ABSTRACT

COMPUTER AND INFORMATION SCIENCE

THOMAS, JAMAR E. B.S. CLARK ATLANTA UNIVERSITY, 2005

ASYNCHRONOUS INSTANT MESSAGING USING SERVICE-ORIENTED ARCHITECTURES (AIMSOA)

Advisor: Professor Roy George, Ph.D.

Thesis dated June 2005

Instant messengers suffer from poor scalability, flexibility, security, and interoperability. This study attempts to solve these problems using the strengths of Service-Oriented Architectures. The key components to achieve these improvements include several Java related technologies such as JAX-RPC, JAXM, SOAP, WSDL, J2EE servlets and Enterprise Java Beans. SOAP provides a universal messaging protocol that heterogeneous parties can understand. JAX-RPC provides synchronous SOAP messaging, as well as a loosely coupled design that allows for a very flexible distributed architecture. JAXM provides asynchronous SOAP messaging. When used together, applications can implement robust instant messaging functionality. Registration, login, and other instant messaging configuration operations can be fulfilled through the use of JAX-RPC while JAXM can be used to fulfill requirements such as send and receive. Servlets and Enterprise Java Beans augment the benefits of Service-Oriented Architectures with the former being extremely scalable, portable, and modular. AIMSOA encapsulates these components to provide an instant messaging architecture solution that will augment the weaknesses of current instant messaging architectures by providing a solution for better scalability, flexibility, and interoperability.

ASYNCHRONOUS INSTANT MESSAGING USING SERVICE-ORIENTED ARCHITECTURES (AIMSOA)

A THESIS

SUBMITTED TO THE FACULTY OF CLARK ATLANTA UNIVERSITY IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

BY

JAMAR E. THOMAS

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

ATLANTA, GEORGIA

AUGUST 2005

ACKNOWLEDGEMENTS

First and foremost, I give all glory and honor to God. Secondly, I wish to thank my mother, Annette Weathington, father, Anthony Thomas, and brother, Malcolm Thomas for their support and unyielding love during my life and especially throughout my matriculation at Clark Atlanta University. I would also like to thank my fiancé and very best friend, Angela Strange, for all of her patience and endurance during this trying and very hectic time of my life. I extend a warm thanks to the PRISM-D and Honors program for being my second family during my time at CAU. Lastly, I would like to thank the entire Department of Computer Science, especially Dr. George, in helping in finalizing my requirements for this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AIM	AOL Instant Messenger
CORBA	Common Object Request Broker Architecture
DCOM	Microsoft Distributed Component Object Model
ebXML	Electronic Business using Extensible Markup Language
EJB	Enterprise Java Beans
IM	Instant Messenger
J2EE	Java 2 Platform Enterprise Edition
Java RMI	Remote Method Invocation
JAXM	Java API for XML Messaging
JAX-RPC	Java API for XML Remote Procedure Call
JMS	Java Messaging Service
LDAP	Lightweight Directory Access Protocol
OSCAR	Open System for Communication in Real-time
SAAJ	SOAP with Attachments API for Java
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration Protocol

WSDL	Web Service Definition Language
XML	Extensible Mark Up Language
XMPP	Extensible Messaging and Presence Protocol

CHAPTER ONE

INTRODUCTION

Many of today's software applications are developed and constructed on traditional architecture strategies to deliver stable services to the user. These architecture strategies range from monolithic to client-server to n-tier development systems. A monolithic architecture is characterized by having all parts of a software system running on the same machine. Client-server and n-tiered based systems have logical or physical separations within the software system. Each architecture suffers from its own set of drawbacks and disadvantages. In today's heterogeneous computing environment, it is necessary for different systems to communicate and even integrate, in which case, drawbacks multiply. This leads to some of the biggest problems Information Technology (IT) enterprises and organizations face: challenging and cost consuming application to application integration, data integration incompatibilities, and inflexible architectures that cannot be reconfigured quickly enough to meet changing needs and requirements. Recently, there has been a paradigm shift towards a new way of constructing applications called Service-Oriented Architectures (SOA) that serve to eliminate such problems. Instant messaging is an area that suffers from the inherent drawbacks of traditional architectures. The researcher is proposing an architecture, AIMSOA, that can be adapted by existing and to be created instant messaging (IM) applications to achieve functionality that is by default scalable, loosely coupled, interoperable, and flexible in nature.

Problem Statement

Progress in IM interoperability and extendibility is stagnant due to the weaknesses of the underlying architecture. IM applications have been slow to extend themselves to other environments such as email and cell phone systems. Although IM to IM interoperability is slowly taking form, it is years too late and has come at expensive costs. Ideally, users should be able to IM other users regardless of firewalls, vendors, or platforms. Unfortunately, to do so means modifying current architecture and design implementations. Interoperability, scalability, and flexibility are major problems in the IM industry. AIMSOA is inherently interoperable, and can integrate easily with other entities. IM environments are suffering from security vulnerabilities and subject their users to unwanted spam. AIMSOA's architecture and security infrastructure can, at the minimum, rival that of the current implementations. Instead of making incremental changes to existing architecture, AIMSOA can be used as a new base infrastructure or as a wrapper to the current infrastructures to achieve the desired improvements.

Background

The Instant Messenger (IM) market provides several alternatives to instant messaging with AOL leading the competition with both its AIM and ICQ commercial chatting applications. The majority of instant messaging solutions are developed around client/server architectures. In client/server systems, clients are logically or physically separated from the server system. Clients submit and retrieve data to and from the server across a network connection. Figure 1 illustrates a traditional client/server system and

Figure 2 depicts a client/server architecture when applied to instant messaging. Client/server architectures provide several advantages including encapsulation, code separation, and enhanced flexibility (McGovern et al. 2003). They also introduce several drawbacks when trying to communicate between the two entities. Indeed, when client/server entities are constrained within an enterprise or organization, operations can run smoothly: the likelihood of data formats, protocols, architectures, and languages all being the same is very high. But when, for instance, a client from an outside organization wants to communicate with the server, incompatibility and integration issues will have to be resolved: the client may function on a different architecture set, programming language, or data format than the server. In addition, the protocols for communication between client and server may be proprietary, further hindering integration of heterogeneous architectures. Multi-tiered or N-tiered architectures add additional client or server layers, and are considered extensions to the traditional client/server paradigm as depicted in Figure 3.

