A SECURE INFORMATION INFRASTRUCTURE FOR SERVICE ORIENTED ARCHITECTURES

by

Joseph Giacomo St. Onge

B.S., University of Pittsburgh, 2004

Submitted to the Graduate Faculty of

the School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2006

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This thesis was presented

by

Joseph Giacomo St. Onge

It was defended on

April 28, 2006

and approved by

Alex K. Jones, Assistant Professor, Department of Electrical and Computer Engineering

J.T. (Tom) Cain, Professor, Department of Electrical and Computer Engineering

Kim LaScola Needy, Associate Professor, Department of Industrial Engineering

Thesis Advisor: Raymond R. Hoare, Ph.D., Assistant Professor, Department of Electrical and Computer Engineering

A SECURE INFORMATION INFRASTRUCTURE FOR SERVICE ORIENTED ARCHITECTURES

Joseph Giacomo St. Onge, M.S.

University of Pittsburgh, 2006

In today's ever-evolving design environments, a focus switch is needed from workstation-centric software tools to distributed services. For Computer–Aided Design, the use of distributed services has the potential to incorporate all of the needed software features for a given project into a new design system that utilizes services. Thus, the designer would have access to features that are not locally installed. This thesis presents a secure middleware solution for design environments. The secure middleware solution provides a system architecture and information infrastructure to facilitate the needs of the designer while also providing access to remote services. The system architecture and information infrastructure are designed with the designer in mind by providing access to any file at any time at any location, and the ability to submit jobs to any available services. These fundamental components are implemented as to not compromise security or accountability. Enabling the system architecture are four fundamental technologies created for this system. They include: (1) a Secure Java Messaging Service, (2) Verification Services, (3) Gateway and Directory Services, and (4) a Secure File System. Through the creation of these four technologies, the system architecture and information infrastructure was developed and deployed into a simulated design environment. Results showing the benefits of this design environment over other design environments are explored within this thesis. Overall, the secure middleware solution for design environments benefits designers and enterprises in a secure, traceable, and accountable manner.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

First and foremost I would like to thank Dr. Bartholomew O. Nnaji for his vision of distributed design environments within Computer–Aided Design. Through his support and dedication to the NSF Center for e-Design, he provided the motivation for this research through guidance, conferences, and collaborative efforts. This vision of service oriented architectures would not have been possible without him.

This thesis would not have been possible without the support of my committee members as well. They offered guidance and wisdom that helped shaped this thesis into the academic paper it is today. Dr. Needy, I thank you for your class on Engineering Management. It gave me insight into everyday life and helped me to better manage my time, a goal I sought to meet when beginning that class. This work was partially supported by the following organizations: National Science Foundation – Center for e-Design, Science Applications International Corporation (SAIC), and Alcoa. I'd also like to thank the School of Engineering for all of its support for attaining a higher level of education in the engineering fields. The world is always in need of more engineers. Another special thank-you goes out to all my engineering partners that have shared many nights in a lab in pursuit of solving a problem or designing a FPGA circuit. Long live DOOM on the ARM!

I would like to thank my parents for their unending support of all that I do. Hopefully, I'll be able to repay you for the financial support that got me through my Bachelor's degree. However, I can never repay you for all the love and spiritual support you have given me all 24 years of my life. Truly, I am grateful for that.

I'd also like to thank my friends for their support of my degrees and academic work. Brian, Jarod, Ashley, Martin, Ryan, and all my friends, thanks for understanding when my academic work became a priority in my life. We will all meet again soon and have many stories to share. Thanks to everyone in my life as it has been a wonderful 6 years!

Finally, I’d like to thank my academic advisor, Dr. Raymond Hoare, for everything I can possibly thank him for. He has been there for me from before I entered graduate school. Without him, I would not have achieved my Bachelor’s degree on time, became a graduate student, made it through my first graduate semester, and I definitely would not have accomplished this final thesis. You have affected my life in more ways than you know. Thank you.

1.0 INTRODUCTION

Distributed design environments provide many benefits over software packages alone. These benefits facilitate global and collaborative design. Although software packages today have more features than in recent years, they are still restricted to their built-in features and cannot access other, possibly improved features without additional software or software updates. The method of "upgrading without upgrading" does not exist within software packages today; this entails adding new products and features without purchasing new software, upgrading existing software, or reinstalling current software. Also, software packages do not allow for dynamically updating components, and some software packages do not allow for distributed and collaborative design environments.

