12-2008

Application of Web Services to a Simulation Framework

Matthew Bennink
Clemson University, mbennin@clemson.edu

Application of Web Services to a Simulation Environment

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Matthew Scott Bennink
December 2008

Accepted by:
Dr. Richard Brooks, Committee Chair
Dr. Adam Hoover
Dr. Christopher Griffin

Abstract

The Joint Semi-Automated Forces (JSAF) simulator is an excellent tool for military training and a great testbed for new SAF behaviors. However, it has the drawback that behaviors must be ported into its own Finite State Machine (FSM) language. Web Services is a growing technology that seamlessly connects service providers to service consumers. This work attempts to merge these two technologies by modeling SAF behaviors as web services. The JSAF simulator is then modeled as a web service consumer.

This approach allows new Semi-Automated Forces (SAF) behaviors to be developed independently of the simulator, which provides the developer with greater flexibility when choosing a programming language, development environment, and development platform. In addition to new SAF behaviors, this approach also supports any external component that can be modeled as a web service. Furthermore, these services are often run over a network, which distributes the computational load across several computers. Finally, hosting copies of a single service on several machines, a concept similar to file-sharing mirrors, offers an environment for load-balancing. This means if several entities are running the same behavior, a single server does not perform the computation for every entity. Instead, each entity is assigned to a specific server, increasing the quality of service seen by the system.

A Web Services framework linking JSAF with several services is designed and implemented. Suppression of Enemy Air Defense (SEAD) behaviors written in MATLAB and a behavior recognition system are integrated with JSAF. These behaviors and the recognition tool were developed by other researchers, independent of this work. Results show that offloading computation to other machines is beneficial, especially when the simulation system is under heavy load. Preliminary results also indicate that load-balancing performs much better than using a single server.

Dedication

I dedicate this work to my loving wife and my family. Without them, I would not be where I am. I am very grateful.

Acknowledgments

I would like to acknowledge my advisor, Dr. Richard Brooks, for his continuing help and constant support of my work. I further acknowledgement thank Dr. Adam Hoover and Dr. Christopher Griffin for being a part of my committee.

This material is based upon work supported by, or in part by, the Office of Naval Research Code 311 contract/grant number N00014-06-C-0022. The authors gratefully acknowledge this support and take responsibility for the contents of this report.

Table of Contents

List of Tables

List of Figures

Acronym Table

ALSP	Aggregate Level Simulation Protocol
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DIS	Distributed Interactive Simulation
HLA	High Level Architecture
JSAF	Joint Semi-Automated Forces
QoS	Quality-of-Service
RPC	Remote Procedure Call
RTI	Run-Time Infrastructure
SAF	Semi-Automated Forces
SEAD	Suppression of Enemy Air Defense
SIMNET	Simulator Networking
SOAP	Simple Object Access Protocol
W3C	World Wide Web Consortium
WSDL	Web Service Description Language

Chapter 1

Introduction

1.1 Motivation

Military forces require extensive training to remain at a strong level of preparedness. However, complete training is not possible, especially in preparation for a war-time setting. Real combat experience requires risking lives and consuming large amounts of resources. Human life should never be available for trade, nor is it logical to spend millions of dollars in maintenance, fuel, and ammunition costs. Modeling and simulation deliver a cost-effective solution for training a country's armed forces. One specific area of military simulation research is Semi-Automated Forces (SAF). SAF are forces which behave according to some artificial intelligence (AI) behavior model. These behavior models are commonly finite state machines. A SAF system simulates physical entities which follow these behaviors. SAF systems are vital tools for battle commanders, often allowing them to explore many different scenarios using digital maps and clickable units.

Earlier SAF systems commonly operated on a single supercomputer. Now, several SAF systems can be linked together to build one large SAF system over a network. For example, suppose the US Marine Corps was running a SAF simulation in California and the US Navy was running a SAF simulation in Maryland. Using the High Level Architecture (HLA) distributed simulation protocol [1], these two simulation systems could interact with each other via HLA's Run-time Infrastructure (RTI), which means the entities inside each simulation system could interact with each other even though they are not originally part of the same SAF system. See Figure 1.1.

This scenario may arise if the Marines were testing out new behaviors and needed an op-

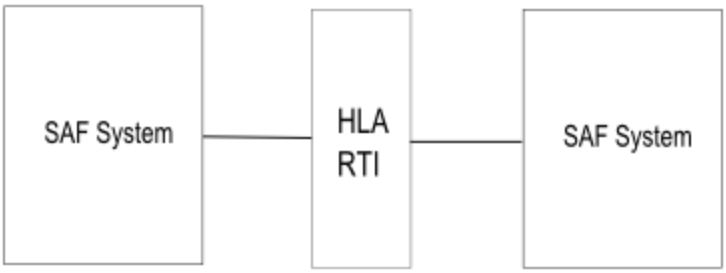

Figure 1.1: Two simulations systems connected via HLA

ponent not involved with the behavior model development. One approach the Marines could take is to build in all of the behavior models they plan to test before starting up their SAF system and connecting to the Navy's SAF system via HLA. However, what would happen if the Marines needed to adjust their behavior model? They would need to completely shut their system down, rebuild the executable, and start up the SAF system again. Then, they would need to reload the scenario. Meanwhile, the Navy SAF system and its users are left to themselves.

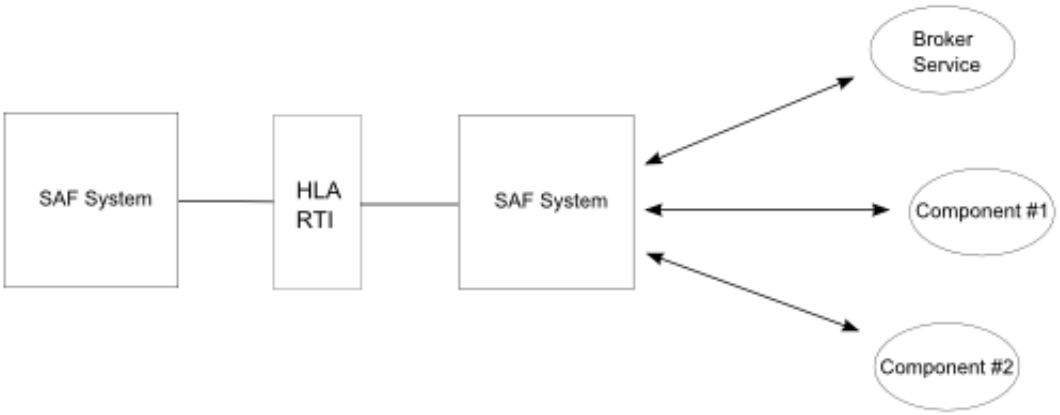

Figure 1.2: Web Services extending current system

We propose to use Web Services to dynamically link behavior models to a SAF system. Figure 1.2 gives an illustration of how we extend the current system. This approach would allow behavior models to be modified on-the-fly without any need to shutdown and restart the system. Furthermore, Web Services offers more flexibility for behavior development, by not restricting the behavior to the language of the SAF system. Finally, these behavior models can be deployed to several web servers. The extra deployment offers the advantage of balancing the load across several

servers, and also the possibility of choosing the faster server, similar to the concept of file-sharing mirrors.

1.2 Research Objectives

The overall intent of this research is to develop a methodology based on Web Services for integrating components, such as behavior models, with SAF systems. The components only interact with one SAF system, and therefore it is unnecessary to include them as a separate federate in the HLA design, which requires that the components interact through the RTI.

Included in this research are the following:

- Research and development of a design methodology for incorporating Web Services into a distributed simulation environment.
- Implementation of a Web Services framework.
- Integration of components with JSAF, to include:
  - two SEAD behavior models developed in MATLAB, and
  - a behavior recognition application developed in Java
- Implementation of a load-balancing mechanism to improve system performance.
- Analysis of system performance under CPU loads and network delays with and without load-balancing.

1.3 Overview

Chapter 2 covers important concepts and technologies related to distributed computing. An overview of several distributed computing paradigms is included, as well as discussions of several standards.

Chapter 3 provides an overview of distributed simulation, including a discussion of DIS and HLA. An introduction to the Joint Semi-Automated Forces (JSAF) simulator is also given.

Chapters 4 and 5 provide a discussion of the design and implementation of a framework combining JSAF with Java Web Services. This implementation allows JSAF to interface with

multiple applications, including behavior models. JSAF will be used to analyze these behavior models and provide feedback for improvement.

Chapter 6 discusses the behavior recognition system as a proof of concept. The chapter includes results from several experiments run with the behavior recognition system while it was attached to JSAF.

Chapter 7 discusses results from the system performance analysis based on simulated CPU loads and network delays.

Chapter 8 examines load-balancing and its tradeoffs.

Chapter 9 gives a summary of the work. It also provides a small discussion and suggests future work.

Chapter 2

Distributed Computing

Distributed computing is computing over several networked computers, each having its own processors and resources. Advantages of using distributed computing include resource sharing, scalability, fault tolerance, and affordability. By sharing resources, a larger number of resources are available for a particular process. Scalability is the ability to add or remove resources as necessary. Fault tolerance is available since the process does not necessarily rely on a single computer. Therefore, if one computer fails, the other computers can still function. Lastly, personal computers are less expensive than ever before, and the World Wide Web infrastructure is a good, stable, and affordable solution for connecting computers world-wide.

Several paradigms exist for distributed computing. Top among these are client-server, peer-to-peer, remote procedure call (RPC), and distributed object paradigms. The client-server paradigm is perhaps the most well-known and often used paradigm. Recently, peer-to-peer and RPC paradigms have become popular. Peer-to-peer networks rapidly emerged due to their ability to share media between several PCs without using a central repository. The RPC paradigm is the basis for Web Services. The distributed object paradigm is the natural extension of the RPC paradigm to an object-oriented architecture. Instead of calling a remote procedure, distributed object code invokes a method on a distributed object. These paradigms vary by their level of abstraction. For example, the client-server paradigm has a lower abstraction level compared to the distributed object paradigm. The trade-off to a higher level of abstraction is lower run-time efficiency, which is due to the inclusion of unnecessary features. Therefore, choosing the appropriate paradigm for a given task is important.

The following sections will explain the various distributed computing paradigms in more de-

tail. A survey of different standards will be provided for each paradigm. These standards will reflect past and current work in the field. The Common Object Request Broker Architecture (CORBA) and Web Services are most relevant to this work. CORBA is a distributed object standard, whereas Web Services is more closely associated with the RPC paradigm. Although these two standards arise from different paradigms, they can often be used interchangeably with minimal changes in the system architecture. Therefore, a section will discuss the pros and cons of these two standards. Finally, an overview of distributed computing in the military domain will be provided.

2.1 Client/Server Paradigm

The client/server paradigm is perhaps the most well-known paradigm for networked computing. The architecture is very simple. There are two processes, a client and a server. The server listens to and accepts requests, and the client issues requests and listens for responses. Many of the more complex paradigms follow from the client/server paradigm. Several standards are available for communicating between a client and a server. HTTP, FTP, and SMTP are well-known standards. Structured Query Language (SQL) also operates within the client/server paradigm. A user may send an SQL request to a database, and receive a response in the form of the requested data.

Hyper Text Transfer Protocol (HTTP)

The Hyper Text Transfer Protocol (HTTP) is a request/response standard between a client and a server. Its use for transferring hypertext documents led to the establishment of the World Wide Web. Development was coordinated by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF). The most used version is HTTP/1.1, which was published in June 1999 as Request For Comment (RFC) 2616.

HTTP defines message formats for a request and a response. A request message is formatted as follows:

- Request Line
- Headers
- Empty Line
- Option Message Body

HEAD	Asks for the response returned by the GET method, but without the message body. This is useful for obtaining meta-data without downloading the entire resource.
GET	Requests a representation of the specified resource.
POST	Submits data to the specified resource for processing. This may result in the creation of a new resource or updating of an existing resource.
PUT	Sends a representation of a resource.
DELETE	Deletes the specified resource.
TRACE	Echoes back the received request.
OPTIONS	Returns the HTTP methods that a server supports. Servers do not necessarily have to support all 8 methods.
CONNECT	Converts the request connection to a transparent TCP/IP tunnel.

Table 2.1: HTTP Request Methods

All but the HOST header is optional. The request line is formed using one of eight request methods. The request methods are available in Table 2.1. These request methods are sometimes thought of as "verbs" that act upon resources. For example, one might GET a webpage document or DELETE a music file. Web servers are required to support the GET and HEAD methods, but OPTIONS is heavily recommended as well.

HTTP is a stateless protocol, meaning the server does not need to retain information about users between requests. However, this forces developers to use alternative methods for storing a user's state. Several solutions exist including cookies, server-side sessions, hidden variables (used with forms), and URI encoded parameters. Also, there are 2 methods for secure HTTP. The first method is a URI scheme which signifies the browser to use SSL/TLS to protect traffic. The user simply uses https: instead of http: in the URI. The second method uses a header field called UPGRADE in the response message. If this message is received from a server, the client's browser then knows to use SSL/TLS.

Simple Mail Transfer Protocol (SMTP)

The Simple Mail Transfer Protocol is another protocol which defines several message formats. The messages for SMTP are all clear-text. They are used to send information to a mail server, which will route the mail to the proper recipients. As its name implies, SMTP is very simple. A user can emulate an e-mail client by simply opening up a telnet session to port 25 of a SMTP server and typing in clear-text commands. RFC 5321 defines the message formats for SMTP. There are several other RFCs which extend SMTP to provide authentication, enhanced status codes, secure

SMTP, and transmission of large and binary Multi-purpose Internet Mail Extensions (MIME).

Structured Query Language (SQL)

Structured Query Language (SQL) is a database programming language. It mainly supports the retrieval and management of data in relational database management systems. It also supports creation and modification of database schema, as well as database object access control management. SQL also includes a Call Level Interface (CLI) which accesses and manages data and databases remotely. A common criticism of SQL is its lack of standardization. While there are defined standards, they are not necessarily implemented the same across all vendors; this leads to ambiguous commands from one vendor to the next. With regard to the client/server paradigm, the database is analagous to the server. SQL in some respects defines message formats for accessing the database, similar to HTTP's message formats for accessing resources.

2.2 Peer-to-Peer Paradigm

The peer-to-peer paradigm is an extension to the client/server paradigm. Where the client/server paradigm has a dedicated server and client, the peer-to-peer paradigm consists of processes that act as both clients and servers. Now, instead of clients going through a server to interact with other clients, they can interact with each other directly. Peer-to-peer networks are ideal for instant-messaging, file-sharing, and video-conferencing. Napster.com is perhaps one of the best-known developers of a peer-to-peer network. Other projects include Juxtapose (JXTA) and Jabber. JXTA is a set of open-source peer-to-peer protocols produced by Sun Microsystems for connecting network components. Jabber is a technology which uses an XML-based, open-source protocol for instant-messaging called eXtensible Messaging and Presence Protocol (XMPP). Another peer-to-peer protocol is the Blocks Extensible Exchange Protocol (BEEP). These protocols are described in the following sections.

Juxtapose (JXTA)

Juxtapose (JXTA) defines a set of XML messages which allow any device connected to a network to communicate with any other device on the network, independent of the underlying network topology. JXTA implementations are available for Java, C/C++/C#, and J2ME. JXTA

defines two groups of peers: edge peers and super-peers. Super-peers are split into rendezvous peers and relay peers. Edge peers are generally on the outer edge of the Internet, such as home users or users behind a corporate firewall. These peers usually operate over a low bandwidth connection. A rendezvous peer coordinates interaction between several other peers. A relay peer allows users behind firewalls or Network Address Translation (NAT) systems to take part in any interactions. Resources are discovered in the network via advertisements. Advertisements are XML documents which describe resources in the network. Communication in JXTA may be thought of as an exchange of one or more advertisements between peers.

eXtensible Messaging and Presence Protocol (XMPP)

The original purpose of XMPP was for near-real-time instant messaging and presence information. However, due to its extensible nature, XMPP has also been applied to Voice over IP (VoIP) and file transfer signaling, among other things. In 2002, the Internet Engineering Task Force created a Working Group to formalize the core protocols. Four RFCs were approved as Proposed Standards in 2004, including RFC 3920 which defines the current XMPP protocol. These standards are still undergoing revisions, and it may be a while longer before they become true standards.

There are several advantages to using XMPP. First, anyone can run an XMPP server. In fact, there are thousands of servers running XMPP software. Second, XMPP has been used for approximately 10 years, a long time in the computing world. Third, XMPP is flexible, allowing custom functionality to be built on top of it. Finally, security has been built into the core XMPP specifications, and the XMPP Standards Foundation runs an intermediate certification authority at xmpp.net under the auspices of the StartCom Certification Authority.