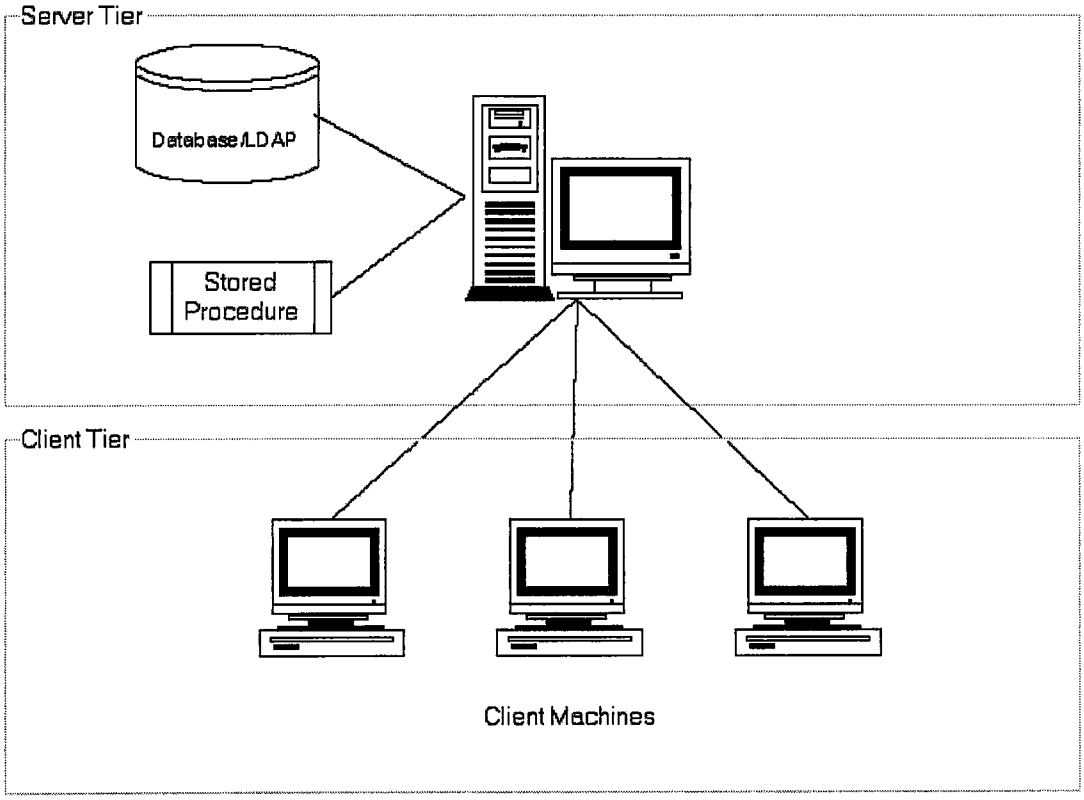

Figure 1: Traditional Client/Server Architecture

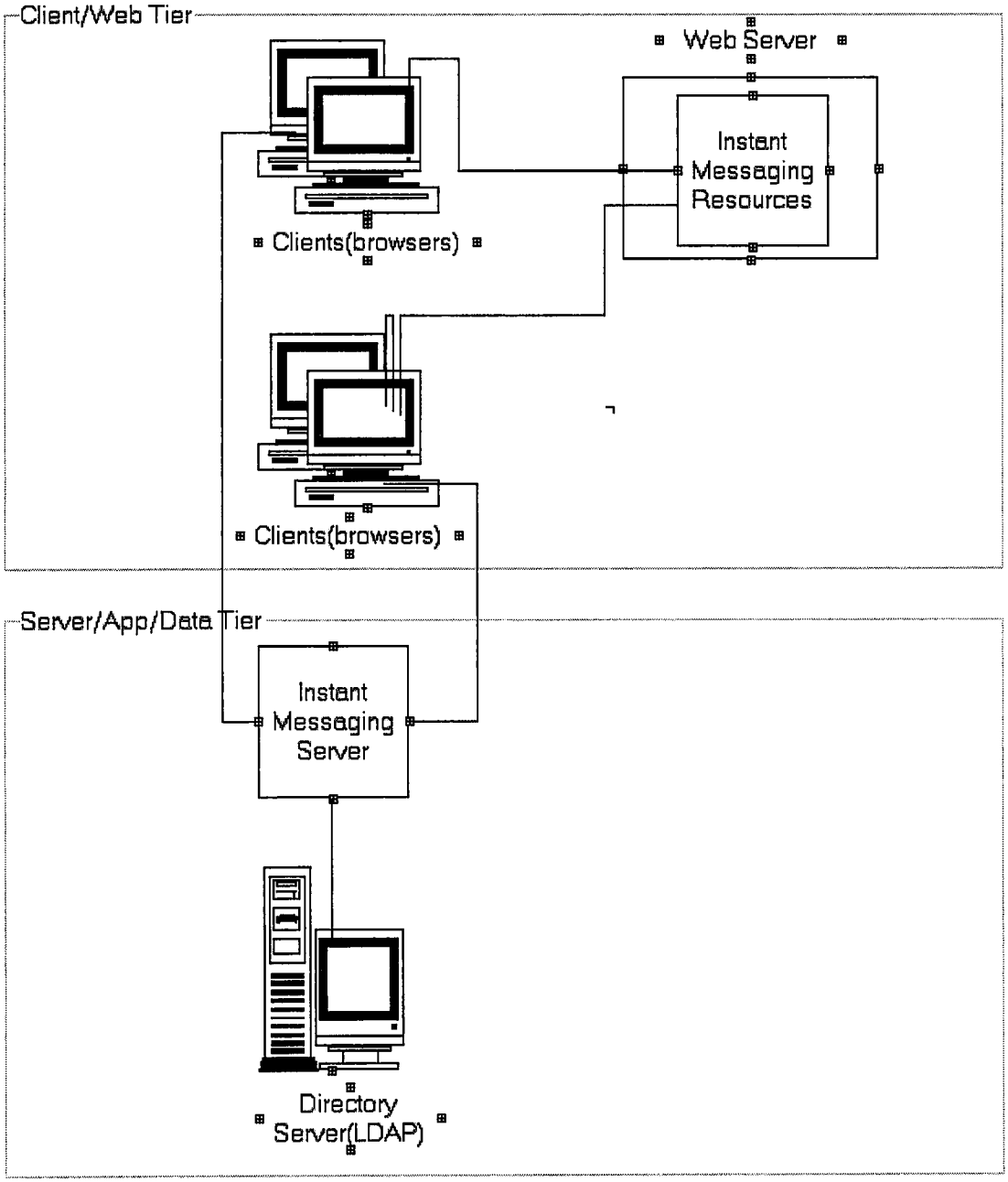

Figure 2: Traditional IM Application Architecture

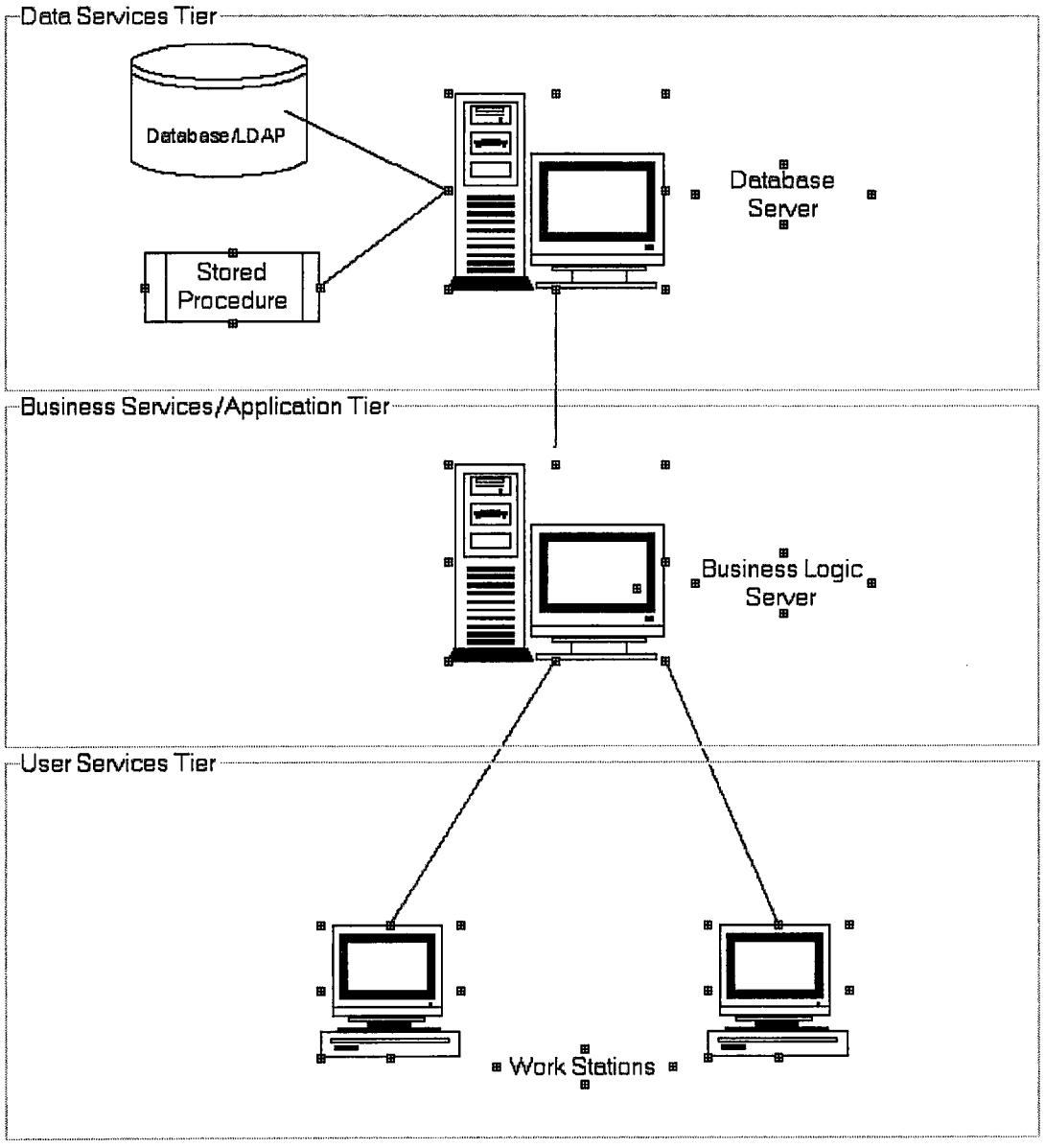

Figure 3: Multi-Tiered Architecture

The largest IM systems are proprietary, and therefore unusable for systems that may need to implement, extend or control the servers (Shigeoka 2002). Because of the limitations of the architecture, it is inherently difficult for two different IM vendors, small or large, to communicate with each other. In addition, IM messaging formats must be the

same. Two parties must agree on a predefined encoding scheme in order to communicate (McGovern et al. 2003). Encoding refers to how messages are serialized and sent over the wire. Recently, work has been done to bridge the incompatibility issues between systems, but because of the inflexible nature of the architectures on many IM systems, the task is difficult.

Significance of Study

I am proposing an instant messaging architecture that can augment the client/server or tiered constructs to deliver flexible software systems. The most widespread architectures are proprietary, and it has become seemingly more inadequate at the inability to chat with users on different IM systems. There has been a substantial amount of effort in the research communities to provide interoperable solutions, only to be limited by the weaknesses of the underlying architecture and design utilized by IM vendors. Additionally, as popularity has increased, IM vendors are finding it increasingly difficult in dealing with security threats. AIMSOA provides a solution to such problems with benefits that include increased flexibility, interoperability, and scalability. These improved IM systems will then be able to offer better services to users. For example, buddy lists and messages can be shared amongst IM vendors. Buddy lists can also be easily linked to contacts provided by email hosts, or even phonebook contacts offered by wireless companies. Instant messengers based on AIMSOA will be more likely accepted by enterprises concerned with security.

CHAPTER TWO

REVIEW OF EXISTING CONCEPTS

Organization of Review

This section breaks down the key concepts related to SOAs that can be applied to create an improved instant messaging architecture. First, a background on instant messaging and service-oriented architectures is discussed. Next, distributed computing design principles that are being sought as a solution are addressed. Instant messengers have been recently deployed to embedded devices like cell phones. AIMSOA's benefits in such an arena are briefly discussed, followed by a definition of key terms. Lastly, the shortcomings of current solutions are investigated.

IM Background

Instant messaging first garnered wide-scale attention and usage when AOL released 'I Seek You' (ICQ) in November of 1996. Instant messaging is used to send text messages electronically in real time. ICQ, like many of its peers, was developed to connect family, friends, and associates over the computer to promote a viable electronic social community. AOL Instant messenger (AIM), Yahoo! Instant Messenger, and MSN Messenger are the three biggest public IM vendors besides ICQ. AIM, the largest of the IM vendors, holds the largest share of the IM market with 30 million active users and 180 million registered members worldwide (Richard 2002). These instant messengers allow for contact lists storage and provide presence notification. Hereby, presence refers to the ability of users to be notified when other users are online. Instant messengers can be

divided into two categories: public and enterprise. Public instant messengers are usually geared for communication amongst friends and family without restriction. Enterprise messengers are usually located behind firewalls, and can only communicate with users within the organization. Although first intended for public commercial use, the popularity for IM usage amongst enterprises and organizations has increased tremendously over the years as a tool to communicate with co-workers. In a society where organizations are distributed across states and countries, companies are driving down the costs of long distance phone calls by communicating via instant messengers over the Web. In addition, IM messengers, unlike phone technology, can keep track of presence information. In the work environment, users can know whether their co-workers are at their desks at all times, a convenient feature that promotes better productivity and consequently better business.

Service-Oriented Architecture (SOA)

SOAs are the perfect solution to providing flexibility, interoperability, and scalability to IM applications. SOAs have been used in other areas to provide similar performance benefits. In general, any existing system that utilizes SOAs within an organization can be easily integrated because they provide well-defined interfaces that can be accessed using standard protocols and transports (Pasley 2005). Abstraction, simplicity, and loose coupling are all components of a flexible system. The interfaces make up what is known as the service tier in a service-oriented architecture. A service tier abstracts the details of the data sources, thus creating a very simple loosely coupled

environment to develop applications such as instant messengers. SOAs are often used to expose application functionality and to integrate systems with newer applications. SOAs are based on standards. The support of these standards gives enterprises the ability to map data from disparate systems, route messages, ensure services are delivered, and enforce security rules automatically by using XML.

In simple terms, SOAs are service providers. To be service-oriented, an architecture must implement the find, bind, and execute paradigm (McGovern et al. 2003). A SOA is composed of three roles: provider, consumer, and service broker. In order for a consumer to find the service, the provider must first register and describe this service in a registry or service broker. UDDI and ebXML are the most popular and standard registries for service discovery. Once the consumer is able to find the service, the next step is to bind to the formats and protocols associated with the service. Registries store WSDL files which describe the functional characteristics of a service: how the service is invoked and where the service is located. The find, bind, execute process can be seen in Figure 4. The J2EE or .NET platforms provide tools to process the given WSDL file to create proxies for the consumer. Once created, the proxies provide the necessary bindings for the consumer to interact with the service. The last step is to execute the service operations.

