)," In *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA, pp. 623-628.

Whittaker,S., L. Terveen, and B. Nardi (2000), "Let's Stop Pushing the Envelope and Start Addressing It: A Reference Task Agenda for HCI." *Human Computer Interaction*, 2000, Volume 15, pp. 76-106.

Xalan (2004), “xmlfilters.gif” Retrieved 10 April, 2004. Available: http://xml.apache.org/xalan-j/images/xmlfilters.gif

XForms (2004), “XForms – The Next Generation of Web Forms.” Retrieved 28 January, 2004. Available: http://www.w3.org/MarkUp/Forms/

XHTML (2004), “W3C HTML Home Page.” Retrieved 15 January, 2004. Available: http://www.w3.org/MarkUp/

XInclude (2004), “XML Inclusions (XInclude) Version 1.0.” Retrieved 20 April, 2004. Available: http://www.w3.org/TR/2004/CR-xinclude-20040413/

XML Infoset (2004), “XML Information Set (Second Edition).” Retrieved 13 March, 2004. Available: http://www.w3.org/TR/xml-infoset/

XML Schema (2004), “W3C XML Schema.” Retrieved 3 February, 2004. Available: http://www.w3.org/XML/Schema

XML (2004), “Extensible Markup Language (XML) 1.0 (Third Edition).” Retrieved 18 March, 2004. Available: http://www.w3.org/TR/2004/REC-xml-20040204/

XPATH (2004), “XML Path Language.” Retrieved 15 January, 2004. Available: http://www.w3.org/TR/xpath

XSLT (2004), “The Extensible Stylesheet Language Family.” Retrieved 8 April, 2004. Available: http://www.w3.org/Style/XSL

VITA

David S. Myers	e-mail: damyers2@vt.edu
Current Address:	**Permanent Address:**
14412 Stroubles Creek Rd, NW	P.O. Box 272
Blacksburg, VA 24060	Annandale, VA 22003
(540) 449-3194	(703) 256-0324

EDUCATION

M.S. Computer Science and Applications, expected June 2004 GPA: 3.9/4.0
Thesis: "Web-Based Queuing System Simulation System"
B.S.E.E Electrical Engineering, Summa Cum Laude, December 2002
B.S. Computer Science, Magna Cum Laude, May 2002
Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, VA
Minors – Mathematics, English Overall GPA: 3.8/4.0; Dean's List every semester

COMPUTER SKILLS

Languages: Java, C++, JavaScript, (X)HTML/CSS, XML, SQL, Visual Basic, VBScript
Libraries and Platforms: J2EE, Servlets/JSP, EJB 1.1/2.0, DHTML (IE, MOZ, DOM), XSLT, XSL-FO, SVG, Swing/AWT
Applications: IBM WebSphere Application Studio Developer, IBM WebSphere Application Server, Macromedia Dreamweaver, MS Office, Apache Web Server

WORK EXPERIENCE

Software Engineer, Research Triangle Park, NC, April 2004 – Present
International Business Machines (IBM)

- Lead tester for Java-based tool allowing functional testing of terminal applications
- Developed and executed test cases to ensure quality of software
- Organized and prioritized testing schedule under tight time constraints
- Aided in development and design of new functionality for tool
- Developed and taught skills transfer material on use of product to group of developers and support personnel.
- Researched and implemented new functionality for maintenance release of product
- Assisted in transfer of codebase to development team on the west coast (transfer from RTP to California labs)

Software Engineer, Blacksburg, VA, August 2001 – February 2004
Orca Computer, Inc

- Worked in a 3-man team to design and implement a collaborative web-enabled software application for performing structured evaluations of complex systems.
- Designed and implemented complex DHTML applications to replace Java Applet functionality.
- Refactored web application to utilize web standards for ease of maintenance and consistency.
- Researched and adapted open-source tools and frameworks to add functionality to the system.
- Assisted with the requirements definition and user interface design of the application.
- Implemented the system using several core Java 2 Enterprise Edition (J2EE) technologies (Java Servlets, JSPs, and EJBs) and traditional web technology (HTML,CSS, JavaScript, SVG, etc.).
- Utilized and extended automated test system based on JUnit to ensure product reliability.
- Deployed product releases to the J2EE application server (IBM Websphere) and administered the production application.

Lead Software Research Assistant, Blacksburg, VA, August 2000 – May 2002
Dr. Robert Hendricks, Dept. of Materials Science and Engineering, Virginia Tech

- Directed a team in developing a stand-alone Java product used to assist students in analyzing complex mathematical and physical phenomena.
- Designed and implemented an extendable GUI based on an XML, Swing, and Java Web Start.
- Researched and tested several toolkits and technologies that could be used for the project

Intern, London, England, July 2001 – August 2001
GE Lighting Europe, Finance Department

- Performed user requirement derivation and analysis through interviews with employees to determine needs of a new financial analysis system
- Designed a prototype for the new system based on requirements gathered.

Intern, Roanoke, VA, May 2000 – August 2000
Meridium, Inc

- Integrated third party product into existing Meridium product suite using COM programming interface.
- Wrote SQL scripts and stored procedures in SQL-Server to incorporate new data stores.

PAPERS

Osman Balci, Robin J. Adams, David S. Myers, and Richard E. Nance (2002), "A Collaborative Evaluation Environment for Credibility Assessment of Modeling and Simulation Applications" In *Proceedings of the 2002 Winter Simulation Conference* (San Diego, CA, Dec. 8-11). IEEE, Piscataway, NJ.

D.S. Myers, J.L. Meyer, P. DePasquale, and R.W. Hendricks
An Interactive Program for Determining Junctions Depths in Diffused Silicon Devices
Proc. 14th IEEE Biennial University/Government/Industry Microelectronics (UGIM) Symposium held in Richmond, Virginia, June 17-20, 2001.

AWARDS

Eagle Scout with 4 palms
Recipient, CS Investment in Excellence Scholarship

ACTIVITIES

Phi Beta Kappa	Upsilon Pi Epsilon, Former President	Solely Swing Club
National Society of Collegiate Scholars	University Honors Program	
Golden Key Honor Society	Association of Computing Machinery	