There are also disadvantages to XMPP. Typically, 70% of XMPP traffic is presence data, with approximately 60% of this data being retransmitted. New research is being conducted to alleviate this overhead. Also, there is currently no encoding for binary data in XMPP messages. Instead, binary data is often sent through an external protocol such as HTTP.

Block Extensible Exchange Protocol (BEEP)

The Block Extensible Exchange Protocol is a framework for creating network application protocols. Unlike a client/server approach, it allows either side to send messages at any time. BEEP

is defined in RFC 3080. BEEP uses MIME encodings to transmit arbitrary file types which relieves the programmer from dealing with any specifics. BEEP also supports encryption, authentication, reporting of errors, and multiple asynchronous requests. While BEEP is not a true network protocol in itself, the ease it provides in creating a new custom protocol is very good for a developer.

2.3 Remote Procedure Call Paradigm

The Remote Procedure Call (RPC) paradigm also extends the client/server paradigm. Remote procedure calls are developed to look and feel like local procedure calls. The purpose of the RPC paradigm is to simplify development. By treating remote calls like local calls, the programmer does not have to think about where the function is executing. For example, suppose there are two processes, A and B. Process A may call a function of Process B, and pass B some parameters. Process B will execute the function, and return a value back to Process A. However, from a developer's point of view, it doesn't matter whether the function is executed locally or remotely, so long as the value returned is correct. There are two APIs commonly used for RPC, ONC RPC and DCE/RPC. Another RPC standard, XML-RPC, is the basis for Web Services. A brief introduction to the first two APIs is given, followed by a discussion of Web Services. As Web Services is the technology used in this work, more detail is provided.

ONC RPC

Open Network Computing Remote Procedure Call (ONC RPC) is a remote procedure call system. It was originally developed by Sun Microsystems as part of their Network File System project. It is defined under RFC 1831. ONC RPC is based on calling conventions used in UNIX and the C programming language. Data is serialized using the eXternal Data Representation (XDR). XDR is an IETF Standard, most recently described in RFC 4506. It wraps data in an architecture-independent format for use over a network. ONC RPC delivers the XDR paylaod via UDP or TCP. Access to a machine's RPC services are provided through a portmap service. This service usually runs on port 111 and listens for RPC requests. For each request, it determines which service is being accessed and the port on which the service is running. It then redirects the caller to the appropriate port. ONC RPC is a powerful system, but is generally not used for large-scale applications. DCE/RPC, CORBA and Web Services are generally used instead.

DCE/RPC

Distributed Computing Environment / Remote Procedure Call (DCE/RPC) is a subset of the Distributed Computing Environment (DCE). DCE is a distributed system that provides a framework for client/server interaction with an RPC mechanism, a naming service, a time service, an authentication service, and a distributed file system. While DCE was used much in the early 1990s, it is not used heavily today. However, Microsoft's DCOM, described later, uses DCE technology for its network transport layer. Although DCE/RPC could be used independently of the rest, it is more likely that the entire DCE system is used.

Web Services

Web Services is a set of standards adopted and maintained by the World Wide Web Consortium (W3C). According to W3C, a web service is "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

A Web Services framework is minimally implemented with two components, a service provider and a service requestor. An optional component is a service registry. The provider and requestor are analagous to a server and client. The service registry is used to locate services on a network.

The provider publishes a service interface document written in Web Service Description Language (WSDL). A client retrieves the service interface document and uses a parser to generate stub code, which makes available a set of function calls for accessing the web service. The stub code is usually in the form of source files.

A service registry is an intermediate component. If the client does not know which service it will use, it may query a service registry. The service registry will return a list of services from which the requestor can choose a specific service to consume.

Communication between the different components is done through messages. Typically, these are XML messages encapsulated in a Simple Object Access Protocol (SOAP) envelope and transmitted across HTTP. SOAP is a mature Web Service standard maintained by W3C. The most

up-to-date version, SOAP 1.2, was published as an official recommendation in June 2003. The reason for using HTTP is its wide-spread use and its ability to traverse firewalls through port 80. However, any transmission protocol may be used. If security is important, WS-I recommends HTTPS. Other protocols include Simple Mail Transfer Protocol (SMTP) and Blocks Extensible Exchange Protocol (BEEP). All of these protocols implement the client/server paradigm mentioned previously.

While Web Services has been heralded as a major revolution, there are several areas for concern. First, the reliability and security of Web Services are incompatible with current industry needs [4]. Web Services must guarantee Quality of Service (QoS) and high levels of security before they can be used for sensitive projects, such as those for the military [7].

From an efficiency point of view, SOAP is not the best solution for modeling and simulation [10]. For the moment, it is an acceptable solution because the high level of interoperability outweighs the need for an efficient transport mechanism. Several binary SOAP formats have been proposed including Fast Infoset [2], EXI [31], and CBXML [17]. Also, many SOAP implementations limit the packet size. Solving this problem simply requires developing a SOAP implementation that allows for larger packet sizes. Likewise, HTTP is a good solution, but is not always the best solution. Simple Mail Transfer Protocol (SMTP) and Blocks Extensible Exchange Protocol (BEEP), mentioned earlier, provide better support for large packet sizes. Pingali and Stodghill [20] implement a new SOAP protocol which addresses many of these concerns.

Finally, Web Services are inherently stateless. They are meant to perform an action on some data, then return a result. The service is not expected to save data between separate sessions, even if it is the same client. One solution proposed is a WS-Resource standard [11]. Another solution in wide use today is cookies. Cookies store state information on the client. The state data is then passed back and forth as necessary. Although this requires more network usage, it saves memory on the server, which is very important when serving several clients.

2.4 Distributed Object Paradigm

The distributed object paradigm is an object-oriented extension of the previous paradigms. The calling process executes a method on a distributed object. For instance, suppose a Car object existed on the network. The calling process might invoke the getMileage() method, which would return the mileage of the Car object. Similarly, the calling process may change the mileage. Even

though the object is not local to the process, the process may still modify it.

There are two possible methods for accessing objects distributed over the network. The first method requires that the calling process specify a distributed object. The calling process then invokes methods on that object. Java's Remote Method Invocation (RMI) uses this method. The second method requires the use of an Object Request Broker (ORB). The ORB finds a suitable object that has the required services that the calling process needs. For example, suppose the calling process wanted to know the stock value of Microsoft. The process may not know of a specific object that can determine the stock value. However, the process can query an ORB, which will locate a suitable object. The ORB will return a reference to the calling process. The process may then access the object as if it is a local entity. The Common Object Request Broker Architecture (CORBA) is the leading standard for the Object Request Broker model. DCOM, Microsoft's standard for the distributed object paradigm, is a strong competitor. A brief overview of Java's RMI, CORBA, DCOM, and .NET are provided below.

Java Remote Method Invocation

Java RMI is an API for the distributed object paradigm. There are two common implementations of the Java RMI. The first implementation is between two Java Virtual Machines (JVMs). Client stub code and server skeleton code is developed to communicate between each other. The client and server interact through the stub/skeleton code. This implementation requires that the client and server be implemented in Java unless they are implemented through Java's native interface (JNI) which provides a method for other languages to run in the JVM. The second implementation interacts with CORBA. Instead of the stub/skeleton code communicating directly, they interact with an ORB. This implementation allows the client/server to be implemented in many other languages, which offers greater flexibility. However, there is a much higher level of complexity.

Common Object Request Broker Architecture

The Common Object Request Broker Architecture (CORBA) is a distributed object architecture which attempts to maximize interoperability between different platforms and programming languages. CORBA was designed by the Object Management Group (OMG) in the early 1990s. It is a set of protocols which describe the implementation of a distributed objects environment, and it

offers a good solution for interoperability between platforms and languages. CORBA has been used for many business and scientific applications, but interest has waned in favor of Web Services.

CORBA connects a client with a server. Logically, the client invokes a method on a distributed object located on the server. The client interfaces with a proxy, called a "stub". The server interfaces with a proxy of its own, called a "skeleton". The client and server proxies interface with an Object Request Broker (ORB) which communicates with the network and operating system of its respective host. By using a common protocol, the ORBs on either side can effectively resolve differences between programming languages, as well as between different platforms. CORBA also provides a naming service to assist the client in locating distributed objects, which operates similar to a Web Services registry.

The "stub" and "skeleton" are created from an interface file. This file is written in CORBA's Interface Definition Language (IDL) with a syntax similar to C/C++ and Java. For several programming languages, including C/C++ and Java, OMG has created a standardized mapping from IDL to the language. This allows a compiler to be used to generate the proxy code for a particular language.

ORBs, although written in various languages, can interoperate due to their common protocol. General Inter-ORB Protocol (GIOP) is a specification that provides a general framework for interoperable protocols to be built on top of a specific transport layer. One special case of the GIOP is the Internet Inter-ORB Protocol (IIOP), which is GIOP applied to the TCP/IP transport layer.

The naming service provided by CORBA is an optional component used to locate distributed objects. A client queries a naming service to find a distributed object fitting a specific description. The naming service supplies the client with a set of objects matching the description. The client then notifies the naming service of the object it wishes to access. Finally, the naming service sends the client a specific endpoint used to reference the object. From this moment onward, the client and the server hosting the distributed object communicate independently of the naming service.

Distributed Component Object Model (DCOM)

DCOM is a Microsoft solution for the distributed object paradigm. It is an extension of COM to network applications, offering two new contributions to COM. First, DCOM contains a solution for marshalling data. Second, garbage collection over the network is addressed. DCOM was a strong competitor of CORBA. However, DCOM had some drawbacks. First, it worked well

for Microsoft systems, but was not completely supported by other systems. Second, the transport protocol was a binary format. While this format is quicker, it requires an interpreter,which would need to be developed for every architecture to marshal and unmarshal data. DCOM is still in wide-use today, but is listed as deprecated in favor of Microsoft's .NET framework.

.NET Framework

Microsoft's .NET provides a virtual machine for running applications, similar to Java's virtual machine. Therefore, .NET's methods for distributed computing are fairly similar to that of Java's RMI. The use of one over the other would then depend only on what platform one is using. If the process will run under a Microsoft operating system, .NET would certainly be the correct software to use. However, if the process is meant to run under other operating systems, Java's RMI might be the better solution since Java clearly supports more platforms and operating systems.

2.5 Web Services and CORBA

There is debate as to whether Web Services or CORBA is the better solution. Many researchers have been wary of Web Services and believe it is a reinvention of CORBA. Other researchers believe Web Services is useful and has its place. For a while, Web Services was still inferior to CORBA, but with several advances in the past few years, Web Services has definitely made its place in distributed computing.

Web Services and CORBA both provide solutions for interoperability between components. The biggest difference between the two is CORBA operates on distributed objects while Web Services is more of an RPC implementation. This means that Web Services is inherently state-less. Stateless is fine as long as multiple instances of the web service are not needed. Otherwise, it will be necessary to modify the web service to make it stateful, which will increase the code complexity. Another difference is the messaging protocol. Web Services commonly uses SOAP over HTTP, whereas CORBA uses its own protocol. SOAP is known to be very inefficient in terms of bandwidth, memory, and CPU usage. However, SOAP also enables a high level of interoperability. As such, although CORBA will almost always be faster than Web Services, Web Services can usually provide a higher level of interoperability. A benefit to using HTTP port 80 is that most firewalls enable network traffic by default. However, with CORBA, firewalls may be an issue.

Whereas CORBA is stateful and runs efficiently, Web Services provides a high level of interoperability and uses the existing Internet infrastructure to its advantage. Both technologies have their place and should be used for their appropriate domains. For example, Web Services should be avoided when state is important and efficiency is a must. However, for stateless services that require a high level of interoperability between components, Web Services is generally better.

2.6 Distributed Computing in the Military Domain

Web Services are considered a part of a larger Web movement called Web 2.0 [19]. [26] suggests that the prominent ideas of Web 2.0 can be used for other military tasks, not just simulation. For example, wikis could allow planners to collaborate and work on a single document together. Services like Twitter could be used to view status updates for a computer network, and online worlds similar to Ragnarok Online and Warhammer Online can be used for meetings using Voice over IP (VoIP).

Also, the military has started to hold a special interest in the gaming industry [25]. In particular, the gaming industry has a much higher demand and, thus, much larger operating budgets. Many games simulate military operations and training, such as the Civilization series, Rise of Nations, HALO, and Call of Duty, to name a few. A game currently used by 9,442,212 registered players is America's Army [3], which is role-playing game used by the US Army to for recruiting purposes. These games use powerful gaming engines which can operate on the user's local computer. Other games interact using a distributed computing paradigm, such as World of Warcraft, Everquest, and EVE Online. These games provide a virtual world accessed by millions of users world-wide. Much of the computation is done locally, while powerful servers mediate between several clients. The idea of creating a virtual training world for soldiers is an exciting prospect for the future. A soldier could travel through portals to different training areas where he or she can practice individual skills, team skills and command skills. Other portals may lead to foreign lanugage training where soldiers can speak with foreign AI characters or live language instructors. For example, the US government uses games developed by Tactical Language Training, LLC [27] which provides language practice for soldiers in real situations.

Some military personnel are hesitant to use gaming technology for serious problems such as war. Others realize that the gaming industry has developed very strong protocols for handling a large

number of clients in a virtual world. The data is passed quickly and efficiently, which is necessary for the game to be successful. In effect, the military would be getting the benefit of gaming's market forces while only spending the money necessary to use technology that has already been developed. One such project that uses commercial gaming software is UTSAF [21], which bridges the Unreal gaming engine with the OneSAF Test Bed (OTB). OneSAF's GUI only displays a 2D view of the battlespace. By integrating the Unreal gaming engine with OTB, the user is provided with a 3D view of the battlespace, which could provide more realistic views of the simulation.

In fact, the use of Commercial Off-The-Shelf (COTS) software is becoming increasingly common in the military [24]. With constrained budgets and the need to supply forces with armor, equipment, medical services, and much more, it is paramount that the military find outside sources for their technology. However, verifying this technology is acceptable for military purposes is a heavy requirement. The military requires high levels of security and reliability that currently do not exist in the many new standards.

Chapter 3

Distributed Simulation

Distributed simulation applies principles and models from distributed computing to simulation. Distributed simulation has been a dominant area of research within the military for its affordability and effectiveness. In the late 1980s, the Defense Advanced Research Project Agency (DARPA) developed SIMNET, which connected several manned tank simulators. Distributed Interactive Simulation (DIS), not to be confused with the field of distributed interactive simulation, was developed in the early 1990s, building on the success of SIMNET. Development of the High Level Architecture (HLA) was begun in the mid 90s. HLA became an IEEE standard in 2000, and is widely-used today. With regard to the military, HLA is primarily used within SAF simulations. The DIS, ALSP, and HLA standards will be explained in further detail over the next few sections. A final section will focus on SAF systems and provide some information on two popular SAF systems, JSAF and OneSAF.

3.1 Standards

3.1.1 Distributed Interactive Simulation

Distributed Interactive Simulation (DIS) is an open standard for distributed real-time simulation over multiple host computers. It is defined under IEEE Standard 1278. Although primarily used by military organizations, it has also been applied to space exploration, medicine, and business. DIS borrows on SIMNET's concept of dead reckoning. Dead reckoning is the process of estimating

Entity Information / Interaction	Entity State, Collision, Collision-Elastic, Entity State Update
Warfare	Fire, Detonation
Logistics	Service Request, Resupply Offer, Resupply Received, Resupply Cancel, Repair Complete, Repair Response
Simulation Management	Start/Resume, Stop/Freeze, Acknowledge, Action Request, Action Response, Data Query, Set Data, Data, Event Report, Comment, Create Entity, Remove Entity
Distributed Emission Regeneration	Electromagnetic Emission, Designator, UA, IFF/ATC/NAVAIDS, SEES
Radio Communications	Transmitter, Signal, Receiver, Intercom Signal, Intercom Control
Entity Management	Aggregate State, IsGroupOf, Transfer Control Request, IsPartOf
Minefield	Minefield State, Minefield Query, Minefield Data, Minefield Response NACK
Synthetic Environment	Environmental Process, Gridded Data, Point Object State, Linear Object State, Areal Object State
Simulation Management with Reliability	Start/Resume-R, Stop/Freeze-R, Acknowledge-R, Action Request-R, Action Response-R, Data Query-R, Set Data-R, Event Report-R, Comment-R, Create Entity-R, Remove Entity-R, Record Query-R, Set Record-R, Record-R
Live Entity	TSPI, Appearance, Articulated Parts, LE Fire, LE Detonation

Table 3.1: DIS Protocol Data Units

remote entity attributes, such as position and orientation, based on previous values. Through dead reckoning, it is possible to efficiently transmit the state of battle field entities. As such, general purpose computers may be used in place of supercomputers, allowing hundreds of online players to participate.