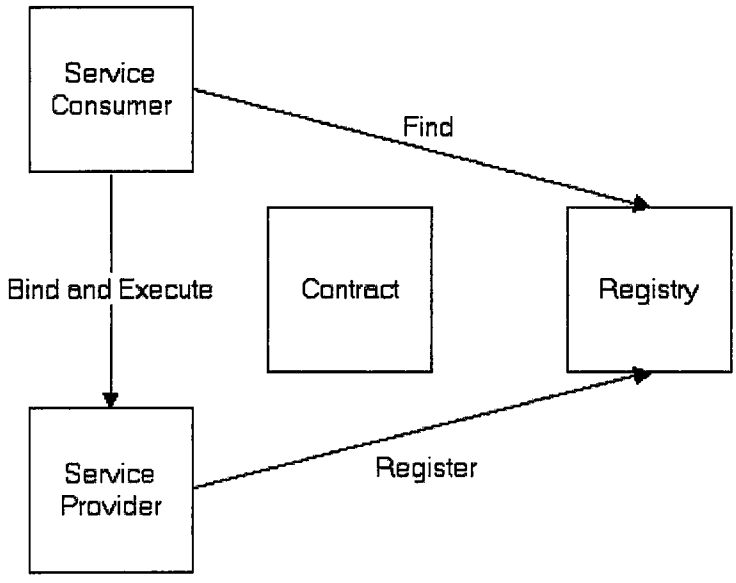

Figure 4: Find, Bind, and Execute Paradigm

SOAs are also extensions of the distributed computing model. Scalability, openness, concurrency, transparency, and heterogeneity are all characteristics of distributed systems, and likewise that for SOAs. Additionally, and more importantly, SOAs are modular; they separate implementation from its interface. Distributed interoperable messaging solutions such as Web services, CORBA, and DCOM are all examples of SOAs.

SOAs can augment existing client/server architectures by providing an additional service tier. Service tiers are characteristics of SOAs because they provide a standardized interfacing layer. The importance of a service layer can be described as follows: "Architecture without a service tier has a huge flaw- the lack of data hiding or abstraction. Without a service tier, application code must have intimate knowledge of the associated details, such as database schemas or low level software APIs" (Bruno 2005).

The service tier lies between the web/user tier and the application tier as described in Figure 2. Figure 5 shows an augmented traditional architecture utilizing a service tier. A service tier adds significant value to an application's architecture because it limits the coupling between components and promotes greater software reuse (Bruno 2005).

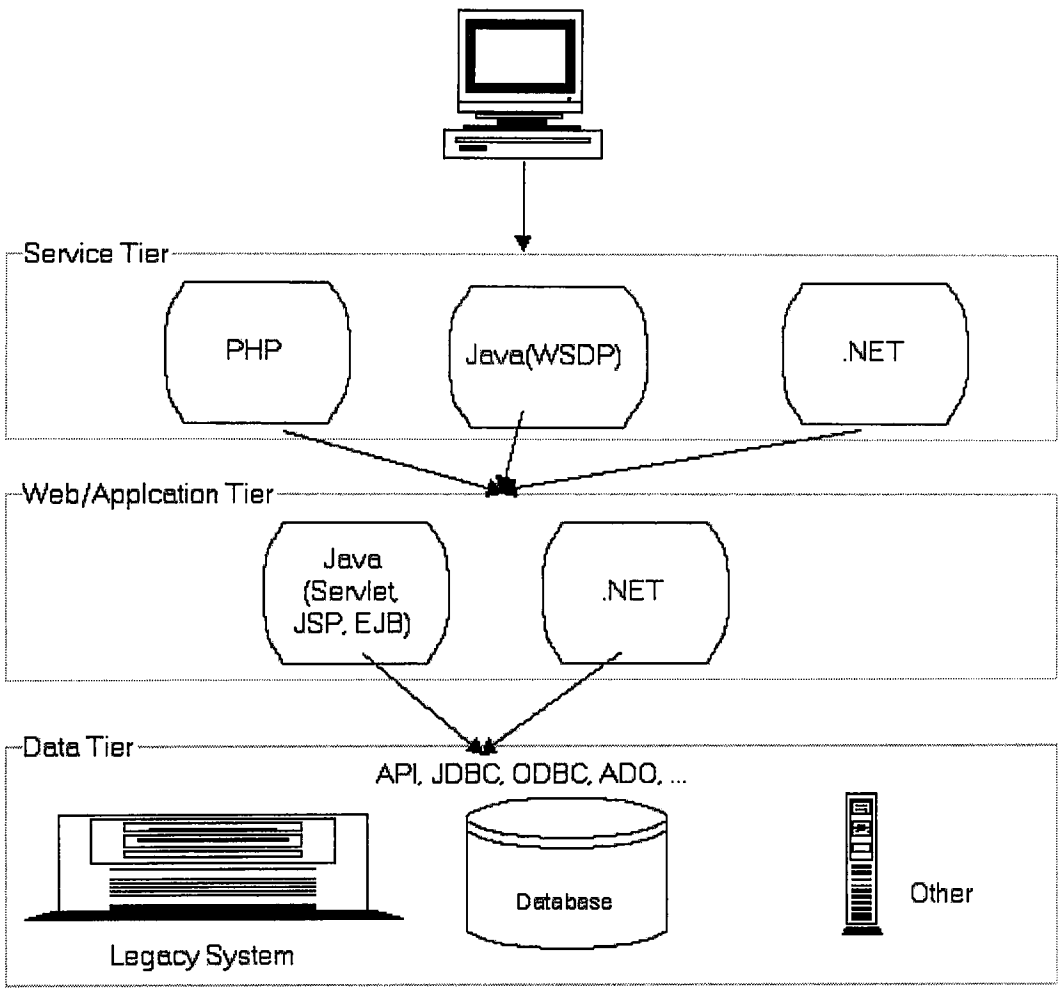

Figure 5: Traditional Client/Server Architecture augmented with a Service Layer

Design Principles of Distributed Computing

Interoperability

Interoperability is defined as the ability of systems using different platforms and languages to communicate with each other. A standard protocol and data format are needed to achieve interoperability. JAX-RPC and JAXM are considered interoperable because of the existence of defined Java type format mappings for SOAP, which is the standard protocol for SOAs. Currently, few applications can communicate across different networks because IM services and systems employ proprietary protocols (Woods, "Interoperability Coming," 2002). To come in correspondence with such proprietary protocols, companies such as Jabber and Trillion offer interoperability solutions with other IM vendors by backward engineering code. Jabber offers what are known as transports to transparently send and receive messages to users and track presence. Transports connect with the foreign message systems and act as a client or server on that system in order to relay the messages and presence updates between the two systems (Shigeoka 2002). There are several legality issues with such a process which keeps enterprise messaging solutions from migrating to those platforms. More importantly, large vendors such as AIM, Yahoo Messenger, and MSN Messenger actively change their protocols to keep 3rd party vendors off their networks.

Flexibility and Scalability

A flexible IM architecture promotes changeability and scalability of an

application. A scalable architecture will be able to handle unanticipated numbers of concurrent users, increased load on the hardware, and unsatisfactory availability. Flexible architectures should be able to scale into unexpected environments or loads seemingly. Additionally, a systems degree of flexibility can be directly related to its modifiability. Inflexible architectures require extensive amounts of maintenance and even downtime. As an example, Bantu, an enterprise instant messaging solution, once had to temporarily sever its link with Yahoo Messenger because the portal giant made changes to its IM network (Woods, "Bantu Restores," 2002). Because of the loosely coupled nature of SOAs, AIMSOA offers very few dependencies, making modifications easy to implement. The J2EE architecture provides plenty of flexibility to accommodate changes as the requirements for throughput, performance, and capacity change (Weaver and Mukhar 2003). Hubz is a Web-based instant messaging solution whose architecture is based on J2EE components. Because Hubz is built on enterprise-level Java Server technology and the company's own enterprise level chat-server technology, it is more secure, scalable, and reliable than many other messaging solutions on the market (Woods, "Web-based," 2002).

Over the years, AIM has experienced significant resistance in entering the enterprise messaging community because of its lack of support for organizations behind firewalls. This is because AIM was never designed to be used in corporate environments. Several modifications have been made to make it more enterprise friendly, but an architecture implementing a service tier can be extended beyond the boundaries of itself, allowing for a more flexible, scalable architecture (Brund 2005).

Portability

Portability refers to software that can be easily moved and implemented from one machine to another. Less portable applications require additional effort to function properly. Java, the language of the J2EE environment is well-known for its portability. In fact, Hubz is a Java-based web applet that allows visitors on the same website to see and interact with one another without using special programs, plug-ins or client-side software (Woods, "Web-based," 2002). Applets and servlets are all Java Web-based technologies that can be combined with Java SOA tools to generate an extremely portable environment that can go and function anywhere. The benefits of Web-based architectures have encouraged Jabber to develop a Web client for its popular IM servers. Web-based architectures allow for firewall friendly connections with no installed client, ideal for the enterprise market.

Instant Messaging and Cell Phones

Wireless vendors currently offer services that require information to be "pushed down" to phones such as weather updates, sports updates, stock quotes, and instant messengers. AIMSOA will allow phones to "pull" or consume services allowing for smarter, smaller applications. Because they are so small, embedded devices such as cell phones have high memory constraints, and restrict the amount of processing that can be made in memory. AIMSOA allows for major processing to be done on a server, while the results are delivered to or retrieved from the cell phone. The other significant

advantage AIMSOA provides is that the user interface deployed on the wireless cell phone utilizes the same Web service a normal, much heavier user interface would; the service deployed need not change.

Key Terms and Concepts

The following are terms and concepts that apply to SOA or IM technology. These terms are listed here as a reference to the AIMSOA architecture presented in the next section. This section concludes by providing an overall summery of the key terms and concepts and how they are related to each other and to AIMSOA.

Web Services

A Web service is a service offered by an application to be consumed by other applications. Therefore, Web services are often used in a B2B (Business to Business) environment. Web services serve as an Application Service Provider (ASP) for many businesses, enabling one business to invoke a service of another business, thus allowing the former business to forgo the cost and time of developing the service itself. These services are programs that perform some logic and generate a result. A common simple example of a typical Web service is a stock quote service. Instead of implementing the service itself, a business can just request to use the service and send the appropriate inputs to the service. In this case, a stock name would be sent, and a stock price would be returned.

The key to this technology is that all information is communicated and transferred in a truly independent manner. Additionally, Web services function around the use of standards. This is especially important in the real world, when many enterprises operate with different information systems. Web services communicate using a platform independent protocol known as Simple Object Access Protocol (SOAP), based on XML (Extensible Markup Language). XML is a language neutral document that enables any application to communicate with another.

Web services improve on the shortcomings of technologies like RPC, DCOM and CORBA by combining a standards based approach to distributed computing with the availability and accessibility of the Internet (Fischer, 2002). Web services are implementations of SOAs.

Security

- Secure Sockets Layer (SSL): SSL is a technology that allows Web browsers and Web servers to communicate over secure connections because data is encrypted and decrypted on delivery. SSL uses server certificates and keys.
- Security Assertions Markup Language (SAML): SAML was first developed by the Oasis Security Services Technical Committee. It is an XML-based framework for security information exchange across heterogeneous business entities. SAML is primarily used for authentication and authorization.
- XML Digital Signatures: XML Digital Signatures specify syntax and processing standards for attaching digital signatures to XML documents.