With constantly changing software packages and globally expanding networks, there is an increasing need for a distributed design environment that incorporates multiple software components into one comprehensive solution. This design environment needs to be constructed with the designer in mind, instead of the software in mind, to increase productivity and encourage collaborative design methodologies. Currently, designers still have to remember where they saved a certain file or what workstation has which software packages installed on it, to work on or process their information. This results in design files sometimes becoming lost, causing many headaches for the designer and their colleagues when working on collaborative design files.

New design environments need to shift focus from the workstation to remote, distributive services. This would incorporate many of the needed software features into a complete design solution and would not restrict the designer to only those features within specific software packages. The design environments would focus on the following concepts: (1) the designer should be able to access any file, from anywhere, at any time; (2) the designer should have access to any service they desire. Dependencies on specific software packages and specific

workstations would be unnecessary as third party providers would provide the features typically found in software packages. This includes unrestricted access to software features that can only be utilized within specific operating systems. With this focus shift comes the need for an integrated architecture within a common framework.

The integrated architecture must be seamless to the end-user to provide maximum efficiency during design development. A designer should not be aware of the processes occurring in the background of the system, but instead think that the features they select within their design software are processing locally on their workstation. Seamless integration within current design software is one of the key focuses of this thesis.

There are many security technologies available today that can be used within a design environment's architecture. Most of them lack a cohesive middleware to integrate the many security benefits into one comprehensive solution. Today's technologies lack an integrated solution that can be built upon to form a complete, robust, and integrated architecture. This thesis defines fundamental technologies that *can* be built upon in a manner that facilitates an integrated architecture. These technologies provide a secure middleware for arbitrary services that appear local to the designer, but are really distributed and secure.

The services accessed within this architecture can incorporate many different features that are cumbersome to setup and administer within a local design environment. One example of a service would be a Finite Element Analysis (FEA) service. To setup a FEA within a local construct would involve several computational machines and a way to enforce access to those machines. If a service was provided by a third party service provider, that access could be centralized within the enterprise and allow many computations to occur, without adding new machines or new software. To the designer it would appear local, but it would really be remote.

Within the several programming languages that exist today, a particular programming language offers point solutions that incorporate security and allow the many features of a distributed environment to exist. Java provides a host of Application Programmable Interfaces that best offers these benefits. Java also provides the ability to facilitate the several different components of a secure, traceable, and non-reputable system architecture, acting as a middleware within design environments; but it does not provide an integrated solution when used alone. Through the use of the Java programming language, a system architecture with these capabilities is formed and several problems relating to these issues are resolved.

After knowledge of the benefits new design environments offer, the problems within these environments are clear. How do you secure the information infrastructure between an enterprise and a service provider, while also incorporating traceability and non-repudiation?

This thesis seeks to define methods that solve these problems, providing a robust middleware for arbitrary services. This middleware can be applied to any design environment to implement the following technologies: (1) security of the information within the infrastructure (2) traceability of design objects and accesses of those objects (3) non-repudiation to ensure peer to peer integrity and honesty.

1.1 PROBLEMS OF EXISTING DESIGN ENVIRONMENTS

Existing design environments have a lack of point solutions and lack a "secure information infrastructure" that integrates those point solutions into a cohesive middleware solution. There must be solutions for guaranteed sending and receiving of information over the infrastructure and those solutions must employ the highest of security standards and traceability. Information sharing of arbitrary data is essential within distributed design environments. A method of sharing this data over common channels is needed to facilitate interoperability between different entities within design environments. *Security*, *traceability*, and *non-repudiation* are lacking within existing design environments. These main concepts must be implemented for a complete integrated solution to exist.

Security is very important when applied to CAD design environments. Designers and CAD providers are extremely cautious of opening their designs to hostile networks, such as the Internet. Security involves encrypting the files sent over networks while also providing perimeter security for those networks. Opening designs to the Internet is essential to the information infrastructure because it provides distributed and collaborative design efforts.

Traceability is important when determining who accessed sensitive information within a system. This is especially important in Computer Aided Design environments and medial fields. Both CAD designs and medical records are considered very sensitive material and a record of who accessed the information and when they accessed it is necessary to retain integrity. For example, certain CAD designs may have restricted access to authorized personnel only.

Traceability provides methods to determine if any unauthorized access to those files has occurred.