DIS is based on a message-passing paradigm. Specifically, simulation state information is encoded in formatted messages, known as Protocol Data Units (PDUs). These messages are transmitted across the network using existing transport layer protocols. Broadcast User Datagram Protocol (UDP) is most commonly used. DIS has defined a set of 66 PDUs, separated into 11 families. A list of these families and their PDUs is displayed in Table 3.1.

3.1.2 Aggregate Level Simulation Protocol

Aggregate Level Simulation Protocol (ALSP) is a protocol and supporting software that enables interoperability between simulations. In 1990, DARPA employed the MITRE Corporation to study the application of distributed interactive simulation principles used in SIMNET to aggregate-level constructive training. Aggregate-level training is training at the staff level. Therefore, the entities are commonly modeled using Lanchester's Laws [15], which apply to large forces. This is

in contrast to entities modeled as physical weapons and individual soldiers. By 1995, the project had transitioned from DARPA and MITRE Corp. to the US Army's Program Executive Office for Simulation, Training and Instrumentation (PEO STRI). Further, ALSP had successfully linked seven existing simulations from several military services. These seven simulations are:

- US Army - Corps Battle Simulation (CBS)
- US Air Force - Air Warfare Simulation (ASWIM)
- US Navy - Research, Evaluation, and Systems Analysis (RESA)
- US Marine Corps - Marine Air Ground Task Force (MAGTF) Tactical Warfare Simulation (MTWS)
- Electronic Warfare - Joint Electronic Combat Electronic Warfare Simulation (JECEWSI)
- Logistics - Combat Service Support Training Simulation System (CSSTSS)
- Intelligence - Tactical Simulation (TACSIM)

ALSP borrowed several things from SIMNET. These include dynamic configurability, geographic distribution, autonomous entities, and an implementation of a message-passing paradigm. Dynamic configurability is the ability to add and remove simulations during runtime. Geographic distribution is hosting simulations without regard to location. Autonomous entities are responsible for keeping track of their own resources, firing their own weapons, and performing damage assessment locally. Finally, message-passing is used to route messages among the simulations.

ALSP also had new design objectives not available in SIMNET. ALSP was to provide simulation time management, data management, and be able to operate independent of the system architectures that hosted the simulations. The time management must manage time across all of the simulations, keeping them in sync with each other. The data management required a common representation that could be used by all simulations.

ALSP consists of three things: (1) ALSP Infrastructure Software (AIS), (2) a reusable ALSP interface, and (3) simulations adapted for use with ALSP. The AIS provides distributed simulation support and management at runtime. It consists of the ALSP Common Module (ACM) and the ALSP Broadcast Emulator (ABE). The ACM provides a common interface for all simulations. Each simulation runs an instance of the ACM. The ACM handles incoming messages, and administers

the attribute database and message filter information. In many ways, the ACM is similar to client stub code in distributed computing paradigms. ABE facilitates broadcasting messages. It routes messages to the other simulations, similar to a network router.

The reusable ALSP interface is a set of generic data exchange message protocols. There are 4 types of messages: update, interaction, refresh request, and deletion. Update messages create or modify an object; Interaction messages indicate that one object is interacting with another; Refresh request messages request an update of attribute values; and deletion messages remove an object.

ALSP was an improvement from SIMNET, but still was not enough. High Level Architecture (HLA), described in the next section, was started to replace both DIS and ALSP. While, HLA did replace ALSP, DIS is still used today in many systems. However, it is usually coupled with HLA.

3.1.3 High Level Architecture

High Level Architecture (HLA) defines a standard framework for supporting simulations composed of different components. These components may be simulation models, human user interfaces, monitors, data analysis tools, and others. With HLA, several different simulation components work together to form one large simulation. HLA uses the term "federate" to mean a simulation component, and the term "federation" to mean a group of these components working together.

Structure

The HLA standard consists of three parts. These parts are Federation/Federate Rules, an Interface Specification, and an Object Model Template. The federation/federate rules govern the behavior of a federation and its federates. The interface specification describes how federates will interact with the federation. Finally, the object model template provides a common framework for object model documentation across all simulation components.

There are 10 rules that govern the behavior of a federation and its federates. These rules must be obeyed for a federation or federate to be considered HLA-compliant. These rules are split into two groups of five rules each; the first group defines rules for a federation, while the second group defines rules for a federate. The rules are:

1. Federations shall have a Federation Object Model (FOM), documented in accordance with the OMT.

2. All representation of objects in the FOM shall be in the federates, not in the Run-Time Infrastructure (RTI).

3. During a federation execution, all exchange of FOM data among federates shall occur via the RTI.

4. During a federation execution, federates shall interact with the RTI in accordance with the HLA interface specification.

5. During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.

6. Federates shall have a Simulation Object Model (SOM), documented in accordancec with the OMT.

7. Federates shall be able to update and/or reflect any attributes of objects in their SOM, and send and/or receive SOM interactions externally, as specified in their SOM.

8. Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOM.

9. Federates shall be able to vary the conditions under which they provide updates of attributes of objects, as specified in their SOM.

10. Federates shall be able to manage local time in a way which will allow them to coordinate data exchange with other members of a federation.

The first five rules establish a common ground for federations, including documentation requirements (Rule 1), object representation (Rule 2), data interchange (Rule 3), interfacing requirements (Rule 4), and attribute ownership (Rule 5). The second five rules are for individual federates and include documentation requirements (Rule 6), control and transfer of relevant object attributes (Rules 7, 8, and 9), and time management (Rule 10).

The interface specification describes how federates will interact with the federation and, ultimately, with each other. The interface specification defines a standard for a Run-Time Infrastructure (RTI). Several RTIs have been developed. A list of available RTIs may be found in [33].

The RTI provides several management services. For example, it separates communication from actual simulation, facilitates construction and destruction of federations, supports object declaration and management between federates, provides efficient communication to logical groups of federates, and provides assistance with time management. The services are split into several groups, called management areas. The groups are:

- Federation Management
- Declaration Management
- Object Management
- Ownership Management
- Data Distribution Management
- Time Management

The Object Model Template (OMT) provides a common framework for object model documentation across all simulation components, which promotes reuse and interoperability. The OMT requires an object class structure table, object interaction table, attribute/parameter table, and FOM/SOM lexicon. Optional information includes a component structure table, associations table, and object model metadata. By properly documenting all object models, researchers can more easily integrate components using HLA. Furthermore, if the documentation were machine-processable, much of the work could be automated.

Current work in HLA

HLA continues to be the technology of choice for connecting separate simulations. One current push in research is to "web-enable" HLA. eXtensible Modeling and Simulation Framework (XMSF) [6] aims to integrate HLA with a Web Services framework. The over-arching concept is to access the Run-Time Infrastructure (RTI) via Web Services allowing the federation to be constructed quickly from several geographically-independent federates. The approach will also introduce a higher level of interoperability between federates and the RTI. SISO is currently working to modify the current HLA standard, possibly as a result of the power of XMSF. Currently, the RTI interacts with the federates through ambassadors, very similar to how Web Services operate using stub code.

The new "HLA-Evolved" RTI will have a Web Services API written in the Web Service Description Language (WSDL).

Some researchers [7] proposed the idea of federates as web services. This work would parallel that of HLA. A similar idea is to develop models as web services, proposed by Zhang et al [36]. These models would then interact through federates in a HLA federation. This approach is very similar to our work. Our work differs from Zhang's work in that we have developed a load-balancing mechanism to explore the possibility of running multiple identical models on the same network. Also, the models referred to in Zhang's paper are physical models running on another machine. Our work is geared towards behavior models. The physical model will exist in the SAF system and behave based on the response from the behavior model web service. There are several benefits to these approaches. For one, models are developed by several different researchers in hundreds of locations world-wide. These models are written in several languages on several different platforms. By deploying the model as a Web Service, a simulation can access the model's latest version (or earlier versions) using a standard interface, regardless of its underlying structure. The model can be developed and deployed by a single research group without the necessity of porting the code to any single platform or computer language. It also provides a method for dynamically loading models as needed. Web Service registries allow for discovery of these models. However, if registries are to be used, it is of extreme importance that a formal description of these models be available.

In general, formalization is necessary in several other areas of modeling and simulation. Several formalization efforts are underway, such as SISO's Base Object Model (BOM) [12] and the Federate Architecture Metamodel (FAMM) [28]. A BOM will give a precise definition to each piece of a simulation to enable rapid composability of a networked simulation. FAMM formalizes the HLA interface specification. This formalization is machine-processable and will allow the use of code-generation tools.

Bridging multiple federations is another ongoing area of research. Federates are commonly developed to interact with a specific RTI implementation. Several implementations exist, some commercial and others not [33]. These implementations often implement additional features not found within the IEEE 1516 standard. Therefore, not all federations are interoperable with each other. One approach is to develop a federate whose sole purpose is to bridge two seperate federations [9]. The federate would join both federations and route information between the two, mediating between data as necessary. Several issues arise with this approach and it has been shown that deadlocks can

occur unless modifications are made to the HLA standard for RTIs. Some researchers extended HLA using CORBA [8]. This allowed for inter-RTI communication, but further complicated the system. The researchers noted that it is only a short-term solution until a proper inter-RTI standard (similar to the IIOP standard in CORBA) is introduced. The use of CORBA also provides an easier method for linking federates not written in a language supported by the HLA standard. At the moment, HLA only supports Java, C/C++ and Ada95. A similar project was done using Web Services [32].

3.2 Semi-Automated Forces

Semi-Automated Forces (SAF), or Computer-Generated Forces (CGF), are forces which behave according to some AI behavior model. SAF are a unique solution for simulating opposition forces (OPFOR), as well as friendly forces which do not necessarily need a human operator. SAF offer a significant cost reduction in terms of time and money by emulating basic human behavior. More advanced behaviors are also applied to SAF, gathering scarce human expertise and making it readily available. A good overview of SAF is found in [22]. Modular SAF (ModSAF) was the first major SAF system to be used. Several newer SAF systems were built from ModSAF's code base, including OneSAF [18] and Joint SAF (JSAF) [29]. ModSAF retired in January of 2001 [30]. However, OneSAF and JSAF are still being developed and used by the US Armed Forces. An overview of JSAF and OneSAF is provided in the next two sections.

3.2.1 Joint SAF and OneSAF

Joint Semi-Automated Forces (JSAF) is an entity-level military simulation system used by many U.S. military groups. Simulations employing up to 100,000 entities are routinely run by USJFCOM (United States Joint Forces Command). Also, the U.S. Navy's Maritime Battle Center uses JSAF as a primary simulation tool for fleet exercises as well as a part of the Joint National Training Center. See Figure 3.1 for a screenshot of JSAF.

JSAF is sponsored by the Joint Concept Development and Experimentation Directorate (J9) of USJFCOM. It evolved from DARPA's Synthetic Theater of War (STOW) Advanced Concept Technology Demonstration (ACTD), whose aim was to demonstrate and evaluate distributed simulation standards to support mission rehearsals and joint command and staff training.

JSAF is an open-architecture, "open-source" endeavor. More precisely, JSAF is "government

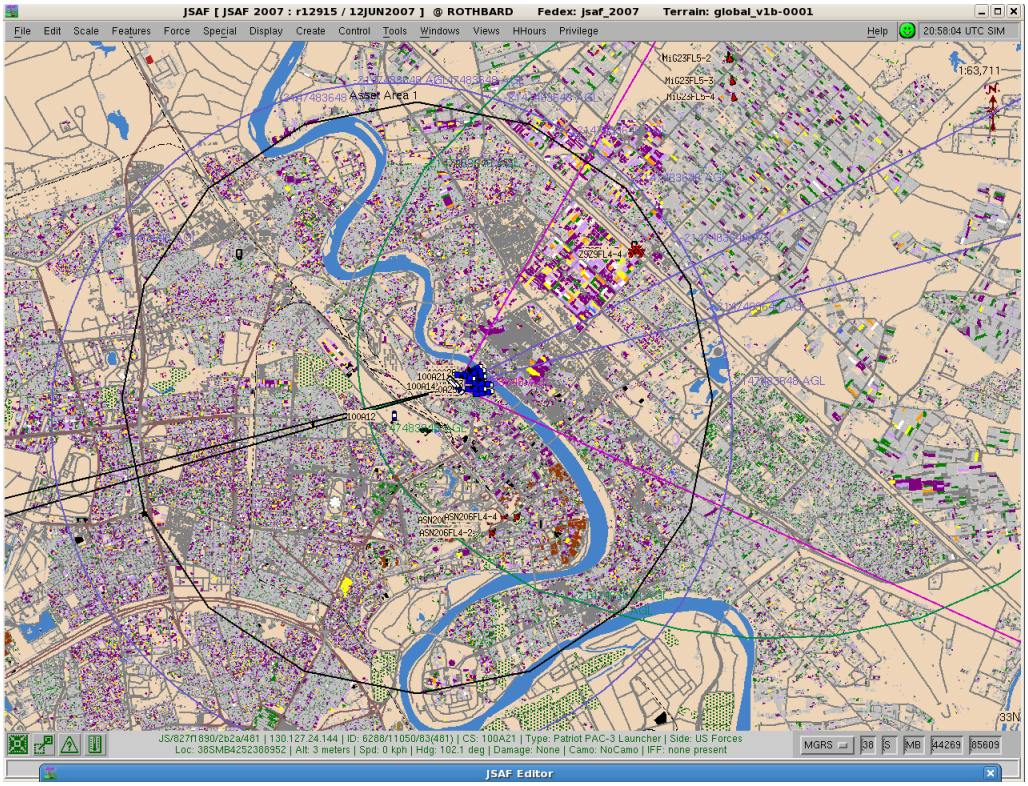

Figure 3.1: JSAF Screenshot

source available". The open-architecture "allows easy internal replacement of both high and low fidelity models" [13]. JSAF simulations can be run locally or distributed on a network. Multiple HLA federations are supported by JSAF.

Entities include infantry, tanks, ships, aircraft, munitions, sensors, and buildings. They can be controlled individually or commanded in a unit, such as a company or battalion. Entity behaviors are affected by line of sight, time of day, currents, tides, slope, smoke, soil conditions, water depth, and cloud cover. Civilian behavior is also simulated in JSAF, critical when representing urban environments. Missions can be developed to explore possible scenarios. JSAF's synthetic environment represents real-world terrains, oceans, and weather conditions, which all affect the execution of the simulation. In particular, the real-world terrains are drawn from a large-scale database, which includes both rural and urban areas.

OneSAF is almost identical to JSAF. The only real difference is the development team. Whereas JSAF is used more by the US Navy, OneSAF is used primarily by the US Army. OneSAF's

mission is to create one SAF system for everyone. A single SAF system would mean developing one model and sharing it with several applications. The model may be a new missile, aircraft, or civilian behavior. OneSAF appears to be the better SAF system, but only a thorough comparison of the two systems could determine if this statement is true. However, it is expected that the US Army's expert knowledge would be superior to most other military services in the area of urban combat, which is where war seems to be heading.

Chapter 4

Design and Methodology

The problem presented in this thesis is how to connect software components to an HLA-compliant simulator, specifically a SAF simulator such as JSAF or OneSAF. These software components may be analysis tools, monitoring programs, or behavior models. The fundamental difficulty associated with this work is developing a solution independent of any one specific simulator. Similarly, the software components should not be restricted to a single language or platform. Finally, the solution should support load balancing between several identical models. For instance, suppose several SAF systems wanted to use the same model. There is a potential bottleneck in the system if all of the SAF systems accessed the same model. However, if multiple, identical models were available, an intelligent solution could assign each SAF system to a specific model.

4.1 Design

The components will be connected over a network. Therefore, it is logical to let the network be the interface between the components. The network protocol should be understood by both sides. Furthermore, the data passed between sides should be understood and language-independent. To be language independent, the data must be interpreted to and from a network format. This format should be abstract enough to encompass all languages. Ideally, only primitive types common to all languages would be available, such as integers and floats. However, to make the programming logic easier, higher-level data structures should be available. These data structures would need to be well-described, possibly using XML or some other description language. HLA, CORBA, and Web

Services are all technologies which use the network as the primary interface. These technologies were discussed in Chapters 2 and 3. They are briefly mentioned here with advantages and disadvantages as a solution to the problem. Afterwards, the reasons for using Web Services are explained, then the system design is explained.