- Extensible Access Control Markup Language (XACML): XACML provides a standard for security access control using XML to state authorization, validation, and revocation rules over a connection
- Key Management Specification (XKMS): XKMS is a standard that details protocols for registration and distribution of public keys, so that keys can be integrated with XML digital signatures and encryption (McGovern et al. 2003).
- XML encryption: The following Web service specifications have been developed to provide SOAs with a robust security infrastructure:
    - WS-Security: Describes how to attach security tokens (certificates and Kerberos tickets),signatures, and encryption headers to SOAP messages
    - WS-Policy: Describes the features and limitations of the security policies on intermediaries and endpoints
    - WS-Trust: Describes a framework for secure interoperation of trust models between Web services
    - WS-Privacy: Describes how both Web services and their requestors specify subject privacy preferences and organizational privacy practice statements
    - WS-Secure Conversation: Describes how to authenticate and manage message exchange between parties
    - WS-Federation: Describes how to manage and broker trust relationships in a heterogeneous federated environment
    - WS-Authorization: Describes how to manage authorization data and policies.

SOA Clients

Unlike traditional architectures, SOAs provide for extremely loosely coupled flexible client solutions. This is made possible through interfaces and WSDL files. WSDL files contain information such as the location of the Web service and the operations and its functional signatures. Because clients interact with the interface only, changes to application logic will not affect the client as long as the return types remain the same. In the event that the interface does change, or the location of the service changes, SOAs provide what is known as early or late binding to keep the client from breaking. Bindings govern the links between the service and the client for interaction.

Early binding processes the WSDL file at compile time to generate client side proxies to interact with the service. Clients interact with the stubs, and the stubs handle transferring the data to the correct endpoint and operation on the server. Additionally, Serializers and Deserializers can be created to encode SOAP data types into types recognized by the client. Early bindings, also known as client side stubs, can function in different forms: static compile time binding, static deploy time binding, static runtime binding. Stubs are characterized by the fact that the service's operations and functional signatures are always known before runtime, but the location may or may not be known before runtime.

Late binding processes the WSDL file at runtime on the fly. Although this method may have slower performance than an early binding scheme, changes to the consumed Web service will have less impact on the consumer (McGovern et al. 2003). This is a truly flexible architecture. For normal cases, early bindings can be utilized

because of its efficiency, and when for instance, the service changes, the client can fall to a late binding technique ad hoc, thus avoiding crashes. Late bindings can be characterized into two types: dynamic binding and dynamic binding with known location. In both types, the functional signature is not known until runtime.

Profiles

A profile is a protocol that extends what is supported by SOAP 1.1 and SOAP with Attachments standards. It provides additional specifics about the message that SOAP does not. Providers rely on profiles because SOAP messages do not provide enough information for delivery and store and forward operations. Clients, therefore, may need to augment their SOAP message with a profile. EbXML is the most common profile used today and is depicted in Figure 6.

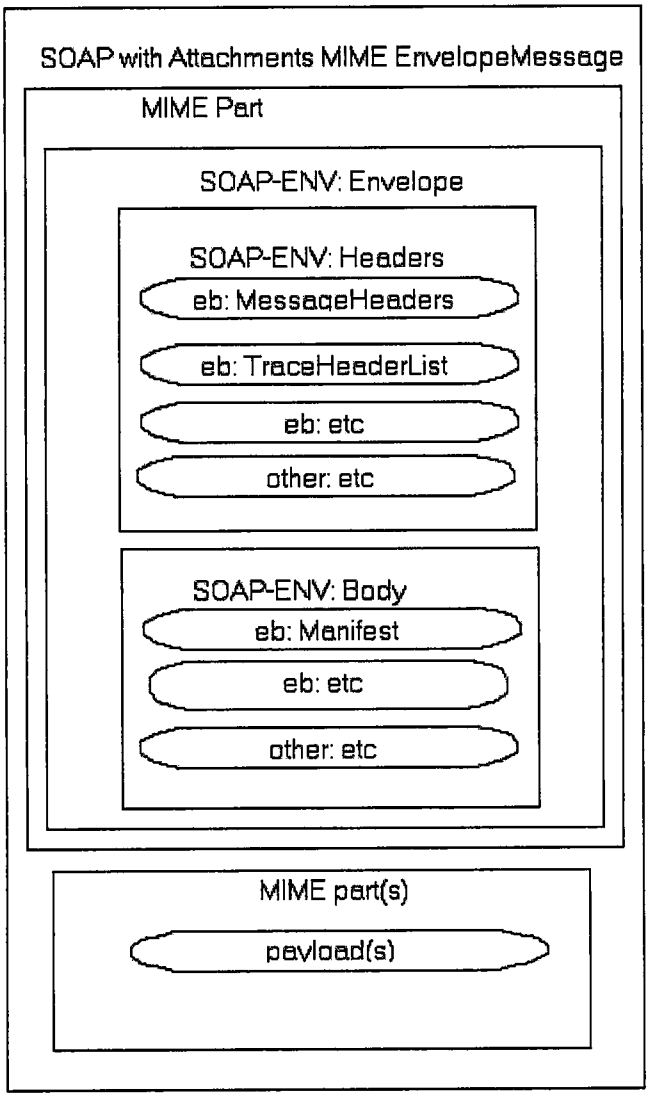

Figure 6: ebXML profile

Simple Object Access Protocol (SOAP)

Soap is an XML based protocol for exchanging messages over a distributed environment. Messages are made up of a SOAP envelope, SOAP header, and a SOAP body. SOAP is the standard protocol for communication among Web services. It specifies, among other things, what is required and optional in a SOAP message and how

data can be encoded and transmitted. It defines standards for messaging and data type mappings. Because SOAP is based on XML, it is very extensible. A graphical description of a SOAP message can be seen in Figure 7.

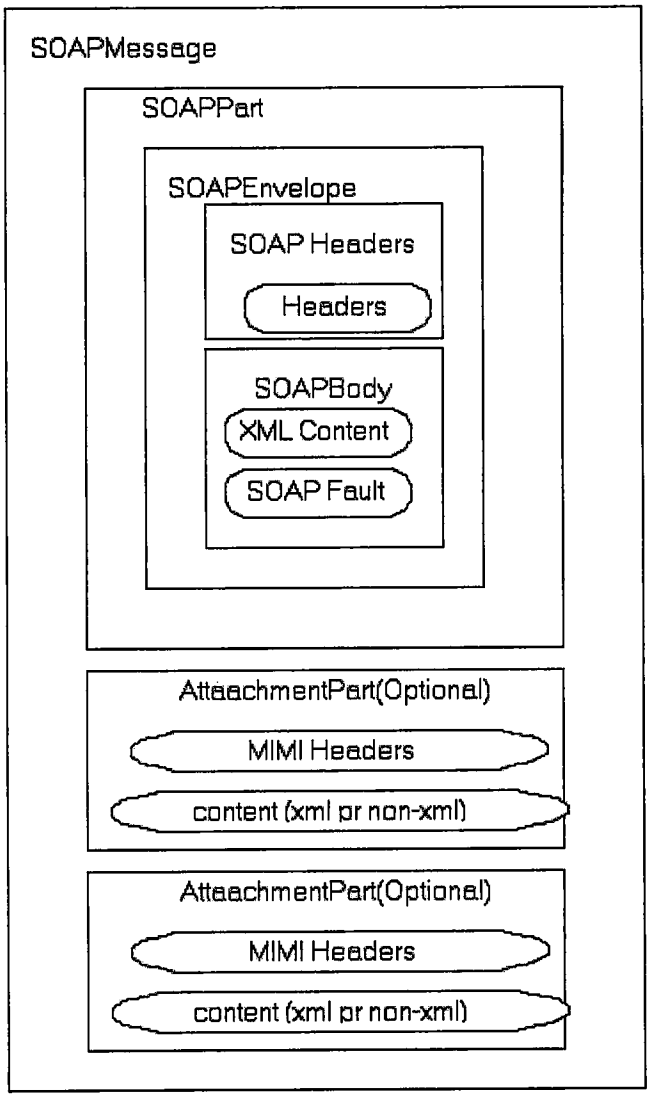

Figure 7: SOAP Message Structure

Registries

Registries serve as a look-up service for consumers to look up business services. Consumers can look up services by an address or identifier, or they can look up services by category. Registries maintain taxonomies to categorize services. Information stored in a registry for s single service includes a business entity, service, specification pointers, and service types. Generally, look-ups to registries are performed by first time parties or clients unfamiliar with the service. If the WSDL document changes or the service location changes, a future look up to the registry may be needed to obtain the out of date information.

DCOM, CORBA, Java RMI

- Common Object Request Broker Architecture (CORBA)
- Microsoft Distributed Component Object Model(DCOM)
- Java Remote Method Invocation (Java RMI)

These technologies are built from a client/server foundation focused on creating a standard technology platform on which to accomplish true application to application communication by providing access to remote methods (Fischer 2002).

Java API for XML based RPC (JAX-RPC)

JAX-RPC is a Java API to expose remote procedure calls using XML as the data format. A Web service based on RPC (Remote Procedure Call) is a set of procedures called by a remote client over the Internet. A Web service on a server implements these

procedures, as well as publishes information that describes the methods that are available in a repository for clients to look up. JAX-RPC hides the complicated infrastructure of transferring data from the developer. It handles the marshalling, unmarshalling, and transmission details so that the developer only has to worry about making method calls. J2EE vendors provide the JAX-RPC functionality to automatically convert Java primitives such as *int* and objects such as *String* into XML data types. This is a very powerful feature of J2EE because developers do not have to concern themselves with such mappings. Because data exchanged to and from Web service operations are of SOAP protocol, JAX-RPC is interoperable. Some important characteristics of JAX-RPC include:

- Synchronous: JAX-RPC is synchronous in nature, performing request blocking calls until a response is received. Although JAX-RPC has send-and-forget capability, it is not able to receive responses later in time, nor is it capable of receiving incoming messages from the server.
- Method oriented: JAX-RPC is method oriented, meaning, the primary means of communication is through a call to a method, and response from its output.
- Point to point: JAX-RPC requests are directed to endpoints, so a service needs to provide an endpoint to the repository
- XML: The data format exchanged is a standard format that any participating party can understand.

Message Oriented Middleware (MOM)

MOM is a piece of software that sits between communicating parties and provides the infrastructure responsible for handling disparate dependencies between them, such as operating systems, hardware, and communication protocols (McGovern et al. 2003).

Java API for XML Messaging (JAXM)

JAXM provides the capability for users to send and receive SOAP messages asynchronously. JAXM interacts with a provider that accepts messages on behalf of intended recipients (McGovern et al. 2003). A profile must be used for asynchronous messaging. JAXM is a type of MOM that is transport independent, meaning it can run over HTTP and other various types of transport protocols.

Java Messaging Service (JMS)

JMS provides Java applications with a standard interface to create, deliver, and retrieve distributed messages. JMS is a type of MOM and is included in the J2EE platform.