High Level Architecture

The first possible solution is HLA. Components can be developed to interface via the RTI. This solution is good because it guarantees interoperability between the components and the simulator. It also uses the existing standard. However, there are three good reasons for not using this approach. First, a large amount of development time is required to transform a legacy component into an HLA-compliant component. In the worst-case scenario, the component may need to be completely rewritten. Second, in our design, the component is only required to interact with a single HLA federate, independent of any other federates. Therefore, it is not necessary to involve the RTI. The RTI will only add overhead and delay to the system. Third, the RTI lacks support for load-balancing, which is a key component of the final solution.

CORBA

CORBA is a distributed computing technology that could also solve the problem. CORBA handles objects and object references, making it very scalable. For example, multiple vehicles may execute the same behavior model by requesting a new instance of the behavior model. However, CORBA's API is difficult to use and burdened with unnecessary complexity. The cost to develop and maintain a CORBA implementation is not viable. Furthermore, CORBA's ORBs are propietary, so this solution would require choosing a specific ORB.

Web Services

A third solution to the problem is Web Services. While Web Services is not the perfect solution, it is a simple solution. The interface between component and simulator would be SOAP, a standard recommended by W3C. SOAP is much simpler than HLA's RTI and CORBA's ORBs. Furthermore, rapid development is possible due to the simplicity of Web Services and the large amount of effort put into IDEs. One pitfall with Web Services is the fact that it is a "stateless"

technology, meaning there is only one instance of an object on the server. Therefore, the creation and deletion of objects is up to the programmer. In a positive light, this provides the programmer with more control over the development of the web service. This will be good when the web service for the SEAD behavior model is developed in the next chapter, since it would be detrimental to performance to create a new MATLAB runtime instance for every client. Finally, Web Services offers the possibility of load-balancing. If each model is run under several web servers, another web service can determine which web server is currently performing the best, and assign a SAF system to that particular web server.

4.2 Methodology

Figure 4.1 shows the top level of the system design. There are two major components of the design: the simulator and the web services. The components are developed independently. The only connection between the two components is the bridge provided by SOAP over HTTP. Independent development offers many advantages with the main advantage being that the projects can be developed by geographically-separated groups with no knowledge of one another. Also, a rebuild of one component does not require a rebuild of the other component. Finally, in the case of Web Services, the web service may be redeployed while the simulator is executing. The simulator will only notice a short loss of connectivity. In the case of a behavior model, the model can be modified on-the-fly and the new behavior will be immediately available in the SAF system. All of these advantages are nullified if the service interface is modified. Therefore, the interface should be decided upon as soon as possible, and changes should be made very seldom.

One web service, known as a Quality-of-Service (QoS) Broker, is designed as a load balancing agent. The Qos Broker web service is similar to a service registry, but chooses a service based on a Quality-of-Service measure. This web service stores information for each web server that hosts services available to the simulator. Some information stored includes the number of users, the number of running processes, the percentage of free swap memory, and the ping delay between the web server and simulator. The web service receives requests from the simulator and returns a server IP address. The server is chosen based on a metric computed from the server information. For instance, a server with a small number of users and running processes is preferred to a server with several users logged in, each running multiple processes. The QoS Broker maintains a list of

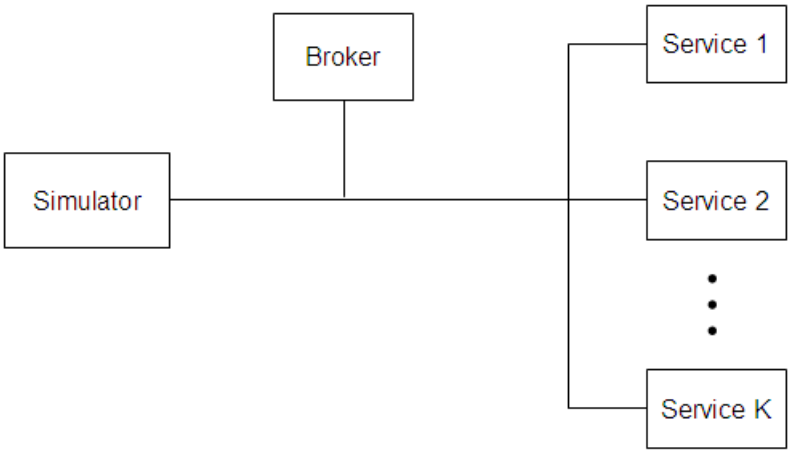

Figure 4.1: System Design

available servers, as does the simulator. If the QoS Broker is unavailable, the simulator can use the last known good server or randomly select a server from its internal list. In a worst-case scenario, the simulator will use a web server local to the machine.

The QoS Broker service will have a set API through which clients may access it. To understand this API, it is necessary to explain the process through which a client interacts with the QoS Broker. Initially, a client registers with the QoS Broker. Upon registering, the client recieves a unique ID. Whenever the client needs to reaccess the QoS Broker, it identifies itself through this ID. Should a client discontinue using the QoS Broker, it must unregister itself. By unregistering, the QoS Broker will know that the client is no longer accessing its assigned server. Once registered, a client can query the QoS Broker for the optimal server. The QoS Broker will then assign the client to a server. These three methods (register, unregister, and connect) are the primary base for communicating with the QoS Broker. A fourth method will also be available to monitor the QoS Broker's internal values. For instance, one might like to monitor server activity over time. It would also be helpful for tuning the weights applied to the QoS metric. See Figure 4.2 for a Java interface of the QoS Broker.

The rest of the web services are developed as software components to be attached to the simulator. In our work, we developed two SEAD behavior models and a behavior recognition service. The SEAD behavior models are functions which accept the current state and return a new state.

```
interface QoSBroker {

    int register ();

    void unregister (int id);

    String connect (int id);

    QoSInfo monitor ();

}
```

Figure 4.2: QoS Broker API

The functions are deployed as web services, which allow them to execute on a remote machine, thereby offloading computation. The behavior recognition service acts as a central storage device for the behavior recognition system and the simulator. The behavior recognition system will be described in more detail in Chapter 6.

```
interface BehaviorModel {

    State simulate_cycle (State oldState);

    State simulate_n_cycles (State oldState, int n);

}
```

Figure 4.3: Behavior Model API

Whereas the QoS Broker has a set API, the other services are not required to conform to a set API. Instead, the API is determined by the service developer. The purpose of the WSDL file is to document this API in a machine-processable language. For instance, a behavior model will have a different API than a monitor. However, since behavior models will be a large portion of the services, a recommended API will be provided. Figure 4.3 gives this recommended API. For most behavior models, only one method is necessary. This method implements a single step in time. It receives the system state as input, then outputs the new state. The developer may also include a second method which determines a new state after multiple time steps. In this case, the number of time steps will also be input into the method. Any other methods should not be publicly accessible. Fewer methods to the client provides more control to the service, which will likely result in fewer errors.

Chapter 5

Implementation

The system design was implemented over several stages. First, the web services were developed within the Netbeans IDE. Second, the web service stub code was generated using gSOAP, and was inserted into the simulator source code. The stub code provides functions for invoking the web services. Third, behavior models within JSAF were modified. The behavior models accessed the web services for functionality, rather than executing local procedure calls. Finally, the behavior models within JSAF were further modified to use a load-balancing web service. The load-balancing web service attempts to distribute the load across several servers, which should increase the system's quality of service.

5.1 Web Services Development

Web service development was done within the Netbeans IDE [Netbeans 6.1]. Netbeans is an open-source IDE sponsored by Sun Microsystems. Although primarily built for Java, Netbeans supports several other programming languages. The Web and Java EE version of Netbeans provides built-in support for constructing web services. For example, Netbeans provides automatic code generation for client-side applications. Also, functionality exists to automatically create an interface file from a Java class, commonly known as a WSDL file. These extensions are invaluable when quickly developing a system prototype. However, it is almost always recommended that programmers write their own WSDL file in the later stages of development, especially for complex systems. Self-written WSDL files are generally cleaner and more accurate representations of the interface.

When preparing to integrate a component with the SAF simulator, three approaches may be taken. The approach taken depends on the programming language of the component. For a Java component, the code may be wrapped as a Java web service and deployed on a Java application server, such as Glassfish, JBoss, WebSphere, or Oracle. If the component was written in C# or Visual Basic, then it would make sense to use Microsoft's .NET framework. The third approach is necessary if a component is written in a language that does not lend itself to any available application servers. A component written in MATLAB is one example. This approach is much more complex when compared to the first two approaches. However, it is also the approach taken in this work since the SEAD behavior was written in MATLAB. Therefore, the next section will discuss this approach by going through the implementation of the SEAD behavior models as web services.

5.1.1 SEAD Behavior Model

The SEAD behaviors used in this work are implemented as MATLAB functions. Here, an example web service which provides a method for calling MATLAB functions is presented. The web service will be implemented in Java, so the following code will be provided as Java code.

MATLAB is similar to Java in that it runs under a virtual machine. As Java requires a virtual machine to be running for code execution, an instance of the MATLAB runtime environment must be started to execute the SEAD functions. Then, the input, output, and error streams of the runtime environment are accessed. The Java code to start a MATLAB process and access its input, output, and error streams is provided below.

```
Process p = Runtime.getRuntime ().exec (
                cmd,null,new File (``/usr/local/matlab2006a/bin/''));


BufferedReader MatlabOutput =
    new BufferedReader ( new InputStreamReader (p.getInputStream ()));
BufferedReader MatlabError  =
    new BufferedReader (new InputStreamReader (p.getErrorStream ()));
PrintWriter MatlabInput     = new PrintWriter (
    new BufferedWriter (new OutputStreamWriter (p.getOutputStream ())));
```

Now, the Java program can write characters to the MATLAB console through the MatlabIn-

put writer and retrieve responses via the MatlabOutput reader. In the case of the SEAD behavior model, the input will be a function call formatted as a string. The output will be any information printed to the MATLAB console. Generally, MATLAB functions store return values as a set of variables. Querying these variables after calling the function would be inefficient because it would be necessary to send multiple calls to the MATLAB runtime. Instead, print statements are placed at the end of a MATLAB function to print the return values before the function returns. These statements can be parsed in order to retrieve the return values. Sample code to call a function and retrieve the return values is provided below.

```
String cmd = "sead(" + xPosition + "," +
                       yPosition + "," +
                       heading   + "," +
                       speed     + "," +
                       targetHeading + ");\n";

MatlabInput.println(cmd);
MatlabInput.flush();

String results = MatlabOutput.readLine();
```

The code above assumes that the results are all written on a single line. More code is necessary if the results are split across several lines. Also, the results are still in a String format. This string must be parsed to obtain the individual values. In the case of the SEAD behavior models, the results included information such as predicted position, speed, and heading.

5.1.2 Behavior Recognition System

The Behavior Recognition system is a component written in Java. As stated before, the recommended method for interfacing this system to a SAF simulator is to wrap it as a web service, and deploy it on a Java application server. However, a different approach was taken due to time constraints and the need for a buffer between the two components. A web service was developed to serve as an intermediary between the component and the SAF simulator. Using this approach, Web Services is effectively connecting two separate components with neither component acting as a web

service.

The web service was developed to act as a buffer of information between the two components. The SAF simulator would upload symbols to the web service. Then, the behavior recognition component would retrieve these symbols, operate on them, and then upload the results to the web service. The SAF simulator could then retrieve the results from the web service. After an initial startup period, the components would poll the service at regular intervals to both push and retrieve data.

The code for this web service is long, but not complex. Appendix A contains a thorough walkthrough of the web services implemented for this work. It contains the code for the SEAD behavior model web service and the web service used with the Behavior Recognition system.

5.1.3 Deployment

Once a web service is constructed, it is deployed in a run-time environment on an application server. The Netbeans IDE supports several application servers, including Tomcat, Glassfish, JBoss, and WebSphere. Glassfish (Version 2 Update Release 2) is the application server used in this work. Deploying a web service can be as simple as uploading the WAR file to an application server via a web browser. There are also command-line tools and GUI frontends available. In particular, scripts can be written to use command-line tools to automatically redeploy a number of web services after a development milestone. Once a web service is deployed, it may be consumed by any application. The application only needs to download the corresponding WSDL file to determine the message format for SOAP messages.

5.1.4 Testing

The web service code was tested by developing simple clients within Netbeans. The automatic code-generation aided in quick development of these test clients. A test client was developed for each web service. The web service methods were called, passing in realistic input, and the output was analyzed to determine if the web services were operating correctly. Iterative testing ensured that most bugs were fixed.

5.1.5 Additional Features

When implementing these web services, several features were not considered. For example, Web Services standards exist for Security and Quality of Service. These features were not included due to unnecessary overhead and complexity. However, it would be worthwhile to explore the effects of adding security and encryption to the web services. These features could have a large impact on the simulation performance. On the other hand, if the simulation data is not sensitive, it would be acceptable for a non-participant to intercept the data.

5.2 Support for Web Services in JSAF

JSAF is written in C. Therefore, a tool must be used for parsing the WSDL file to produce client stub code in C. The tool used for this work is gSOAP. gSOAP contains two binaries for creating client stub code from a WSDL file, **wsdl2h** and **soapcpp2**. **wsdl2h** transforms the WSDL file into a C header file. This header file defines several different data types and contains several function prototypes. **soapcpp2** compiles this single header file into a set of header and source files. These files implement the functions contained in the first header file produced by **wsdl2h**. A brief description of each file is below.

soapH.h	Header file associated with soapC.cpp
soapStub.h	Header file associated with SoapClient.cpp
PortBinding.h	Namespace mapping file
soapC.cpp	Contains routines for serializing/deserializing data
SoapClient.cpp	Contains routines for accessing remote procedures

The following commands will create stub files for JSAF, which will allow it to operate as a web service client. See the gSOAP documentation on-line for an explanation of the command-line options.

```
> wsdl2h -s -t ~/gsoap-linux-2.7/bin/typemap.dat -o tmp.h
  http://localhost:8081/JSAFTracking/JSAFTrackingService?wsdl
  http://localhost:8081/seadRealtime/seadRealtimeService?wsdl

> soapcpp2 -C -p soap -x -w -L -I:~/gsoap-linux-2.7/import tmp.h
```

JSAF is built up of hundreds of different libraries. There is a library for the GUI, a library for sensors, a library for time management, libraries for each behavior within JSAF, and several others. A Web Services library was developed to include the client stub functionality in JSAF. The library primarily consists of the files listed above. Two other files, stdsoap2.h and stdsoap2.c, were also included as dependencies of the above files. The Web Services library was appended to the list of JSAF's libraries which includes it in any future rebuilds of the simulator. As such, the functionality is available to any other libraries within JSAF. Thus, JSAF's internal SEAD behavior and the new Vehicle Tracking behavior will have access to all of the web service function calls. These behaviors will be discussed later.

The other method for integrating the web service client code into JSAF was to include client code in each library. For instance, the SEAD behavior would contain client stub code specific to the SEAD behavior model web services. Similarly, the Vehicle Tracking behavior would contain client stub code specific to the web service used with the Behavior Recognition system. This method is more logical, and was used at first, but resulted in a large number of files spread out through the simulator source directories. By combining all of the web service functionality into one place, maintaining the code was much easier. Further work could be done to automate the code maintenance. In this case, the client code could be distributed only to the libraries which depend on it, which would keep other libraries from incorrectly calling other web service methods.

5.3 Integration of SEAD Behavior

JSAF's internal SEAD behavior was extended to include the custom SEAD behaviors available through Web Services. When configuring a vehicle's behavior during run-time, the user selects which SEAD behavior will be executed. The custom behavior operates by sending the current position, speed, and heading of the aircraft to the SEAD web service. The web service then returns a

suggested position, speed, and heading. The aircraft is instructed to fly to the new position using the new speed and heading.

One complication that still arises with Web Services is the interface between JSAF and the web service. Although the the WSDL file contains the API, the programmer must still use the web service correctly. For example, the position coordinates must match, as well as speed and heading units. Furthermore, the MATLAB simulation is formed using discrete time steps of about 5 seconds each. In JSAF, the aircraft must fly to the new location using much smaller time increments, on the order of several milliseconds. Therefore, JSAF must interpolate between the old position and the new position as best as possible. Also, in the MATLAB simulation, the aircraft fires when within range. In JSAF, firing is controlled by another software library and is somewhat unpredictable.

Thus, the API must be very clear as to what the expected parameters are and what will be returned. The MATLAB simulation may be improved without any need to change the simulator. Likewise, if any changes are necessary within the simulator's source, it is not necessary to modify the web service. This modular development is a key advantage to using Web Services.

5.4 Integration of Behavior Analysis and Prediction System

The Vehicle Tracking behavior is a new behavior inserted into JSAF. The behavior tracks a single vehicle and uploads tracking information to a Vehicle Tracking web service. The tracking information includes position, speed, and heading. The data is obtained directly from the tracked vehicle and not from a sensor onboard the tracking vehicle. Obtaining the data directly from the tracked vehicle is good for evaluating the Behavior Recognition system, but it will be important to obtain data from the tracking vehicle's sensors at the later testing stages for more realistic data. In particular, the behavior recognition system would need to deal with any noise in the system.

Inserting new behaviors into JSAF was not a simple task. The documentation is outdated, requiring reverse-engineering to determine how behaviors exist in the system. However, the addition of a new behavior is now documented for future reference. It requires using a template to develop a behavior library, and modification of several files under JSAF's main source directory. Then, there are other more obscurely located files that must also be edited. However, once the framework is in place, editing the behavior is relatively easy. JSAF does provide documentation for its more common libraries. In particular, the Vehicle Tracking behavior uses the libentity library's API for

most of its functionality, directly accessing entity attributes such as position and speed. It also uses the newly added Web Services library to send this attribute data to the intermediate web service which interacts with the Behavior Recognition system. Besides sending attribute data, the Vehicle Tracking behavior also polls for the last recognized behavior.

On the other end, the Behavior Recognition system polls the web service to obtain the uploaded tracking information. This information is then processed and a predicted behavior is returned to the web service. The details of the data processing may be found in Chapter 6 under the Background section.

5.5 Load Balancing

Load balancing is the process of distributing the load across several nodes. There are several methods for load balancing. Tao et al [35] describe two efficient algorithms for selecting web services with Quality of Service (QoS) constraints. Kuipers et al [14] provide a second overview of constraint-based selection algorithms. These algorithms are used to determine the optimal path through several services. In this work, the SAF system is choosing between several servers, and not a path through several services. This simplifies the work needed considerably.

Menascé [16] describes QoS issues in Web Services. Four qualities that every service has is availability, security, response time, and throughput. For this work, we are not as interested in security. However, availability, response time, and throughput are all very important. Availability is simple to determine. A plausible solution for determining response times would be to periodically poll the web service to determine an average response time. However, an average response time would require polling the service several times. In effect, the system would contribute to slowing down the service. Similarlly, throughput is not an easy measure to obtain. Instead, our work predicts the QoS based on the current computer load as determined by the measured CPU load, number of users, number of running processes, etc. Using these values, a broker system can indirectly estimate the QoS of each server, and the total QoS available.

The load-balancing mechanism in this work is a web service acting as a broker, similar to the work done in [34]. The web service periodically gathers load data from a pre-determined list of web servers. This load data includes the CPU load, number of users, number of running processes, percentage of free swap memory, and the ping time between the server and the workstation running

JSAF. The data is obtained using Nagios. Nagios is a software system which monitors computers and stores various statistics. An add-on for Nagios, NDOUtils, dumps the various statistical measures to a MySQL database. The web service accesses this database to get the load data. The data is then used to compute a QoS measure.

This measure is computed as a weighted sum of the load values. The weights provide a method for normalizing the load values, such that one value is not necessarily more important than another. The weights may also be biased to prioritize some load values over others. For example, one may choose to weight the CPU load higher than the amount of free swap memory available. The equation below is a very simple expression for the QoS measure.

$$\text{QoS} = \sum_{i=0}^{n} w_i load_i$$

After computing the QoS measures for each server, the servers are sorted from best performance to worst performance. A total QoS measure is then computed as the sum of the individual QoS measures. Dividing the individual QoS measure by the total measure produces a discrete probability distribution. This distribution is used when choosing a web server for the client. This method ensures that the top server does not necessarily receive every client. Also, as clients are assigned to servers, the distribution will change to properly reflect the quality-of-service provided by each server.

Chapter 6

Behavior Analysis

6.1 Introduction

A Behavior Recognition system was developed to recognize a behavior given some history of the vehicle or entity executing the behavior. In Chapter 5, the implementation of a Vehicle Tracking behavior was described. This behavior interacts with the Behavior Recognition system via a web service, also described in Chapter 5. To provide proof of concept, several simple tests were performed using the Vehicle Tracking behavior with the Behavior Recognition system. The next section provides background on the Behavior Recognition system. The background is taken from an earlier unpublished paper [23], and should provide enough information to understand the general operation of the Behavior Recognition system. The remaining sections will describe the simple tests and the results obtained.

6.2 Background

The Behavior Recognition system uses Markov models to represent behaviors. The system attempts to match input data streams to behaviors in a dictionary. A dictionary is a set of behaviors. If the behavior does not match a known behavior, a Markov behavior model is dynamically generated and may be added to the dictionary.

The system has three phases: (i) transform the input data streams into symbol streams; (ii) determine if the symbolized stream matches any of the a priori generated behavioral models;

and (iii) return the matched behavior to the SAF simulator. The system requires an alphabet to transform the data stream into a symbol stream. For example, suppose the system is tracking a vehicle based solely on distance. Values of 0-10m may be assigned to a value of A. Distances from 11-20m may receive a value of B, and so on. The alphabet defines this transformation. The choice of the optimal alphabet to symbolize the input data streams is an open problem. An alphabet with a logical meaning for the behavior being investigated was chosen. For example, patrol behaviors are generated using an alphabet that recognizes the difference in x and y from an initial position, while follow behaviors recognize changes in distance and the difference in vehicle headings.

To determine if the symbolized stream matches any behavior models, a bootstrapping process and confidence interval approach is used. The details of the bootstrapping process and confidence interval are not important to this work. However, if the reader is interested, an explanation is available in [5]. The following algorithm is used to test the data streams against the dictionary of behaviors.

```
1.  Retrieve a set of 15 data items from the web service
2.  Symbolize the input data items
3.  For each individual symbolized data item
  a.  Place the item into the active data window
  b.  Set a maximum of 30 items
  c.  If the set of behaviors that recognized the previous window is empty
    i.   Use the bootstrap process described above to find behaviors that
           potentially match the current window
    ii.  If none are returned, continue to the next individual symbolized
           data item
  d.  For each behavior in the set returned from step 3c
    i.   Use a confidence interval approach to find the percent of
           transitions falling within their respective confidence interval
    ii.  If the percentage is greater than the threshold needed for the
           behavior, then accept the model as a match for the window
    iii. Else, remove the model from consideration
  e.  Return the behavior with the highest percent of transitions falling
```

```
within their respective confidence intervals to the web service
```

Note that this process returns the model with the highest percentage of transitions falling within their respective confidence intervals. There may be multiple models that match the data window. The key to handling this occurrence is to note the class of the models matching the data window. For example, if three different flanking behaviors match the window, we may reasonably assume that the data under analysis represents a flanking behavior. If two flanking behaviors match and one orbit behavior matches, we cannot conclude with 100% accuracy that the behavior is not an orbiting behavior, but must wait for further input data that either drops the flanking behaviors or drops the orbit behavior from consideration. For the following tests, it is sufficient to only consider the behavior with the highest percentage of transitions. The above method is extensible and can be made to consider all possible interpretations of the input data stream, instead of only the behavior with the highest percentage of transitions.

6.3 Testing

Several scenarios were developed in JSAF to test the behavior recognition system. Due to its detailed implementation in JSAF, the M1 tank model is used for the majority of the experiments. The scenarios generally include two tanks, a tank doing the tracking (represented by BLUE) and the tank being tracked (represented by RED). In the case of air scenarios, the tracked vehicle is the F-18EF Superhornet fighter jet.

Scenarios were developed for all of the behaviors modeled in the dictionary, and additional scenarios were generated to perform a statistical analysis of the flanking behaviors. The statistical analysis provides a quantitative (and in many ways, qualitative) determination of the effectiveness of the behavior recognition system. The results of the analysis of the flanking behaviors are provided below along with a description of the tests that were run.

6.3.1 Distance Tests

Normal and wide flanking behaviors were run ten times each with small variations in the tracked (RED) vehicle's driving path introduced for each test-run. RED started at a range of 1km and flanked BLUE on the right side. At ranges of 800m, 600m and 400m, the predicted behavior was noted as "flanking" or "not flanking". Tables 6.1 and 6.2 display the results of each test. A plus sign

(+) indicates the behavior was identified as a flanking behavior, while a minus sign (-) indicates the behavior was identified as a "non-flanking" behavior. The results show that the behavior is usually predicted at the beginning of a test run, but is not always predicted as the run progresses. These results are due to RED beginning its turn toward BLUE. At this moment, RED no longer exhibits a flanking behavior, but instead exhibits an assault behavior. An assault behavior is characterized by RED heading directly at BLUE. Since RED's path is longer than the wide flanking behavior, the behavior was noted at the 900m mark as well. The 900m mark is roughly equivalent to the 800m mark in the normal flanking behavior.

	1	2	3	4	5	6	7	8	9	10
800m	+	+	+	+	+	+	+	+	+	+
600m	-	-	-	+	+	+	-	+	-	+
400m	+	+	-	-	-	-	-	-	-	-

Table 6.1: Normal flanking behavior

	1	2	3	4	5	6	7	8	9	10
900m	+	+	+	+	+	+	+	+	+	+
800m	-	-	+	+	-	+	+	-	+	+
600m	-	+	-	+	-	-	-	-	-	+
400m	-	-	-	-	-	-	-	-	-	-

Table 6.2: Wide flanking behavior

A second set of distance tests was carried out using a "non-flanking" behavior. RED approaches as if he will flank BLUE, but instead of turning toward BLUE, RED continues on a straight path, passing by BLUE at a range of approximately 300m. Since the path does not curl in, the behavior recognition system continues to predict a flanking behavior until RED is in a position where he can no longer flank BLUE. A series of 10 tests were used to determine the average range at which the behavior recognition system stopped predicting a flanking behavior (i.e. RED has safely passed by). A confidence interval was computed to provide bounds. Table 6.3 shows the results of the tests. The average range is 814m with a confidence interval of 32m.

	1	2	3	4	5	6	7	8	9	10
Range	792	795	798	818	783	760	828	787	913	867

Table 6.3: Non-flanking behavior

6.3.2 Behavior Coverage Test

Tests were also run to determine how much of the symbol stream did not match a flanking behavior. The normal and wide flanking scenarios were used. Results from twelve tested paths indicate that approximately 40% of a given flanking path does not match the flanking behavior. We provide the percentage of the path that did not match the flanking behavior in Table 6.4. The inability to capture 40% of the path may be due to the granularity of the alphabet used to symbolize the data. The flanking behaviors in the dictionary were designed to be flexible enough to handle small variations in the test path. If the variations in the test paths exceed the limits of the models, then the granularity chosen for the alphabet is probably too high.

	1	2	3	4	5	6	7	8	9	10
Normal	37.0%	31.3%	37.3%	33.9%	34.8%	36.5%	43.2%	33.8%	43.5%	44.5%
Wide	41.0%	40.4%	36.5%	55.3%	42.5%	37.5%	30.4%	27.8%	33.5%	55.6%

Table 6.4: Behavior coverage

Normal Average (w/ confidence interval): 37.5 $\pm$ 3.26%

Wide Average (w/ confidence interval): 40.0 $\pm$ 6.69%

Chapter 7

Performance Analysis

7.1 Introduction

In theory, the simulation environment and all external components can run on a single computer. However, the true power of the Web Services approach is executing the external components on remote servers. This would allow a powerful server to handle the CPU-intensive operations, while a local workstation running the simulation environment only needs to be concerned with sending and receiving network packets.

To evaluate the performance of using remote servers, tools were used to (1) scale the CPU frequency of the workstation and the remote servers, and (2) delay network traffic to and from these computers. Of most concern is the CPU frequency scaling of the workstation, which effectively emulates various CPU loads. As the simulation environment loses the CPU to other processes, it is more necessary to offload the computation elsewhere. Similarly, if network traffic delays are large enough, a single computer setup may perform better.

7.2 Setup

Performance analysis was done using three different computers. The three computers are referred to as WORKSTATION, LAPTOP, and CLUSTER for ease of discussion. WORKSTATION runs the JSAF simulation environment. LAPTOP and CLUSTER are remote servers. Each server runs a web server which hosts two web services, *seadRealtime* and *behaviorAnalysis*. These services are

discussed in Chapter 5. See Figure 7.1 for an illustration of the setup. The single measure compared in all scenarios is the mean response time (computed from 250 iterations) for a specific SOAP call or series of SOAP calls. For *seadRealtime*, the tick() call was used, whereas for the *behaviorAnalysis* web service, an average of several SOAP calls was used.

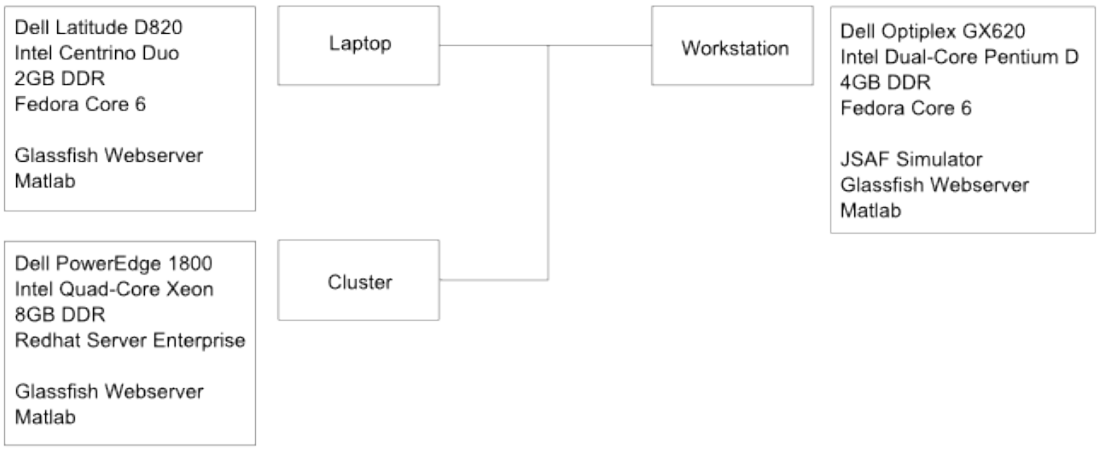

Figure 7.1: Performance Analysis Setup

7.3 CPU Performance Analysis

Usually, the term "CPU performance analysis" means profiling code to determine bottlenecks, and developing methods for speeding up processes. In this context, CPU performance analysis is used to answer the question: how well does the system perform under various CPU loads? In the case of all processes executing on a single computer, the system performance is expected to decrease as the CPU load increases. However, when linking to a remote server, the system performance is expected to remain steady as CPU load increases. The only CPU usage should be in the form of network calls, and network calls are generally not CPU-intensive. The next section discusses the tools and methods used in analyzing the system performance with respect to increasing CPU loads. After, a quantitative analysis over several scenarios is provided to test the above hypotheses.

7.3.1 Tools / Methods

Analyzing system performance based on the CPU load requires a method for emulating different CPU loads. CPU frequency scaling was found to be an effective solution. By increasing

or decreasing the CPU clock speed, the execution time is directly altered. Two Linux command-line tools for frequency scaling are **cpufreq-info** and **cpufreq-set**, both found in the cpufreq-utils package. **cpufreq-info** lists the current CPU frequency scaling info and policy governor. **cpufreq-set** allows the user to configure the current CPU frequency scaling settings. The syntax for these commands as used to analyze performance is as follows:

```
> sudo cpufreq-set -c 0 -f freq
> sudo cpufreq-set -c 1 -f freq
> sudo cpufreq-info | grep "current CPU frequency",
```

where the frequency *freq* is varied to emulate different CPU loads. The first and second lines set the CPU frequency for each CPU. The third line verifies the set frequency for all CPUs with a call to hardware.

There are two prerequisites for using these tools. First, the CPU(s) must support frequency scaling. Second, the frequency scaling kernel module specific to the CPU model must be loaded. Most Intel processors use either the "p4-clockmod" module or the "speedstep-centrino" module. AMD processors usually use the "powernow" kernel module.

7.3.2 Test Scenarios

Web Server on WORKSTATION

The web server is run on WORKSTATION. JSAF runs the *seadRealtime* custom scenario, which transmits a SOAP call every 5 seconds. The CPU frequency on WORKSTATION is set to 700Mhz and response times are collected for the SOAP calls. This process is repeated for CPU frequencies of 1.40Ghz, 2.10Ghz, and 2.80Ghz. The results are given in Figure 7.2. As the CPU frequency increases, the response times decrease.