JAXM Provider

A provider is an implementation of the JAXM APIs. It layers over existing applications.

ebXML (Electronic Business using Extensible Markup Language)

The ebXML initiative developed by OASIS (Organization for the Advancement of Structured Information Standards) is a set of specifications for message formatting, business processes, and registry services (McGovern et al. 2003). EbXML profiles provide several additional fields to SOAP required for asynchronous usage including ConversationID, MessageID, TimeStamp, Service, and Action.

Java 2 Platform Enterprise Edition (J2EE)

The J2EE is a specification developed by Sun Microsystems and the Java Community Process for an application server to implement. Any application server that fully implements the specification is considered J2EE compliant. This is important in the enterprise world for portability reasons; applications created on one server should be able to run on any compliant server. Examples of such servers include IBM Websphere, JBOSS, Sun Java, BEA, Redhat, and Borland Application Servers.

The J2EE platform provides an infrastructure for the development and deployment of server applications. The infrastructure provides services that are needed for distributed architectures or server related applications commonly used in the enterprise level of companies. Such services include management, resource, transaction, mail, messaging, deployment, and security services. These services facilitate the developer in creating software, because the developer can spend the majority of time writing business logic that directory adds value to the company or organization. The J2EE platform also provides a framework for the development of Web services and

layers on top of the Java SDK which provides an API for Java Programming. A graphical description of the components can be seen in Figure 8.

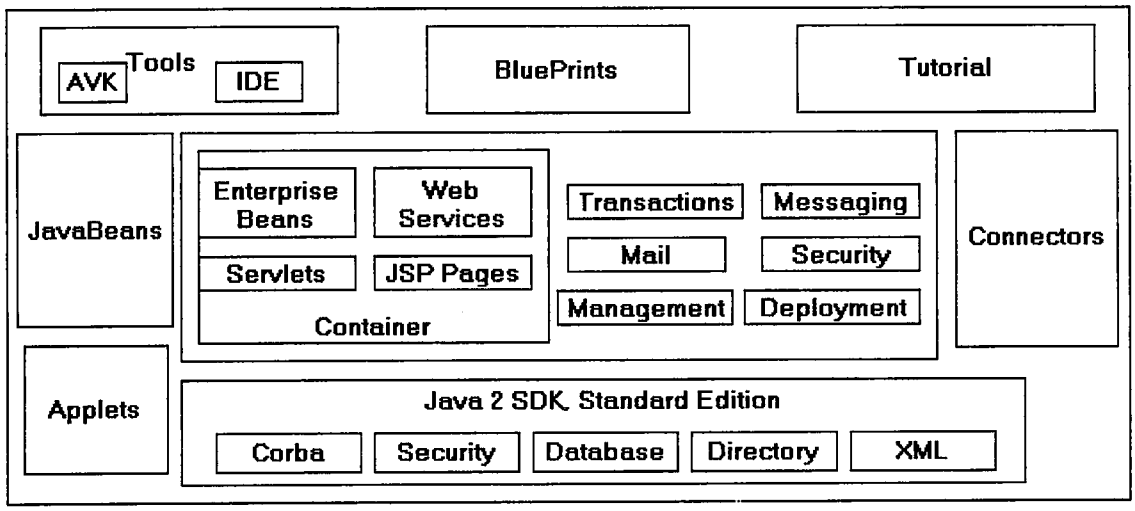

Figure 8:J2EE Platform

The J2EE architecure provides flexibility to accommodate changes as the requirements for throughput, capacity, and performance change (Weaver and Mukhar 2003). Much of this is due to its multi-tiered approach and its robust scaleble services: clustering, connection pooling, and failover. Because J2EE is based on the the Java programming model, it benefits from platform independence. Additionaly, J2EE functions as a specification as well. There are a muliitude of vendors who offer J2EE application products, but each product is governed by the specfication, ensuring compliemce amongst vendors.

Container

This layer serves as an interface to the application server services. Two important containers include Java servlet and Enterprise Java Beans (EJB). A servlet container provides services for Web development like session management, communication, and Web server integration. A common servlet container for Web services is Tomcat developed by Apache. EJB containers provide services such as transaction management and threading.

Servlets

Servlets are similar to CGI applications, but without its performance limitations. CGI applications are limited by scalability because each new request starts a new process. Servlets create new threads for each request, improving scalability functionality. In simple terms, they are server applications that can generate dynamic content to Web clients. Servlet technology is used to extend the capabilities of a Web server. Besides being platform independent, they are server independent as well (unlike Netscape server API and Apache models). Servlets have access to all Java APIs. Servlets process requests from clients and return responses.

Enterprise JavaBeans (EJB)

EJBs are often used in a J2EE environment for developing business logic because EJB containers provide services for client communication, session state management, transaction management, database connection management, user authentication, role-

based authorization, asynchronous messaging, and application server administration. They are server based components used for building data access.

Current Protocols used for IM

- Session Initiation Protocol (SIP): SIP is a flexible text based protocol over HTTP and MIME that can be used for many different media types, particularly voice over IP.
- SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE): SIP adds presence and instant messaging capabilities to the SIP protocol.
- Extensible Messaging and Presence Protocol (XMPP): XMPP is a set of streaming XML protocols and technologies that enable entities on the Internet to exchange XML messages and presence in real time. Jabber provides what are known as gateways or transports to legacy IM services. XMPP was developed by the Jabber organization.
- Open System for Communication in Real-time (OSCAR): Oscar was developed by AOL for AIM and is used for core IM functions.

Limitations of Current IM Architectures and Solutions

Security

Security holes plague the majority of instant messengers out on the market. In

October 2003, Symantec has found that 19 of the top 50 virus threats the first half of the year were related to IM and peer-to-peer(P2P) technology, resulting in a 400 percent increase from the previous year (Saunders, "IM Viruses," 2003). With the exception of Jabber's decentralized client server architecture, the majority of instant messengers route all communications through a central server (AIM, Yahoo Messenger, MSN Messenger). Some instant messengers such as the PP Messenger by Code Generation are P2P. P2P connections allow connections through a variety of ports that may not be safeguarded by firewalls, allowing entry points for viruses and hackers. The second major security vulnerability is in the fact that conversations between public IM networks (AIM, MSN Messenger, Yahoo Messenger, ICQ), appear in clear text, making it easy for hackers to listen in on chats (Woods, "Impasse Rolls," 2002). Many IM products transmit unencrypted data outside of the firewall, making them an easy target for interception on a network. The minimal security associated with P2P and IM invites malicious code propagation because of the absence of a server responsible for routing and filtering of request. Additionally, client machines are not equipped with robust enterprise firewalls to block necessary security threats.

Several solutions have been created to handle the aforementioned security issues. IMpasse, created in 2002, is a security platform solution that encrypts and decrypts text messages on the fly. Its major disadvantage lies in its portability and interoperability; to work, users on both ends must have Impasse installed on their computers. Trillion has also developed measures to encrypt communications to vendors such as AIM or ICQ. But just like Impasse, both communicating parties must use Trillion in order to operate.

Jabber and other vendors have created Web-based clients, unlike the traditional client of AIM and Yahoo. "Most companies are clamping down and not even letting other non-Web traffic in and out (of their systems). The fact that it is Web-based makes it easily compliant with corporate security guidelines" (Woods, "Jabber goes," 2001).

Companies are also utilizing enterprise-grade IM solutions to circumvent firewall issues. Companies are paying extra costs for copies of current IM solutions to be implemented within the enterprise. In this scenario, an IM server exists within the company's firewall. Messages are transferred within the bounds of the firewall, keeping them from the outside world. Anything that needs to go outside the organization requires proxy and logging actions to maintain the integrity of the system.

Security holes in current software designs are hampering the interoperability effort between vendors. The "unfortunate truism" (Svoboda 2002) of all security systems is that the level of security of the system as a whole is the same as the level of the security of its weakest part (Svoboda 2002). Although smaller IM vendors like Jabber have achieved a measure of interoperability with larger vendors like AIM, the larger vendors actively block such advances because interoperability may bring more opportunity for malicious code and hackers within their own networks.

SOAs are perfect solutions to the current IM problems because all communication is done through HTTP, alleviating any firewall restrictions or problems that can occur. Additionally, SOAs have developed over time a very comprehensive security infrastructure to combat the on-going prevalence of viruses and worms that plague instant messaging software. Unfortunately, the main problem with using most common security

architectures with SOAs is that the security infrastructure is distributed and these architectures usually require that key security features and algorithms are implemented by all parts of the system (Svoboda 2002). Providing a common security standard for diverse applications and systems is nearly an impossible feat.

SOAs provide Single Sign-On (SSO) capabilities that can be utilized to bypass such problems. SSO packages all the security requirements into a SSO Web service, releasing such obligations from other pats of the system. The SSO service then becomes the single authentication point for the software. The SSO server or Web service, acts as the wrapper around the existing security infrastructure that exports various security features like authentication and authorization (Svoboda 2002). IM vendors can use this as a common security base for AIMSOA implementations. Som Sengupta and Venkatakrishnan Padmanabhan present a SOA for enabling centralized authentication SSO implementation that can be used in correspondence with AIMSOA to enable the robust set of SOA security specifications and tools (2004). SAML is the language used to provide single sign on capability in their implementation.

Online Messenger Service (OMS)

OMS is a new IM application that utilizes the SOA architecture to provide its IM operations. Although it is able to achieve true interoperability by exchanging SOAP messages, it is limited by its synchronous messaging. Clients must poll the server in order to retrieve one way messages from other clients. AIMSOA augments this behavior by providing an architecture for asynchronous capability.

Current Client Architecture Design

Oftentimes, even within a layered approach, clients become tightly coupled with business logic and data processing. Application layers that present data to users can be intermingled with the layers of the application that act on the database (McGovern et al. 2003). The main drawback of this type of architecture is in its inability to be flexible enough to cope with change. Updates to servers can affect client behavior and cause problems in a tightly coupled environment. In an effort to thwart spam, Yahoo! once made upgrades to its highly popular Yahoo Messenger service, and as an unintended result, rendered several permitted third party clients unusable (Saunders, "Yahoo! Closes Door," 2003)[1]. Problems ranged from incorrect passwords to all out crashes on the client.

SOAs allow for applications to be tailored to an organization's business need. The current solution amongst the three major IM vendors is to package application logic into a single software unit to be delivered to enterprise and commercial users. Companies are choosing the on-demand model instead of packaged software (Knorr 2005).

On Demand computing can be described as providing resources and services to organizations as needed. Organizations have different needs, so the idea of creating a single piece of software to serve the requirements of many different groups is problematic. Some companies may prefer a simple, easy to use client over a clunky

[1] As of June 2004, Yahoo no longer permits these 3rd party clients to access to their IM network

complex software package that barely does what is desired (Woods, "MSN Messenger," 2002). A user or organization may not need all the features of an instant messenger, but another user may require IM capabilities that far exceed the common case. For example, users in Europe tend to use IM applications to send SMS messages to cell phones, and messages as mail to email servers. "SMS has yet to make a serious market impact in the United States. AIM users, on the other hand, tend to just message between other computer users and don't seek to turn their AIM clients into mail or SMS clients" (Richard 2002). Healthy enterprises need to develop their own unique applications, and any modern IT infrastructure needs to be fully integrated in a manner that can't be achieved with the current solutions of today (Knorr 2005).