Web Server on LAPTOP

The web server is run on LAPTOP. JSAF again runs the *seadRealtime* custom scenario. The CPU frequency on WORKSTATION is again scaled between 700Mhz and 2.80Ghz in increments of 700Mhz, while the CPU frequency on LAPTOP is constant at 2.0Ghz. The response times are collected. The results are given in Figure 7.3. The results indicate that the mean response time

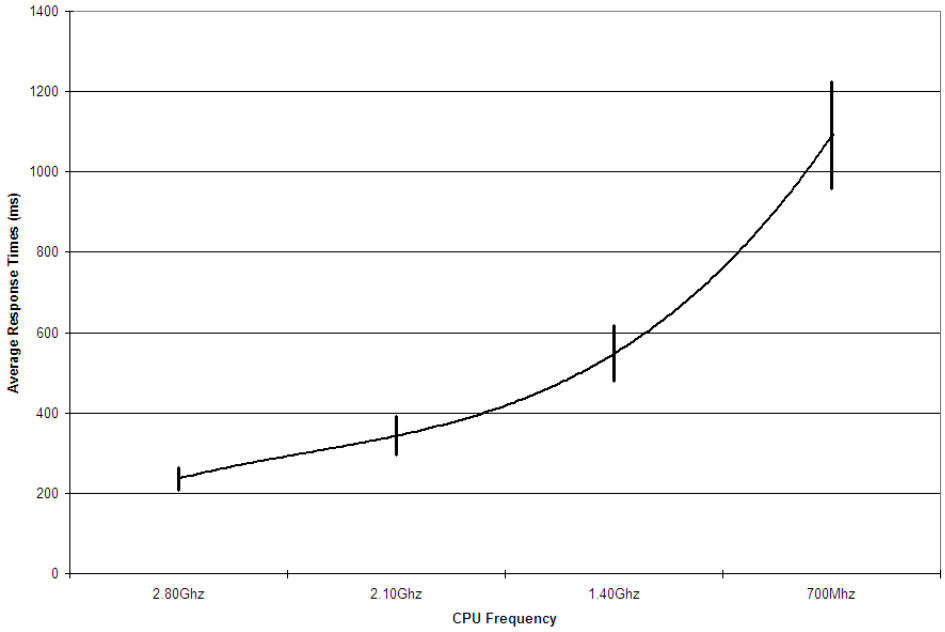

Figure 7.2: Response Times for WORKSTATION

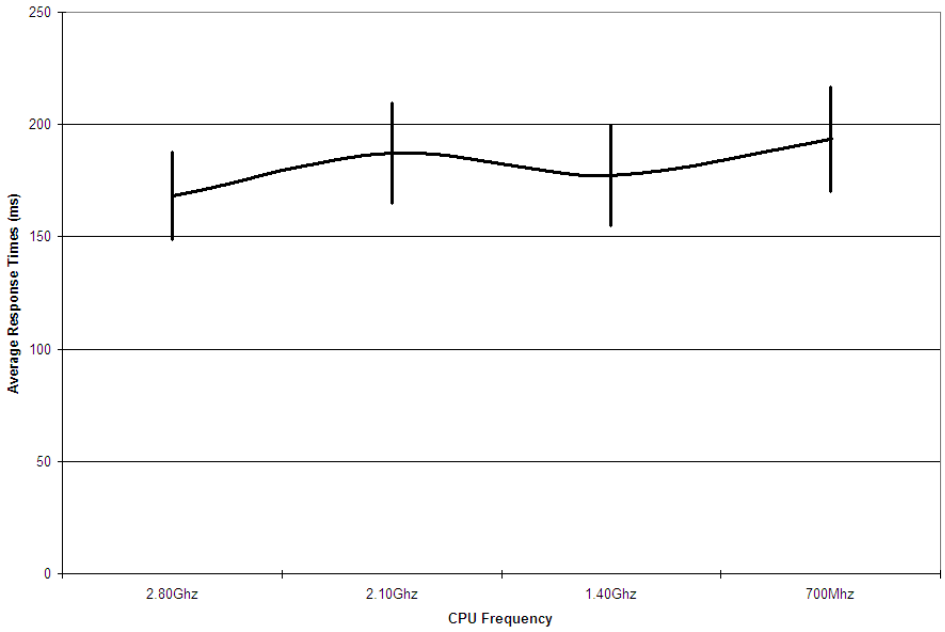

Figure 7.3: Response Times for LAPTOP

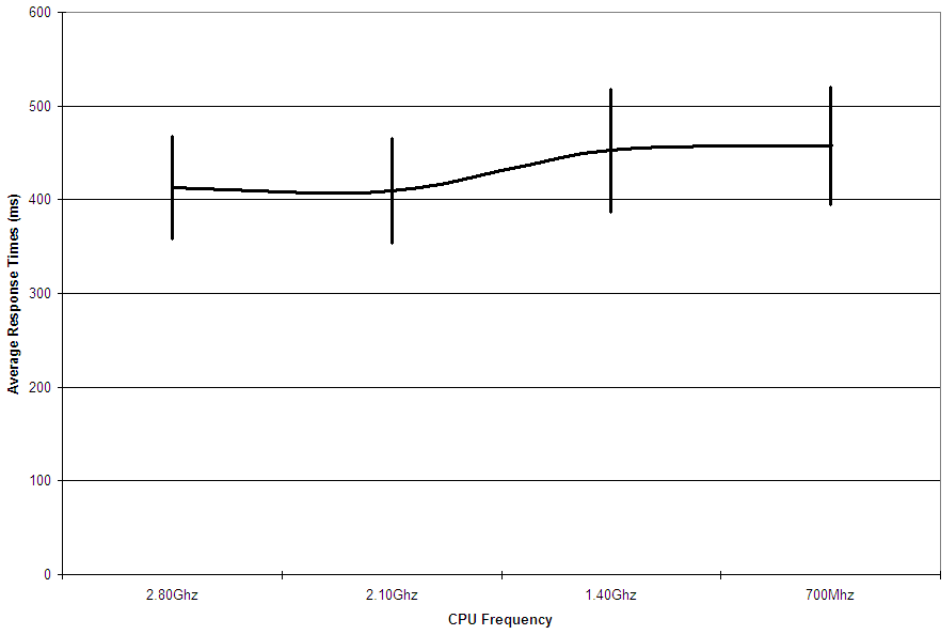

Figure 7.4: Response Times for CLUSTER

is effectively independent of WORKSTATION's cpu frequency. Furthermore, the mean response times for all 4 frequency steps are less than the best response time in the previous setup. This suggests that a remote server can be very effective.

Web Server on CLUSTER

The web server is run on CLUSTER. The steps are identical to the previous setup. The results are shown in Figure 7.4. The results provide further proof that the mean response time is effectively independent of cpu frequency.

7.4 Network Load Analysis

Several advances in network technology have provided almost unlimited bandwidth in today's networks, but traffic delays can still occur. From JSAF's point of view, network delay (or load) may result from an excessive amount of packets on the network, or it may result from a slow server. It may even result from a noisy line (ie. dropped packets, scrambled packets, etc). Whatever the case, it is imperative that the system not rely completely on the network. It was predicted that an

increase in network delay will show an increase in response times, simply because they are directly related. These tests are trivial, but they are important for a complete understanding of how the system reacts to loads on the network. Below, the tools and methods used in emulating network delay are discussed. Then, a quantitative analysis is provided to test the above hypothesis.

7.4.1 Tools / Methods

Analyzing the system from a network standpoint requires a method for emulating a slow network since actually slowing down the network would be difficult to control for testing purposes. The method chosen was to use the traffic control tool, **tc**, written by Alexey N. Kuznetsov and added in Linux 2.2. **tc** provides control over several queues and classes which filter incoming packets. The commands used in this project are below.

```
>  sudo tc qdisc add dev eth0 root handle 1:0
   netem delay mean bound distribution normal

>  sudo tc qdisc del dev eth0 root
```

The first command adds a queue to the ethernet port under the root node. Adding a queue to the root node forces all IP traffic over eth0 to pass through this queue. The queue adds a time delay to each packet using a normal distribution. The *mean* time value is the average delay added to each packet. The *bound* time value provides an upper and lower limit for the time delay. For example, "netem delay 50ms 5ms distribution normal" adds a 50ms $\pm$ 5ms delay. The second command removes the queue, allowing the ethernet port to function normally again.

tc also allows the user to emulate packet loss, packet duplication, packet corruption, and packet re-ordering. For simplicity, these cases were not taken into consideration when analyzing the system performance with various network delays. However, a complete study would certainly test all possible scenarios.

7.4.2 Test Scenarios

Web Services on LAPTOP with Slow Network

The web server is run on LAPTOP. JSAF runs the *seadRealtime* custom scenario. The CPU frequency on WORKSTATION is set to 2.80Ghz. The network delay between WORKSTATION and LAPTOP is set to be a normal distribution with a mean of 50, 100, 200, and 500ms. The response times are collected. The results are displayed in Figure 7.5. As is expected, the response times increase for each increment in network delay.

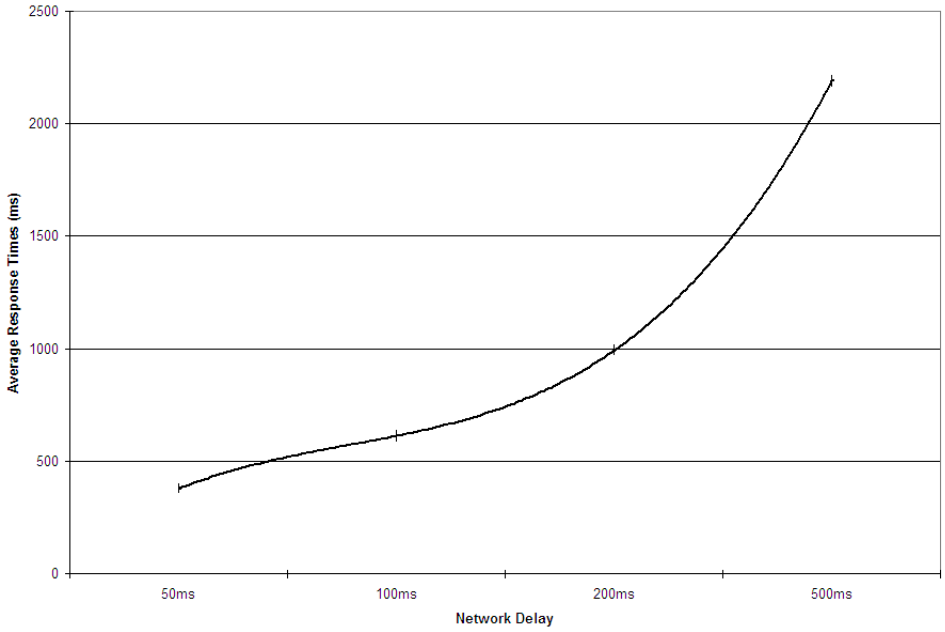

Figure 7.5: Response Times for LAPTOP with Slow Network

Web Services on CLUSTER with Slow Network

The web server is run on CLUSTER. JSAF runs the *seadRealtime* custom scenario. The CPU frequency on WORKSTATION is set to 2.80Ghz. The network delay between WORKSTATION and CLUSTER is set to be a normal distribution with a mean of 50, 100, 200, and 500ms. The response times are collected. The results are displayed in Figure 7.6. As is expected, the response times increase for each increment in network delay.

Web Server on Slow LAPTOP

The web server is run on LAPTOP. JSAF again runs the *seadRealtime* custom scenario. The CPU frequency on WORKSTATION is constant at 2.80Ghz, while the CPU frequency on LAPTOP is scaled between 1.00Ghz and 2.00Ghz in increments of 333Mhz. The response times are collected. The results are displayed in Figure 7.7. As expected, the response times increase.

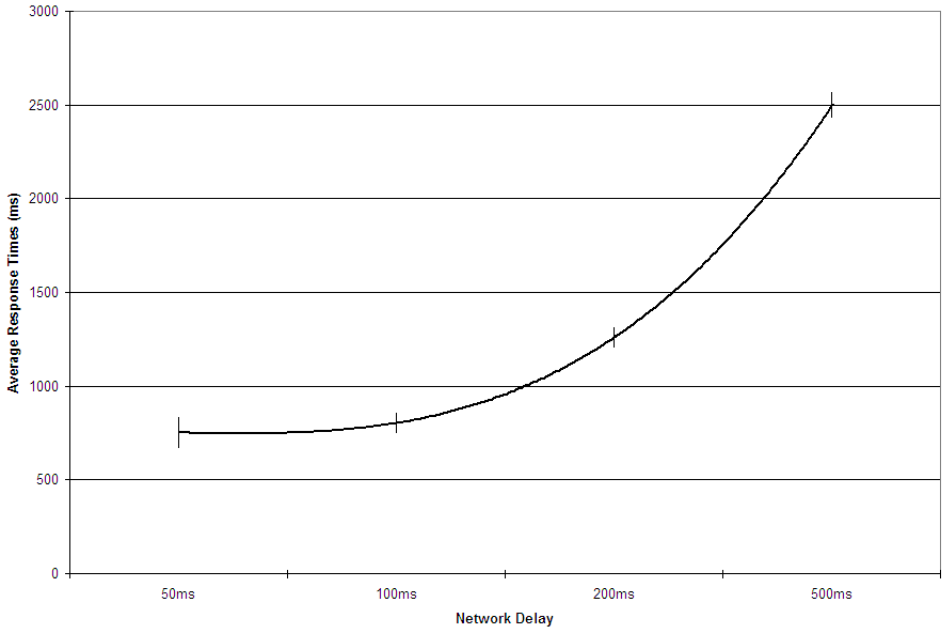

Figure 7.6: Response Times for CLUSTER with Slow Network

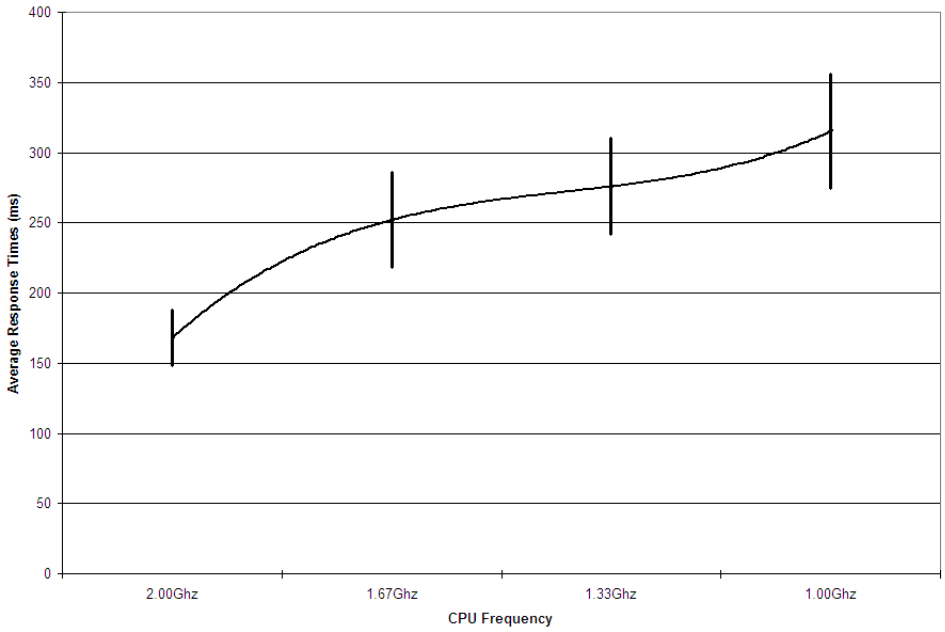

Figure 7.7: Response Times for Slow LAPTOP

Chapter 8

Load Balancing Analysis

8.1 Introduction

In the previous chapter, experiments showed that using network machines can improve system performance. The experiments also showed that a slow network could be detrimental to performance, making it worse than it would have been otherwise. The load-balancing mechanism was designed to choose the optimal server, whether it be a local or network machine. Here, the performance of the load-balancing mechanism will be examined by comparing it to a scenario where load-balancing is not used.

While the load-balancing should have a good effect on the overall system performance, it does add extra overhead into the system. Therefore, it is necessary to quantify the amount of overhead to understand when load-balancing is not good. One scenario in particular is when there is only one server available. Another scenario is the case when the web server running the broker service is congested or heavily loaded. Both scenarios will be examined to determine the cost of using load-balancing versus not using it.

8.2 Benefits of Load-Balancing

Setup

A new machine configuration was used to determine the benefits of using a load-balancing mechanism. While using physical machines, such as in the last set of experiments is acceptable for the previous tests, it was decided that more machines would more accurately determine how well the load-balancing mechanism worked. The approach used was to build 5 virtual machines using VMWare. VMWare is an application which emulates full machines, but in reality, these machines are just processes on a host machine. Figure 8.1 shows the setup of the virtual machines. The first machine hosts the broker web service, Nagios, and the MySQL database used with Nagios. The other 4 machines all host the SEAD behavior model web service. The host machine is the workstation from the last round of experiments, and it runs JSAF and its own web server which hosts the SEAD behavior model service. Therefore, we have 5 machines with the SEAD service and one machine with the broker service. The 4 VMs hosting the SEAD service were set to varying process priorities, effectively changing the quality-of-service seen from each server. A lower priority will be a slower machine and vice versa.

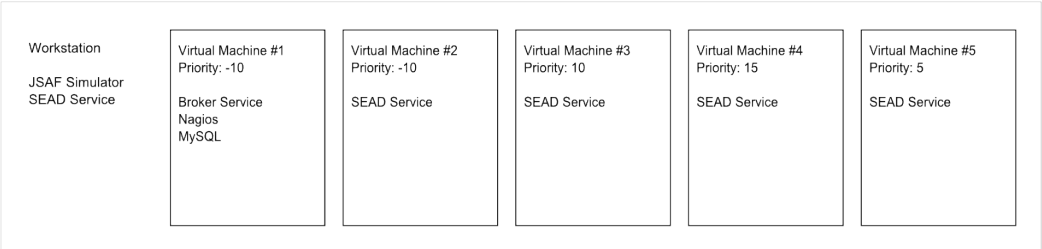

Figure 8.1: Load Balancing Experiment Setup

Once the machine setup was complete, the experiments were run. First, The broker service was configured to choose the server based on the probability distribution discussed earlier in the thesis. JSAF was started with 10 aircraft running SEAD behaviors. One thousand response times were gathered, approximately 100 response times per aircraft. The average response time was determined over all 1000 response times, as well as the 95% confidence interval. Then, the broker service was configured to choose the server from a uniform distribution, and the experiment was run again.

Results

Running these two scenarios, the configuration which intelligently chooses the server is expected to perform better. The results are available in Figure 8.2. Indeed, the intelligent configuration does perform better, and by a seemingly large amount. However, more tests under various different scenarios are needed to completely determine if these results are accurate or not.

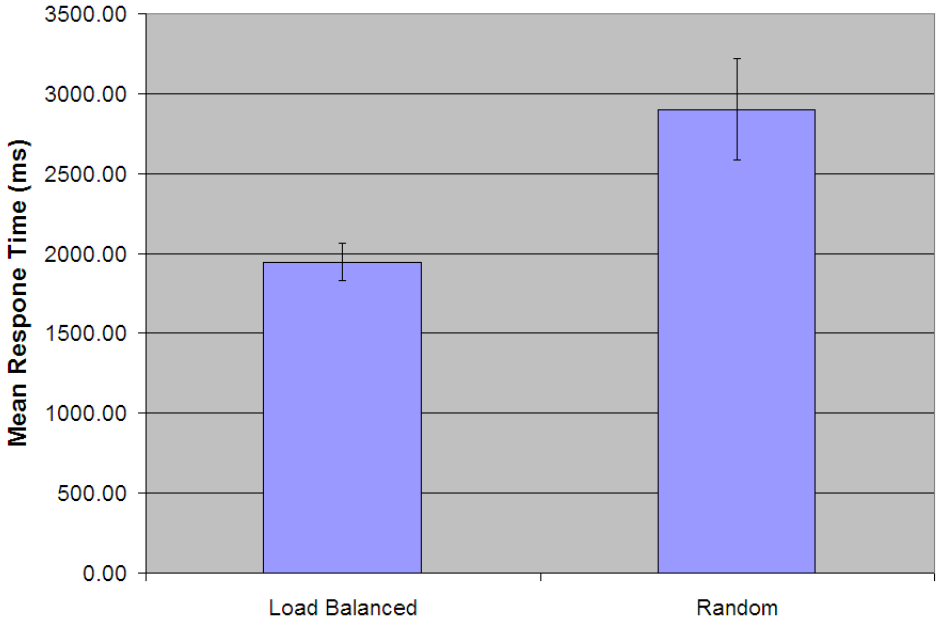

Figure 8.2: Benefits of Load-Balancing

8.3 Drawbacks of Load-Balancing

Setup

The setup for these experiments are very similar to the previous experiments. The differences are as follows. For the first experiment, only one server will be available. Response times will be gathered for the SEAD service always choosing that particular server. Then, response times will be gathered for the SEAD service querying the broker service to get the server address. The average difference in response times should measure the overhead of using the load-balancing mechanism. For the second experiment, the process priority of the virtual machine running the broker service will be

lowered significantly to emulate a heavily-loaded server. The process priorities for the other virtual machines will remain the same. Therefore, while there was an improvement in system performance in the last set of experiments, there is not expected to be the same improvement in this set. In fact, it is possible that there could be a negative effect in system performance when using the load-balancing mechanism.

Results

In the first experiment, the average overhead was determined for the broker service. This overhead is estimated at 40% of the tick cycle time. However, the overhead is dependent on the time required to run the behavior service. Therefore, a longer behavior service run time would decrease this value. This result suggests that a behavior model should only be developed into a web service if (a) the execution time is large and/or it requires a lot of resources or (b) the other benefits of using Web Services are desired such as the dynamic-linking capabilities.

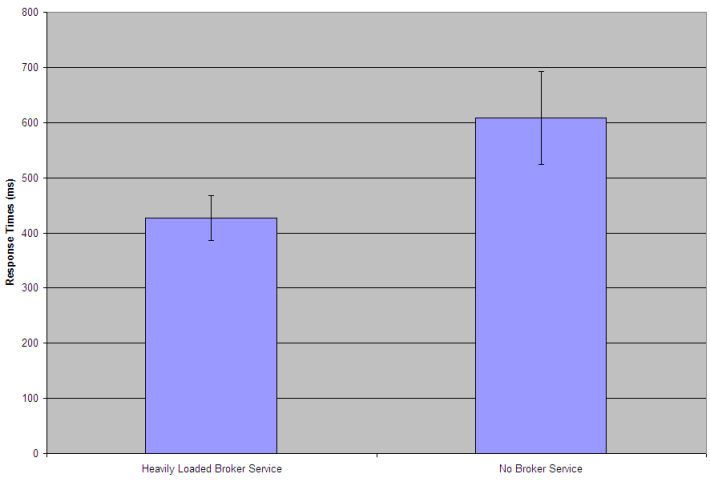

Figure 8.3: Heavily-Loaded Broker Service vs No Broker Service

In the second experiment, the system performance was examined for a heavily-loaded broker service. The results are available in Figure 8.3. These results are a bit unexpected since the heavy load should have negated the help of the load-balancing mechanism. Instead, these results seem to further affirm that load-balancing can be of great help. Even though the load balancing mechanism introduces significant overhead, the performance can still be improved over not using a load-balancing

approach. The key fact to note here is the different priorities set to the virtual machines, which allow an intelligent approach to help.

Chapter 9

Conclusions

9.1 Summary

A method for integrating remote components into an HLA simulation environment is needed. The solution proposed is Web Services. Web services are simple and easy to implement. Network transfer is acheived using the SOAP protocol, which almost guarantees interoperability between different operating systems and programming languages. The other portion of interoperability is in the form of a well-documented API to ensure the data requested is the data returned.

In this work, several web services are developed. Two web services are behavior models. A third interfaces with behavior recognition software. The fourth implements a QoS broker agent. The QoS broker agent performs load balancing among all available servers to improve system performance.

A Web Services library is added to JSAF. This library contains the header and source files necessary to invoke the web services. These header and source files were created using GSoap, and need to be recreated each time a web service interface is changed or added. The function calls implemented in these source files are included in the JSAF behaviors modified for this thesis. The first modified behavior is the Fixed-Wing Aircraft Ground Attack behavior, and the second is a new Vehicle Tracking behavior.

Testing was performed to analyze this Web Services approach. System performace improved when computation was offloaded to a faster server, perhaps in the case that the simulator machine's CPU was heavily loaded. Furthermore, testing was performed with the load-balancing mechanism

in place. Preliminary results show that intelligently choosing a service endpoint will improve system performance more than selecting a server at random. The load-balancing mechanism is especially important when several clients are running at once.

9.2 Discussion

JSAF and OneSAF are very good SAF systems. However, they are stand-alone systems that must use internal behavior models and physical models. These models may be changed, but not without trouble. It is difficult to test new models with the current software. This work provides a method for connecting external components to the SAF simulation systems. These external components may serve any purpose. However, we are most interested in behavior models. Encapsulating a behavior model as a web service allows any SAF system to access this model. Furthermore, the user may dynamically load these behavior models at runtime. Suppose there are several hundred models available as web services. The user could choose from all the available models, rather than just the internal models shipped with the original SAF system. These models may change from day to day as they are improved. However, the SAF system user will not be required to perform a system rebuild each time a model is changed.

An advantage of using Web Services is the ability to perform load-balancing. Several identical models may be deployed in different servers. These models may be consumed by several SAF systems, executing what may be several hundred lines of code for a complex behavior model. The offloading of this computation frees up the SAF system user's CPU, which at the least would provide a more enjoyable experience. At most, the user may be running a very large simulation. By offloading most of the heavy computation, the simulation should run smoothly, even though thousands of entities may be moving around the battlespace. The drawback is it would require a good ethernet card and a very fast uplink to the network.

Web Services provides a good solution for deploying new behavior models and offloading computation. However, it should not necessarily be used for everything. HLA and DIS are still very good simulation protocols. HLA is designed to handle simulation management, and DIS is designed specifically for military simulation. It would be a step backward to apply the general nature of Web Services to the specific area of military simulation. On the other hand, merging the two technologies may prove beneficial. Having a RTI accessible through Web Services could further increase the

interoperability between federates and the RTI. The same applies for federates modeled as Web Services. If done correctly, the basic architecture of HLA would not change, but the interoperability between components would greatly increase.

This work is only a small part of the large amount of work in service-oriented architectures SOAs). While most SOA research is done with regard to the business world, there is no reason that SOA principles cannot be applied to other areas, like military simulation. In a sense, SOA is very similar to how people operate. There is not just one barber shop, but several. In the same way, there should not be just one absolute simulation model. Not every soldier or aircraft will act exactly the same. The differences may be enough to shed light on any vulnerabilities. For example, a simulation may show that an attack is successful 49 out of 50 trials. One may conclude that the attack will work with 98% certainty. However, the interesting trial is the one that didn't succeed. It is important to understand why it didn't succeed and find ways to make it succeed. Therefore, as Web 2.0 stresses the individuality of the users, and not the providers, it is important that simulations are products of many different researchers, and not a select few. By using Web Services to expand SAF simulation systems, the knowledge of the world can be harnessed, rather than just that of a single group.

9.3 Future Work

Future work involves the development of more behavior models to test how well the system works. Whereas the models used in this thesis are developed in MATLAB, it might be appropriate to have models developed using C/C++, Java, or Prolog. They could then be more easily developed into a stand-alone web service.

Developing a Web Services library for JSAF which downloads the WSDL file during run-time and parses it to determine message formats may provide a better method for integrating web service calls into JSAF. It would remove the need to rebuild JSAF each time an interface is changed or added. However, the overhead and complexity may not be worth the effort.

More work with the load-balancing mechanism will improve system performance, specifically with respect to the weights used in the computation of the QoS metric. The weights are currently defined as they are because they work. A more thorough analysis will provide better weights, and thus a better metric for comparing servers.

Appendix

Suppression of Enemy Air Defenses (SEAD)

The SEAD behavior is a component written as a MATLAB function. There are several possibilities for communicating between Java and MATLAB. The Java Native Interface (JNI) is one solution, although difficult to implement. Another solution is to create a Java extension in the MATLAB function. This extension would interface with the MATLAB variables and communicate through some form of IPC to the Java web service. However, this solution requires a lot of extra implementation. Furthermore, both solutions are specific to MATLAB. A general solution that will work for components written in any computer language is desired. The chosen solution is to execute the component within its runtime environment, then access the standard I/O of the component. With this solution, the communication is provided using the standard I/O which makes the communication trivial.

The SEAD web service consists of three methods. These methods are *start*, *stop*, and *tick*. *start* and *stop* methods start and stop the MATLAB run-time environment. *tick* encapsulates the SEAD behavior model by calling the MATLAB function and reading the results from the standard output stream. The reader may note that this differs from the API mentioned in Chapter 4. Because the MATLAB runtime environment must be started and stopped, two extra functions are publicly available for this purpose. First, the web service state variables will be discussed. Then, a walkthrough of each method will be given.

State Variables

```
private        Process p;
private BufferedReader output;
private BufferedReader error;
```

```
private    PrintWriter input;
private        boolean MatlabOpen = false;
```

p is the run-time process. *output* reads from the run-time output stream, *error* reads from the run-time error stream, and *input* writes to the run-time input stream. *MatlabOpen* is **true** if the run-time process is started correctly, and **false** otherwise.

Starting the MATLAB Run-time Environment

```
@WebMethod(operationName = ‘‘start’’)
public boolean start()
{
  if(MatlabOpen) return true;

  try {

    // Start MATLAB run-time environment; wait 15 seconds for startup to complete
    p = Runtime.getRuntime().exec (‘‘/home/mbennin/installs/matlab/bin/matlab’’);
    Thread.sleep(15000);

    // Get Matlab output stream
    output = new BufferedReader (new InputStreamReader (p.getInputStream ()));

    // Get Matlab input stream
    input = new PrintWriter (
             new BufferedWriter (new OutputStreamWriter (p.getOutputStream ())));

    // Get Matlab error stream
    error = new BufferedReader (new InputStreamReader (p.getErrorStream ()));

   // Clear input and output streams
   input.clear();
    output.clear();

    // Output contents of error stream, if any
    if(error.ready())
      while(!error.empty())
        System.out.println(error.readLine());

    // Change current working directory (cwd) within MATLAB
    output.println(‘‘cd /home/mbennin/Desktop/seadWork’’);
    output.flush ();

    // MATLAB run-time environment is now running
    MatlabOpen = true;

  }
  catch (Exception e) {
```

```
    System.out.println (``(start) error: `` + e.getMessage ());
  }

  return MatlabOpen;
}
```

An instance of the MATLAB run-time environment is called to execute. The thread sleeps for 15 seconds to allow MATLAB to fully start. Then, the input, output, and error streams are attached to their respective Reader and Writer objects. The input and output streams are cleared of any data, and the error stream is read. Finally, the cwd is changed within MATLAB so that the SEAD function may be run. Also, MatlabOpen is set to **true**, unless an exception was thrown.

Stopping the MATLAB Run-time Environment

```
@WebMethod(operationName = ``stop'')
public boolean stop()
{
  if(MatlabOpen)
  {
    output.println(``quit'');
    output.flush();

    Thread.sleep(5000);
    p.destroy();

    MatlabOpen = false;
  }

  return true;
}
```

If the MATLAB run-time environment is open, try to close it normally. Forcefully kill the process after 5 seconds to make sure the run-time environment is closed. Set MatlabOpen to **false**.