Using AIMSOA, users are no longer tied to client graphical user interfaces so prevalent in many of today's packaged instant messaging software. Instead of being tied to a single user interface, enterprises can purchase the software as a service, and use custom user interfaces that may better fit the style or need of the organization.

Distributed Messaging Tool Limitations

CORBA, Java RMI, DCOM

These distributed communication standards suffer from several drawbacks that keep them from being viable solutions for IM communication. Interoperability is the key issue here. Java RMI is only compatible with other Java clients. CORBA is not firewall friendly because it uses the Internet Inter-Orb Protocol (IIOP) and port which is not

accepted by all firewalls. DCOM possesses functionality that is only operable with Windows platforms. In order to achieve an open, scalable, interoperable application, these solutions are unsuitable choices for deployment.

JAXM

Although developed to be included in the Web services toolkit for J2EE, JAXM was unfortunately left out of the specification. Therefore, JAXM must be downloaded and configured separately to be used with J2EE compliant application servers. Additionally, because many application servers are not required to support JAXM, the portability of applications utilizing JAXM may suffer. Lastly, unlike JAX-RPC, there is no support for WSDL file automatic generation.

JMS

JMS was designed as a common messaging API to be layered over existing enterprise message-oriented middleware applications, and not for Internet SOAP messaging. JMS was designed to be used within an enterprise, and not across enterprises using HTTP. Additionally, JMS clients are only interoperable with other clients over the same messaging system. Some vendors have tried circumvent these problems by wrapping SOAP messages into JMS text messages and passing them over HTTP. Unfortunately, the message format ends up becoming vendor proprietary. Even more importantly, SOAP messages are intended to function with attachments. By wrapping

SOAP messages into JMS text messages, there becomes no way to create and manipulate SOAP message envelopes that have attachments.

JAX-RPC

Although JAX-RPC Web services are the most common services utilized in many organizations, they lack asynchronous capability. JAX-RPC based messaging can only function in a synchronous based manner, meaning, if a process on the server takes a long time to compute, then the calling client is forced to wait until the process is finished.

Summary of Existing Concepts

AIMSOA is classified as a Web service. The security specifications and technology can be used to provide AIMSOA with a secure instant messaging environment that exceeds that of current architectures. DCOM, CORBA, and Java RMI are tools considered for communication in AIMSOA, but due to unacceptable shortcomings, were rejected as viable solutions. JAX-RPC and JAXM are SOA distributed messaging solutions because of their ability to transfer XML like SOAP messages. SOA clients can locate services in a registry for interaction with the service. In order for new SOA clients to perform JAXM based messaging, a reference to an ebXML registry must be done to be able to conduct future communications with a service. Additionally, in order for the client and service to communicate, they have to agree on a common structure and the kind of information being transferred; the JAXM profile provides such information. JAXM, like JMS, is an implementation of the MOM

concept, which means that it interacts with a provider that handles the forwarding and persisting of messages. Since JMS was not designed to operate outside of an enterprise, JAXM is a better solution to provide interoperable messaging across environments. The J2EE architecture offers implementations for the SOA standards and tools. More importantly, it offers tools such as servlets, EJBs, and containers that can be used to deliver and deploy powerful IM applications. AIMSOA looks to provide an alternative solution to the current protocols used by current IM architectures including SIMPLE, SIP, XMPP, and OSCAR.

CHAPTER THREE

DESIGN

Architecture

AIMSOA is a modular architecture, and can be seen in Figure 9. The architecture is broken into four tiers: user, service, application/web, and data access. The middle layer encompasses both the service tier and the application tier. The IM client and the IM registration client fall into the presentation layer. JAX-RPC and JAXM provide the services layer. The servlets and EJBs perform the business logic or server processing of the architecture. The database makes up the data access layer. A combination of JAX-RPC and JAXM communication APIs have been chosen for the AIMSOA messaging solution.

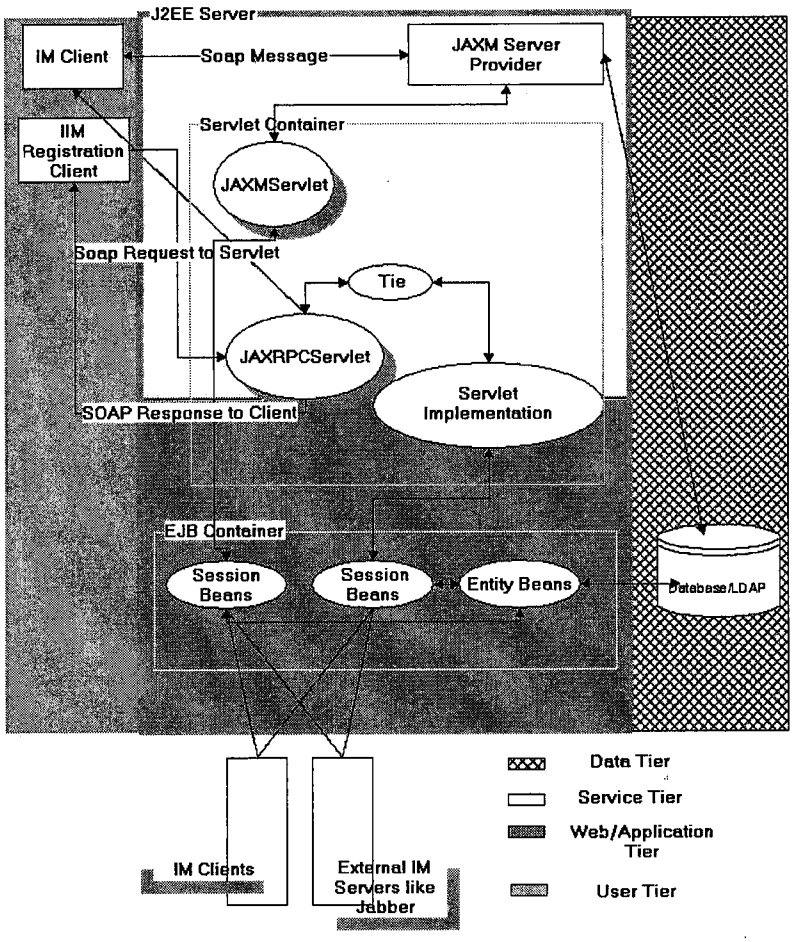

Figure 9: AIMSOA Architecture

Presentation Layer: Web clients

Because the majority of processing will be done on the server, software Web clients can be characterized as thin clients. Clients can be coded in any language as long as they can read XML messages and can communicate with a provider that functions over HTTP. The J2EE platform provides a tool named *wscompile* that clients can use to bind to the services offered by AIMSOA. *Wscompile* can be used on the client side and the

server side passing *client/server* flags respectively as input to the command. Other platforms have their own tools to bind languages such as C# and C to the service. To invoke methods exposed by RPC based Web services, the location of the service, protocol, and the service's methods and signatures must be understood beforehand. An application that is deployed based on the AIMSOA architecture must have a WSDL file that can be located from an ebXML registry or URL. The WSDL file contains the protocol to be used (HTTP) and the methods and method signatures of the IM service. *wscompile* is responsible for importing the WSDL file and generating specific Serializer, Deserializer, and stub classes for the client. The stubs, in conjunction with other *wscompile* generated classes, are used to handle any conversions needed such as data type mappings to communicate with the service.

In the architecture diagram, two clients are depicted: one a basic IM client and the other a registration client. With the exception of registration, the basic IM client can implement all necessary IM operations. The registration client is used specifically for registration, thus only interacting with the JAX-RPC servlet. A client developer can choose to merge the functionality into a single interface. There are no performance disadvantages, just a matter of style and abstraction.

Middle Layer: Provider

Providers act as proxies to forward messages to other users. The JAXM provider can be configured to forward messages to endpoints, either server endpoints or other provider endpoints. If an external client is deployed on a platform other than J2EE, any

provider that supports HTTP and SOAP can be utilized. The JAXM Provider can be deployed within a firewall if specific security requirements are demanded. The JAXM provider receives and delivers SOAP messages when available. If a particular endpoint is not available to receive, messages can be stored in a database and forwarded at a later time, thus creating persistence capability in a non persistence (HTTP) environment.

Middle/Application Layer: Servlet Container

The main focus of the container is to accept requests from clients. The majority of processing will be done by the EJB container. At times it may be necessary for a servlet to perform processing, which can be easily created and invoked from the container.

Middle/Service Layer: JAXM Servlet

The JAXM servlet's main function is to provide asynchronous capability. Actions such as sending and receiving messages should go through the JAXM servlet for delivery. For performance reasons, JAX-RPC should only be used for IM operations that do not require asynchronous functionality. This is because JAX-RPC is call oriented and JAXM is document oriented. JAX-RPC sends SOAP messages that contain only the information necessary for a methods execution. JAXM delivers full blown XML documents to and from the service. JAXM messages are larger and require more bandwidth so JAX-RPC should be used over JAXM whenever possible.

JAXM servlets must implement the *javax.xml.messaging.OnewayListener* interface to provide capabilities for asynchronous communication. The *OnewayListener* interface allows the servlet to listen for incoming messages. Messages are funneled into a method called *onMessage()* which accepts SOAP messages as a parameter. The local JAXM provider forwards messages to this servlet, which in turn can parse the message, and send to a session bean for processing. Headers in ebXML contain *To* and *From* fields that can be used to identify the sender and receiver. Additionally, the payload in the MIME Part can be used to store the actual message being sent. SAAJ can be used to easily parse these fields. Once parsed, a session bean can be used to send the message to the target recipient. The target recipient can be external clients like Jabber or other internal clients running on the server. Because JAXM servlets inherit all functionality of traditional servlets, session information can be stored and retrieved via the *HttpSession* object to retain certain state variables. Because the container controls all servlets, JAX-RPC servlets have access to the same variables, allowing for easy communication between the two.

Middle/Service Layer: JAX-RPC Servlet

The JAX-RPC servlet is used for IM operations that do not require asynchronous functionality. Operations such as registration, login, logout, add users, and remove users can function in a synchronous environment. Developers do not code the JAX-RPC servlet. It is the servlet container's job to create and deploy the servlet. Developers code an interface and implementation class which gets invoked by the servlet. In order to have

access to the servlets behavior, the implementation class must implement the *javax.xml.rpc.server.ServiceLifeCycle* interface. This allows the implementation to have access to objects such as *HttpSession* which maintains state and can be shared with JAXM. JAX-RPC hides all SOAP related information from developers. The SOAP message is extracted by the JAX-RPC servlet. The servlet can then invoke the appropriate method call exposed by the service (from the interface) passing it the extracted parameters. Unlike JAXM, the developer never comes in contact with the SOAP message. Clients indirectly interact with the implementation class through representatives called ties. Ties are also created by the *wscompile* tool used at the server level by passing a *server* flag as input to the command.