Calling the MATLAB Function

```
public class FlightCommands {

  public FlightCommands ()
  {
    x = 0; y = 0; heading = 0; speed = 0; altitude = 0; targetHeading = 0;
    error = false; err = ``'';
  }
```

```
  public FlightCommands(float _x, float _y, float h, float s, float a,
                        float targetH, boolean _error, String _err)
  {
    x = _x; y = _y; heading = h; speed = s; altitude = a; targetHeading = targetH;
    error = _error; err = _err;
  }

  public float x;
  public float y;
  public float heading;
  public float speed;
  public float altitude;
  public float targetHeading;
  public boolean error;
  public String err;
}

@WebMethod(operationName = ``tick'')
public FlightCommands tick(@WebParam (name = ``xPosition'')
                           int xPosition, @WebParam (name = ``yPosition'')
                           int yPosition, @WebParam (name = ``heading'')
                           float heading, @WebParam (name = ``speed'')
                           float speed, @WebParam (name = ``targetHeading'')
                           float targetHeading) {

  FlightCommands result = new FlightCommands();

  if(MatlabOpen) {
    String in,err;
    String cmd = ``sead('' + xPosition     + ``,'' + yPosition + ``,'' +
                             heading       + ``,'' + speed     + ``,'' +
                             targetHeading + ``);'';

    output.println(cmd);
    output.flush();

    try {
      while(error.ready())
        System.out.println(``(error) '' + error.readLine());

      if(input.ready())
      {
        in = input.readLine();
        if(in.contains(``>> ``))
          in = in.substring(in.lastIndexOf(``>> ``) + 3);
        System.out.println(``in: ``+in);
        while (in.contains(`` ``)) {
          in = input.readLine();
          if(in.contains(``>> ``))
            in = in.substring(in.lastIndexOf(``>> ``) + 3);
```

```
          System.out.println(``in: ``+in);
        }

        result.x = new Float(in).floatValue();
        in = input.readLine();
        result.y = new Float(in).floatValue();
        in = input.readLine();
        result.heading = new Float(in).floatValue();
        in = input.readLine();
        result.speed = new Float(in).floatValue();
        in = input.readLine();
        result.targetHeading = new Float(in).floatValue();
      } else {
        System.out.println(``(error) no input!'');
        result.error = true;
        result.err = ``error: error with params'';
      }
    } catch (IOException e) {
      System.out.println(``(tick) error: `` + e.getMessage());
      result.error = true;
      result.err = ``error: IOException'';
    }
  } else {
    System.out.println(``(tick) error: MatlabNotOpen'');
    result.error = true;
    result.err = ``error: MatlabNotOpen'';
  }

  return result;
}
```

Vehicle Tracking with Behavior Analysis and Prediction

The Vehicle Tracking behavior was designed from the start to be a Web service. Therefore, inter-process communication is unnecessary. A Java class for vehicle tracking was created and populated with several methods. The methods were either get or set methods for various values, such as the latest tracking information, the current predicted behavior, the time delay between the prediction and the time of the simulator, etc. The state variables and each method are described below.

State Variables

```
public class TrackingInformation
{
public TrackingInformation()
```

```
{
  x = 0; y = 0;
  rangeToTarget = 0.0; angleToTarget = 0.0; targetHeading = 0.0;
  timestamp = 0;
  command = ‘‘’’;
}

public TrackingInformation(int _x, int _y, float _rtt, float _att, float _th,
                           long _t, String _c)
{
x = _x; y = _y;
rangeToTarget = _rtt; angleToTarget = _att; targetHeading = _th;
timestamp = _t;
command = _c;
}

public TrackingInformation(String _c)
{
  x = 0; y = 0;
  rangeToTarget = 0.0; angleToTarget = 0.0; targetHeading = 0.0;
  timestamp = 0;
  command = _c;
}

  public   int x;
  public   int y;
  public float rangeToTarget;
  public float angleToTarget;
  public float targetHeading;
  public  long timestamp;

  String command;
}

ConcurrentLinkedQueue trackingInfo;
String predictedBehavior;

long lastGrabbedTime;
long lastAddedTime;
```

Initialize State Variables

```
@WebMethod(operationName = ‘‘startTracking’’)
@Oneway
public void startTracking(@WebParam(name = ‘‘vehEnvironment’’)
  String vehEnvironment)
{
  trackingInfo = new ConcurrentLinkedQueue();
  trackingInfo.offer(new TrackingInformation(vehEnvironment));
```

```
  PredictedBehavior = new String(``Unknown Behavior'');

  lastGrabbedTime   = System.currentTimeMillis ();
  lastAddedTime     = System.currentTimeMillis ();
}
```

Upload Tracking Information

```
@WebMethod(operationName = ``putTrackingInfo'')
@Oneway
public void putTrackingInfo(@WebParam(name = ``dx'')
  float dx, @WebParam(name = ``dy'')
  float dy, @WebParam(name = ``rangeToTarget'')
  float rangeToTarget, @WebParam(name = ``angleToTarget'')
  float angleToTarget, @WebParam(name = ``targetHeading'')
  float targetHeading)
{
  lastAddedTime = System.currentTimeMillis ();
  if(trackingInfo.size() < 127)
  {
    TrackingInformation info = new TrackingInformation(
      dx, dy, rangeToTarget, angleToTarget, targetHeading, lastAddedTime, ``'');
    trackingInfo.offer(info);
  }
}
```

Retrieve Position Relative to Origin

```
@WebMethod(operationName = ``getXYPositionRelativeToOrigin'')
public String getXYPositionRelativeToOrigin (@WebParam(name = ``length'')
  int length)
{
  // Check whether 'length' elements exist in queue
  if(trackingInfo.size () < length) return new String(``UNDERFLOW'');

  String result = new String (``'');
  TrackingInformation info;

  for (int i = 0; i < length-1; i++)
  {
    info = (TrackingInformation) trackingInfo.poll();
    if(info.commamnd.size())
      result += info.command + `` '';
    else
      result += info.x + ``,'' + info.y + `` '';
  }
  info = (TrackingInformation) trackingInfo.poll();
  if(info.commamnd.size())
```

```
    result += info.command;
  else
    result += info.x + ‘‘,’’ + info.y;

  lastGrabbedTime = info.timestamp;

  return result;
}
```

Retrieve Range to Target and Angle to Target

```
@WebMethod(operationName = ‘‘getRangeAndAngleToTarget’’)
public String getRangeAndAngleToTarget (@WebParam(name = ‘‘length’’) int length)
{
  // Check whether ’length’ elements exist in queue
  if(trackingInfo.size () < length) return new String(‘‘UNDERFLOW’’);

  String result = new String (‘‘’’);

  for(int i = 0; i < length-1; i++)
  {
    info = (TrackingInformation) trackingInfo.poll();
    if(info.commamnd.size())
      result += info.command + ‘‘ ’’;
    else
      result += info.rangeToTarget + ‘‘,’’ + info.angleToTarget+ ‘‘ ’’;
  }
  info = (TrackingInformation) trackingInfo.poll();
  if(info.commamnd.size())
    result += info.command;
  else
    result += info.rangeToTarget + ‘‘,’’ + info.angleToTarget;

  lastGrabbedTime = info.timestamp;

  return result;
}
```

Retrieve Range to Target and Target Heading

```
@WebMethod(operationName = ‘‘getRangeAndTargetHeading’’)
public String getRangeAndTargetHeading(@WebParam(name = ‘‘length’’) int length)
{
  // Check whether ’length’ elements exist in queue
  if(trackingInfo.size () < length) return new String(‘‘UNDERFLOW’’);

  String result = new String (‘‘’’);
```

```
  for(int i = 0; i < length-1; i++)
  {
    info = (TrackingInformation) trackingInfo.poll();
    if(info.commamnd.size())
      result += info.command + `` '';
    else
      result += info.rangeToTarget + ``,'' + info.targetHeading+ `` '';
  }
  info = (TrackingInformation) trackingInfo.poll();
  if(info.command.size())
    result += info.command;
  else
    result += info.rangeToTarget + ``,'' + info.targetHeading;

  lastGrabbedTime = info.timestamp;

  return result;
}
```

Upload Predicted Behavior

```
@WebMethod(operationName = ``putPredictedBehavior'')
@Oneway
public void putPredictedBehavior (@WebParam(name = ``PredictedBehavior'')
  String _PredictedBehavior)
{
  PredictedBehavior = _PredictedBehavior;
}
```

Retrieve Predicted Behavior

```
@WebMethod(operationName = ``getPredictedBehavior'')
public String getPredictedBehavior ()
{
  return predictedBehavior;
}
```

Retrieve Time Delay

```
@WebMethod(operationName = ``getTimeDelay'')
public long getTimeDelay ()
{
  return (lastAddedTime - lastGrabbedTime);
}
```

Upload Command

```
@WebMethod(operationName = ‘‘putCmd’’)
@Oneway
  public void putCmd (@WebParam(name = ‘‘command’’) String command)
{
  trackingInfo.add (new TrackingInformation(command));
}
```

QoS Broker Web Service

The QoS Broker web service keeps track of the several servers available to JSAF. QoS measures are kept for each server to form a probability distribution. Clients can register to use any of the servers. The QoS Broker then returns a server address to each client when requested. In this way, the client can be assured it is using the optimal server.

State Variables

```
private Vector<BrokerClient> clients;
private volatile BrokerServer[] servers;
private int numClients;

private BrokerSqlParser parser;

private boolean initialized = false;

@Resource
WebServiceContext wsCtxt;
```

Initialization

```
@WebMethod(operationName = "init")
public void init () {

  servers = new BrokerServer[5];

  servers[0] = new BrokerServer("http://172.16.178.1:8081/");
  servers[1] = new BrokerServer("http://172.16.178.129:8080/");
  servers[2] = new BrokerServer("http://172.16.178.130:8080/");
  servers[3] = new BrokerServer("http://172.16.178.131:8080/");
  servers[4] = new BrokerServer("http://172.16.178.132:8080/");

  parser = new BrokerSqlParser(servers);
  parser.start();
```

```
  clients = new Vector();
  numClients = 0;

  initialized = true;
}
```

Stop Web Service Nicely

```
@WebMethod(operationName = "destroy")
public void destroy () {
  if(initialized)
    parser.quit();
}
```

Register a Client

```
@WebMethod(operationName = "register")
public int register () {

  if(!initialized) init();

  MessageContext msgCtxt = wsCtxt.getMessageContext();
  HttpServletRequest req =
    (HttpServletRequest)msgCtxt.get(MessageContext.SERVLET_REQUEST);

  String clientIP = req.getRemoteAddr().trim();
  int id = numClients++;

  clients.add(new BrokerClient(clientIP,id));

  return id;
}
```

Unregister a Client

```
@WebMethod(operationName = "unregister")
public synchronized void unregister (@WebParam(name = "id") int id) {

  if(!initialized) init();

  // invalid id
  if(id < 0 || id >= numClients) return;

  // invalid id
  int index = clients.indexOf(new BrokerClient("",id));
  if(index < 0) return;
```

```
  BrokerClient client = clients.elementAt(index);

  if(client.server != null)
    client.server.clients.remove(client);

  clients.remove(id);
  numClients--;
}
```

Connect Client to a Server

```
@WebMethod(operationName = "connect")
public synchronized String connect (@WebParam(name = "id") int id) {

  if(!initialized) init();

  // invalid id
  if(id < 0 || id >= numClients) return null;

  // invalid id
  int index = clients.indexOf(new BrokerClient("",id));
  if(index < 0) return null;

  BrokerClient client = clients.get(index);

  if(client.server != null)
    client.server.clients.remove(client);

  // Find server with best performance
  double maxValue = 0.0;
  int  maxIndex = 0;
  for(int i = 0; i < 5; i++)
  {
    servers[i].computeQoSMetric();
    if(servers[i].getQoSMetric() > maxValue)
    {
      maxValue = servers[i].getQoSMetric();
      maxIndex = i;
    }
  }

  client.server = servers[maxIndex];
  servers[maxIndex].clients.add(client);

  return servers[maxIndex].getUrl();
}
```

Provide QoS Information

```
@WebMethod(operationName = "getQoSInfo")
public synchronized QoSInfo getQoSInfo () {
  if(!initialized) init();

  for(int i = 0; i < servers.length; i++)
    servers[i].computeQoSMetric();

  return new QoSInfo (servers,clients);
}
```

Bibliography

[1] Ieee std. 1516 – standard for modeling and simulation high level architecture.

[2] Asn.1 - xml - fast infoset. `http://asn1.elibel.tm.fr/xml/finf.htm`, 2006.

[3] US Army. America's army: Special forces. `http://www.americasarmy.com`, 2008.

[4] K. Birman. The untrustworthy web services revolution. *Computer*, pages 98–100, February 2006.

[5] R. Brooks, J. Schwier, and C. Griffin. Behavior detection using confidence intervals of hidden markov models. 2008.

[6] D. Brutzman, M. Zyda, M. Pullen, and K. Morse. extensible modeling and simulation framework (xmsf): Challenges for web-based modeling and simulation. In *XMSF Workshop and Symposium*, 2002.

[7] S. Chandrasekaran, G. Silver, J. Miller, J. Cardoso, and A. Sheth. Web service technologies and their synergy with simulation. In *Proceedings of the 2002 Winter Simulation Conference*, pages 606–615, 2002.

[8] A. D'Ambrogio and D. Gianni. Using corba to enhance hla interoperability in distributed and web-based simulation. In *Proceedings of the 19^{th} International Symposium on Computer and Information Sciences (ISCIS'04)*, pages 696–705, 2004.

[9] J. Dingel, D. Garlan, and C. Damon. Bridging the hla: Problems and solutions. In *Proceedings of the 6^{th} International Workshop on Distributed Simulation and Real-Time Applications (DS-RT'02)*, 2002.

[10] R. Elfwing, U. Paulsson, and L. Lundberg. Performance of SOAP in web service environment compared to CORBA. In *Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC'02)*, 2002.

[11] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. Modeling stateful resources with web services. `http://www.ibm.com/developerworks/webservices/library/specification/ws-resource/ws-wsrfpaper.html`, 2004.

[12] P. Gustavson and T. Chase. Using xml and boms to rapidly compose simulations and simulation environments. In *Proceedings of the 2004 Winter Simulation Conference*, pages 1467–1475, 2004.

[13] F. Hassaine, N. Abdellaoui, A. Yavas, P. Hubbard, and A. Vallerand. Effectiveness of jsaf as an open architecture, open source synthetic environment in defense experimentation. In *Transforming Training and Experimentation through Modeling and Simulation*, pages 11–1–11–6, 2006.

[14] F. Kuipers, P. Van Mieghem, T. Korkmaz, and M. Krunz. An overview of constraint-based path selection algorithms for qos routing. *IEEE Communications Magazine*, 2002.

[15] F. Lanchester. *Aircraft in Warfare: The Dawn of the Fourth Arm*. D. Appleton and Co., New York, 1916.

[16] D. Menascé. Qos issues in web services. *IEEE Internet Computing*, 2002.

[17] Microsoft. .net binary format: Xml data structure. `http://msdn.microsoft.com/en-us/library/cc219210.aspx`, 2008.

[18] OneSAF. Onesaf public site. `http://www.onesaf.net`, 2008.

[19] T. O'Reilly. What is web 2.0: Design patterns and business models for the next generation of software. `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`, 2005.

[20] K. Pingali and P. Stodghill. A distributed system based on web services for computational science simulations. In *Proceedings of the 20^{th} ACM International Conference on Supercomputing (ICS'06)*, pages 297–306, 2006.

[21] P. Prasithsangaree, J. Manojlovich, J. Chen, and M. Lewis. Utsaf: A simulation bridge between onesaf and the unreal game engine. pages 1333–1338, 2003.

[22] R. Reddy and R. Garrett. Future technology challenges in distributed interactive simulation. In *Proceedings of the IEEE*, volume 83, pages 1188–1195, 1995.

[23] J. Schwier and M. Bennink. Verification of behavior recognition using java web services. 2008.

[24] R. Smith. Next generation technology for simulation and training. NATO Nations and Partners for Peace: Special Issue on Simulation and Training, 2007.

[25] R. Smith. The long history of gaming in military training. Simulation and Gaming, 40th anniversary issue, 2008.

[26] R. Smith. Web 2.0 and warfighter training. In *European Simulation Interoperability Workshop*, pages 1–8, 2008.

[27] LLC Tactical Language Training. Tactical language and culture training systems. `http://www.tacticallanguage.com/`, 2008.

[28] O. Topçu, M. Adak, and H. Oğuztüzün. A metamodel for federation architectures. *ACM Transactions on Modeling and Computer Simulation*, 18(3):10:1–10:29, 2008.

[29] USJFCOM. Joint semi-automated forces. `http://www.jfcom.mil/about/fact_jsaf.html`, 2008.

[30] D. Vaden and G. Miller. Modsaf fades away. `http://www.sisostds.org/webletter/siso/iss_75/art_367.htm`, 2001.

[31] W3C. Efficient xml interchange. `http://www.w3.org/XML/EXI/`, 2008.

[32] H. Wang and H. Zhang. Collaborative simulation environment based on hla and web service. In *Proceedings of the 10^{th} International Conference on Computer Supported Cooperative Work in Design (CSCWD'06)*, 2006.

[33] Wikipedia. Run-time infrastructure. http://en.wikipedia.org/wiki/Runtime_infrastructure, 2008.

[34] T. Yu and K. Lin. A broker-based framework for qos-aware web service composition.

[35] T. Yu, Y. Zhang, and K. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. In *ACM Transactions on the Web*, 2007.

[36] H. Zhang, H. Wang, and D. Chen. Integrating web services technology to hla-based multidisciplinary collaborative simulation system for complex product development. In *Proceedings of the 12^{th} International Conference on Computer Supported Cooperative Work in Computer Design (CSCWD'08)*, pages 420–426, 2008.