Middle/Application Layer: EJB Container

As mentioned earlier, the EJB container provides AIMSOA with extended scalability by providing services such as clustering, connection pooling, and failover to the developers. The EJB container is where the majority of IM processing is done.

Middle/Application Layer: Session Beans

Session beans are modular and consist of three classes: a home interface, a bean interface, and a bean implementation. The home interface provides access to the bean interface. The bean interface is what a developer uses to invoke methods on the bean's implementation class. The J2EE application server can be used to configure either

stateful or stateless session beans. Stateless session beans can be implemented to provide the necessary processing for the majority of IM operations. Session beans can also be implemented as stateful if the developer needs a more stateful environment for processing. Beans are categorized into local and remote beans. Remote beans utilize Java RMI in order to function on other machines; this allows for greater scalability and distributed functionality. If the performance requirements are not as necessary, local beans can be implemented on the same server.

AIMSOA provides the functionality for complete IM application development as well as wrapping functionality for existing IM applications. IM vendors that seek the benefits of AIMSOA can wrap their code around AIMSOA. In this scenario, instead of using session beans to implement custom IM functionality, session beans will act as clients to the vendor's software, forwarding and receiving messages to the vendor's server. AIMSOA will then take on the role as a new client to the vendor's implementation.

Data Layer: Data Tier

Entity beans interact directly with a database and provide Java developers with an object oriented view of its tables. Entity beans also consist of an interface, implementation, and home class. Entity beans and the EJB container provide developers with concurrency control and other database specific necessities. If needed, Lightweight Directory Access Protocols (LDAP) can also be accessed in this tier from EJBs.

Methodology

The J2EE architecture is chosen as the base architecture of choice because of its powerful distributed computing infrastructure and service-oriented capabilities. As discussed earlier, software applications developed today are based on either the J2EE or .NET platform. The .NET platform offers tools and subcomponents analogous to those of J2EE. Because .NET is developed and controlled by Microsoft Corporation, which is known to be extremely proprietary, AIMSOA is designed around the J2EE architecture. Additionally, and although .NET utilizes a comparative C# language, Java has proven to be an extremely portable application.

The SOAP protocol is used for messaging between parties. The underlying transport protocol is HTTP. Although HTTP is an unreliable transport mechanism without store and forward capabilities, it is a universal standard transport protocol for communication, which is necessary to achieve interoperability with existing systems. JAXM providers provide the reliability and forwarding capabilities needed for messaging.

This paper only presents an architecture to where a Web service can be wrapped around existing or to be developed fully functional IM applications. To show the benefits of this architecture, three sequence diagrams of operations are displayed in figures 10, 11, and 12 respectively: registration, login, send, and receive messages. Registration and login are shown on the same diagram. Fortunately, the process for implementing these four operations is the same process that can be utilized for most IM services to create fully featured robust applications.

Sequence Diagrams of Methods

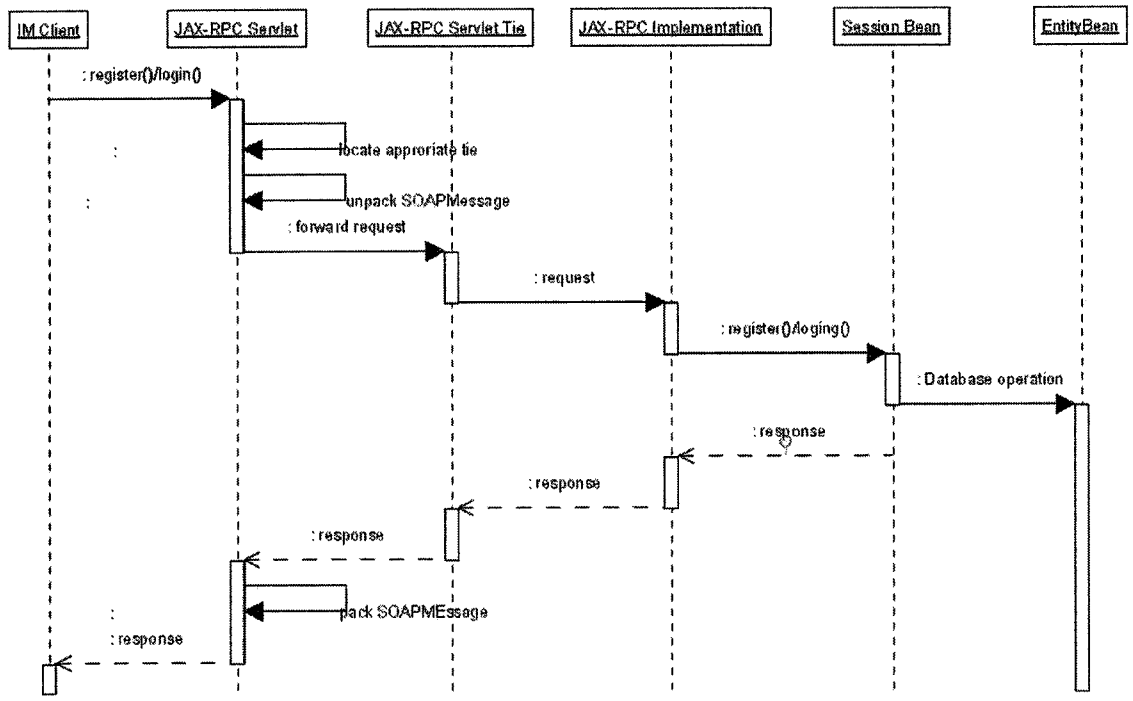

Figure 10: Sequence Diagram for login and registration IM operations

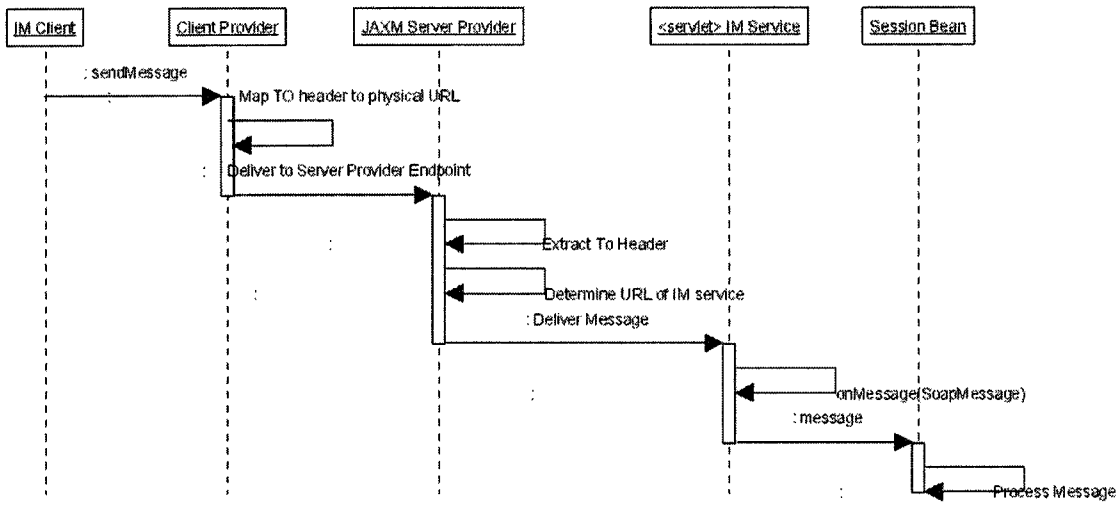

Figure 11: Sequence Diagram for Send Message operations

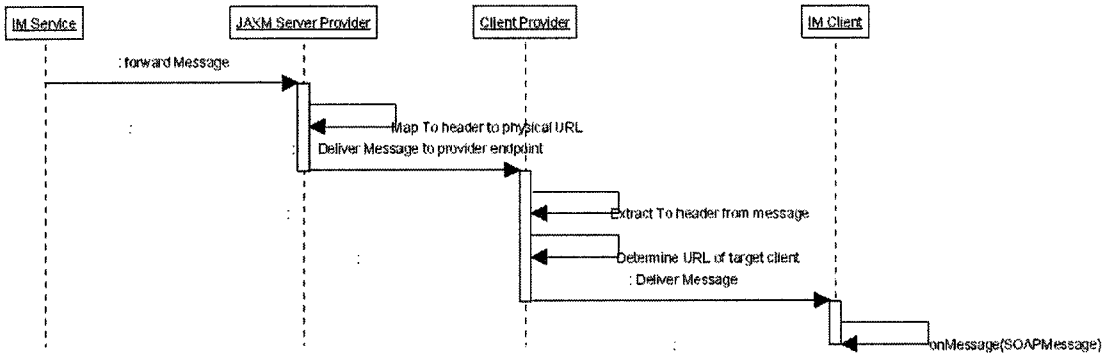

Figure 12: Sequence Diagram for Receiving Message operations

CHAPTER FOUR

DISCUSSION

The methodology discusses how clients can bind to AIMSOA and how AIMSOA provides IM functionality. To be fully considered a service-oriented architecture, clients must be able to find or locate the service in some type of registry. WSDL files are the entities that get published to a registry. An example of a WSDL file for an instant messenger that implements the registration operation can be found in Figure 13. Because AIMSOA uses ebXML as a SOAP profile to send messages, an ebXML registry is the registry to be used with AIMSOA. Because publishing and locating services are standard and well known operations in SOAs, they will not be discussed further in this paper, but can be referenced in McGovern (2003).

```
- <definitions name="JimService" targetNamespace="urn:Jim">
  - <types>
    - <schema targetNamespace="urn:Jim">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      - <complexType name="JimError">
        - <sequence>
            <element name="message" type="string"/>
          </sequence>
        </complexType>
      </schema>
    </types>
  - <message name="JimRegistration_registerJb">
      <part name="String_1" type="xsd:string"/>
      <part name="String_2" type="xsd:string"/>
      <part name="String_3" type="xsd:string"/>
      <part name="String_4" type="xsd:string"/>
    </message>
  - <message name="JimRegistration_registerJbResponse">
      <part name="result" type="xsd:boolean"/>
    </message>
  - <message name="JimError">
      <part name="JimError" type="tns:JimError"/>
    </message>
  - <portType name="JimRegistration">
    - <operation name="registerJb" parameterOrder="String_1 String_2 String_3 String_4">
        <input message="tns:JimRegistration_registerJb"/>
        <output message="tns:JimRegistration_registerJbResponse"/>
        <fault name="JimError" message="tns:JimError"/>
      </operation>
    </portType>
  - <binding name="JimRegistrationBinding" type="tns:JimRegistration">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    - <operation name="registerJb">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" namespace="urn:Jim"/>
        </input>
      - <output>
          <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" namespace="urn:Jim"/>
        </output>
      - <fault name="JimError">
          <soap:fault name="JimError" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" namespace="urn:Jim"/>
        </fault>
      </operation>
    </binding>
  - <service name="JimService">
    - <port name="JimRegistrationPort" binding="tns:JimRegistrationBinding">
        <soap:address location="http://thomas-p406g3gd.cau.edu:8080/jim-jaxrpc/jim"/>
      </port>
    </service>
</definitions>
```

Figure 13: WSDL for implementing only IM registration functionality

This paper also introduced several security mechanisms that can be applied to AIMSOA to deliver secured messaging. Security is a very exhaustive topic and cannot be adequately discussed within the scope of this paper. The goal here is to present the

many security related technologies that can be and should be used in conjunction with AIMSOA.

SOAP functions over a variety of transports, therefore, HTTPS can be used to deliver messages over SSL. SSL allows for secure encrypted document transmission over the transport. SSL does not handle security requirements once IM messages are persisted nor does it possess the capability to encrypt specific parts of a SOAP document carrying the IM message. Additionally, most SSL deployments only utilize server side authentication through server side certificates. In a distributed secure SOA oriented IM environment, its necessary to provide client/requestor side authentication. SSL operates at the session layer of the OSI model and cannot secure communication end to end; it can only secure communication to the next hop. The chain in a communication path may involve un-trusted links or communication that lacks adequate security. These are the reasons why IM vendors such as AIM do not allow third party vendors such as Jabber and Trillion to use their protocol. SSL also cannot protect AIMSOA from buffer overflow and replay attacks. For these reasons, other security mechanisms such as XML encryption and Web service security specifications are better suited for secure messaging..

SAML, XML digital signatures, XACML, XKMS, and XML encryption are all XML security strategies for securing XML based SOAP messages. These mechanisms offer security at the message level instead of at the session or transport level. XML encryption allows for SOAP messages to be encrypted designated by the *EncryptedKey* and *EncryptedData* XML fields. SAML can be used to guarantee a certain level of

security in a distributed environment by providing authorization and authentication information that can be maintained across participating parties. XML Digital Signatures is a standard that allows XML to functionally sign itself over insecure networks. XACML can provide access control to individual messages being transferred in an Instant Messenger; only users with the rights to access the message can read or modify the message. SOA security mechanisms support a variety of encryption mechanisms including AES, Blowfish, CAST-256, GOST, IDEA, RC-6, Serpent, Triple DES, and Twofish. WS-I specifications are Web service specific security specifications that can provide additional trust, policy, proof of identity, privacy, and authorization requirements to instant messengers and the messages they deliver.

Security requirements for an instant messenger are based on the requirements of the parties and organizations the IM looks to serve. This section presents a summary of the many security related mechanisms that can be applied to AIMSOA to satisfy such requirements. For a more thorough investigation on how to implement the security features mentioned in this paper, please refer to McGovern (2003).

The main limitation in the AIMSOA architecture is the lack of automatic WSDL file generation. JAX-RPC provides tools for automatic WSDL file generation, but JAXM does not, and when used conjunctively, there is no support. This is a major drawback as WSDL files are very extensive and error prone. A future study is needed to provide such functionality. Unfortunately, because JAXM is not included in the J2EE architecture, vendors are not expected to provide such support. In addition, JAXM is not included in the J2EE standard specification or implantation. J2EE vendors are not required to

support it, therefore, portability may suffer when moving IM code from one application server to another. JAXM has been tested and deployed on the Sun Java Application Server.

The last limitation to this architecture lies in performance. Because SOAP messages are transferred between client and server, parsing must be done behind the scenes to grab the necessary information. This is a performance sacrifice for standardization and interoperability. Likewise, document oriented SOAP message can become large, and may occupy bandwidth. Because SOAP supports messages with attachments, IM applications can implement functionality that allows for files to be included in the sending of messages. Binary data formats can greatly reduce the bandwidth required by a system as well as provide other features such as error detection and correction. These benefits are sacrificed for standardization and interoperability that XML based messaging provides. These performance limitations are acceptable for the IM environment unless attachments are used heavily, clogging the bandwidth necessary for communication. For attachment use, IM requirements should specify maximum attachment sizes very much like email systems utilize. In addition, J2EE provides tools to zip and unzip files for faster delivery to limit the effects of large file transfer. There also may be a small performance cost for implementations that intend to use AIMSOA as a wrapper to existing IM functionality because AIMSOA adds an additional layer of communication and processing. Again, this cost is very much acceptable for the majority of IM environments.

In conclusion, AIMSOA can be used to augment existing architectures to provide better scalability, interoperability, flexibility, and security. AIMSOA can also be used to create robust custom IM systems that mirror the same functionality. J2EE provides ideal service-oriented implementations as well as a robust application programming platform. Vendors such as AIM, Jabber, Yahoo!, and MSN can utilize AIMSOA to help mitigate the many problems they are facing at minor costs. Technologies such as EJBs, JAX-RPC, JAXM, and servlets can be combined to provide an extremely loosely coupled, interoperable IM architecture.

REFERENCES

Bruno, Eric. "Java Web Services & Application Architectures." *Dr. Dobb's Journal,* February 2005, 16-21.

Fischer, Peter. "Answering the critical Web services questions." *Application Development Trends,* July 2002, 52.

Knorr, Eric. "A Field Guide to Hosted Apps." *InfoWorld,* 1 April 2005, 39-40.

McGovern, James, Sameer Tyagi, Michael Stevens, and Sunil Mathew. *Java Web Services Architecture*. San Francisico: Morgan Kaufmann Publishers, 2003.

Richard, Kevin. "AOL, ICQ to Interoperate—But in a Limited Fashion." *atnewyork.com*, 30 October 2002. Article on-line. Available from http://www.atnewyork.com/news/article.php/1490771. Internet.

Saunders, Christopher. "Yahoo! Closes the Door on Third Party IM." *Instant Messaging Planet.com,* 26 September 2003. Article on-line. Available from http://itmanagement.earthweb.com/entdev/article.php/3084151. Internet.

_____. "Report: IM Viruses on the Rise." *Instant Messaging Planet.com,* 1 October 2003. Article on-line. Available from http://www.instantmessagingplanet.com/security/article.php/3086291. Internet.

Sengupta, Som and Venkatakrishnan Padmanabhan."A Service-Oriented Architecture for Enabling Centralized Authentication Across WebLogic Domains." *dev2dev*, September 2004, 1-10.

Shigeoka, Iain. *Instant Messaging in Java.* Greenwich: Manning, 2002.

Svoboda, Zdenek. "Securing Web Services with Single Sign-On: Developing Web Services Series Part VI." *Systinet,* February 2002. Paper on-line. Available from

http://www.theserverside.com/articles/article.tss?l=Systinet-web-services-part -6 Internet.

Weaver, James L. and Kevin Mukhar. *Beginning J2EE 1.4.* Birmingham: Wroxx Press, 2003.

Woods, Bob. "Jabber goes to the Web," *Instant Messaging Planet.com,* 29 November 2001. Article on-line. Available from http://www.instantmessagingplanet.com/enterprise/article.php/928171. Internet.

_____. "Report: Interoperability Coming to IM" 8 *Instant Messaging Planet.com, 8* Jan 2002. Article on-line. Available from http://www.instantmessagingplanet.com/public/article.php/950431. Internet.

_____. "New: Web-based IM," *Instant Messaging Planet.com,* 21 May 2002. Article on-line. Available from http://www.instantmessagingplanet.com/public/article.php/1142121. Internet.

_____. "Review MSN Messenger," *Instant Messaging Planet.com*, 7 June 2002. Article on-line. Available from http://www.instantmessagingplanet.com/public/article.php/1348031. Internet.

_____. "Bantu Restores Yahoo Interoperability," *Instant Messaging Planet.com,* 10 June 2002. Article on-line. Available from http://www.instantmessagingworld.com/public/article.php/1356811. Internet.

_____."Impasse Rolls out IM Security Platform" *atnewyork.com* 20 June 2002. Article on-line. Available from http://www.atnewyork.com/news/article.php/1368821. Internet.

server side passing *client/server* flags respectively as input to the command. Other platforms have their own tools to bind languages such as C# and C to the service. To invoke methods exposed by RPC based Web services, the location of the service, protocol, and the service's methods and signatures must be understood beforehand. An application that is deployed based on the AIMSOA architecture must have a WSDL file that can be located from an ebXML registry or URL. The WSDL file contains the protocol to be used (HTTP) and the methods and method signatures of the IM service. *Wscompile* is responsible for importing the WSDL file and generating specific Serializer, Deserializer, and stub classes for the client. The stubs, in conjunction with other *wscompile* generated classes, are used to handle any conversions needed such as data type mappings to communicate with the service.

In the architecture diagram, two clients are depicted: one a basic IM client and the other a registration client. With the exception of registration, the basic IM client can implement all necessary IM operations. The registration client is used specifically for registration, thus only interacting with the JAX-RPC servlet. A client developer can choose to merge the functionality into a single interface. There are no performance disadvantages, just a matter of style and abstraction.

Middle Layer: Provider

Providers act as proxies to forward messages to other users. The JAXM provider can be configured to forward messages to endpoints, either server endpoints or other provider endpoints. If an external client is deployed on a platform other than J2EE, any

server side passing *client/server* flags respectively as input to the command. Other platforms have their own tools to bind languages such as C# and C to the service. To invoke methods exposed by RPC based Web services, the location of the service, protocol, and the service's methods and signatures must be understood beforehand. An application that is deployed based on the AIMSOA architecture must have a WSDL file that can be located from an ebXML registry or URL. The WSDL file contains the protocol to be used (HTTP) and the methods and method signatures of the IM service. *Wscompile* is responsible for importing the WSDL file and generating specific Serializer, Deserializer, and stub classes for the client. The stubs, in conjunction with other *wscompile* generated classes, are used to handle any conversions needed such as data type mappings to communicate with the service.

In the architecture diagram, two clients are depicted: one a basic IM client and the other a registration client. With the exception of registration, the basic IM client can implement all necessary IM operations. The registration client is used specifically for registration, thus only interacting with the JAX-RPC servlet. A client developer can choose to merge the functionality into a single interface. There are no performance disadvantages, just a matter of style and abstraction.

Middle Layer: Provider

Providers act as proxies to forward messages to other users. The JAXM provider can be configured to forward messages to endpoints, either server endpoints or other provider endpoints. If an external client is deployed on a platform other than J2EE, any

CHAPTER THREE

DESIGN

ARCHITECURE

AIMSOA is a modular architecture, and can be seen in Figure 9. The architecture is broken into four tiers: user, service, application/web, and data access. The middle layer encompasses both the service tier and the application tier. The IM client and the IM registration client fall into the presentation layer. JAX-RPC and JAXM provide the services layer. The servlets and EJBs perform the business logic or server processing of the architecture. The database makes up the data access layer. A combination of JAX-RPC and JAXM communication APIs have been chosen for the AIMSOA messaging solution.

CHAPTER THREE

DESIGN

ARCHITECURE

AIMSOA is a modular architecture, and can be seen in Figure 9. The architecture is broken into four tiers: user, service, application/web, and data access. The middle layer encompasses both the service tier and the application tier. The IM client and the IM registration client fall into the presentation layer. JAX-RPC and JAXM provide the services layer. The servlets and EJBs perform the business logic or server processing of the architecture. The database makes up the data access layer. A combination of JAX-RPC and JAXM communication APIs have been chosen for the AIMSOA messaging solution.