Non-repudiation ensures that a transferred object has been sent and received by the parties claiming to have sent and received the object. Non-repudiation is a way to guarantee that the sender of an object cannot later deny having sent the object and that the recipient cannot deny having received the object. This is fundamental to the remote services concept because how is one guaranteed that a provider has successfully received a digital job, or that a design firm successfully sent a digital job to the service provider? For example, a design firm cannot send its intellectual information and then later deny sending it. At the same time, the service provider who is receiving that information cannot deny receiving it. Non-repudiation allows future disputes over data and information to be resolved quickly and efficiently.

1.2 A SECURE MIDDLEWARE SOLUTION FOR DESIGN ENVIRONMENTS

The proposed system architecture described within this thesis is a comprehensive and secure middleware solution that solves many of the design environment problems that exist today. It provides a system architecture that has the ability to dynamically and seamlessly incorporate: *design objects* for interoperability, *software components* for constantly evolving and improving design environments, and *remote computation providers* for value added services and features. It also provides an information infrastructure for transmitting data and software throughout the Internet to remote computation providers, without compromising security, traceability, and non-repudiation.

To solve the aforementioned problems within design environments, four core contributions that are fundamental to supporting the system architecture and information infrastructure are introduced in this thesis. Those contributions include:

(1) **Secure JMS**: a secure and robust messaging service

(2) **Verification Services:** a transaction verification and tracking mechanism

(3) **Gateway Services:** a service firewall

(4) **Secure File Sharing System**: a system for secure file sharing and distribution.

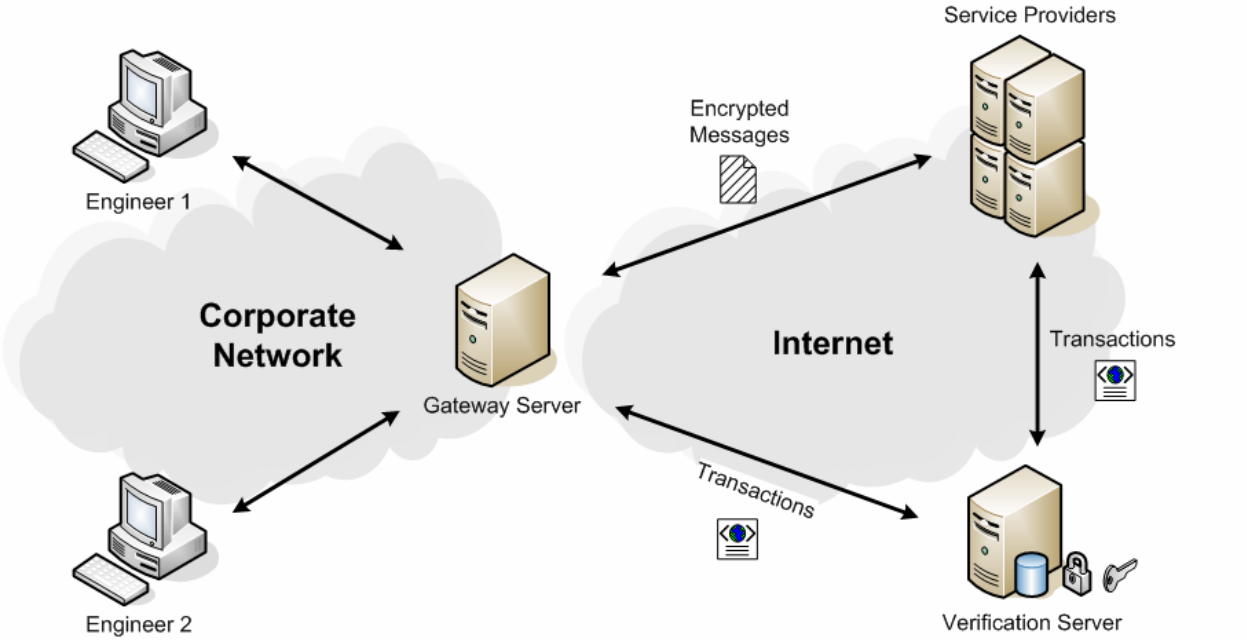


Figure 1. System Architecture Overview

Figure 1 displays a system overview of the integrated architecture and the interconnectivity between the various system entities. The entire system seamlessly integrates into any existing design environment and is transparent to the user. This figure shows several different functions occurring within the system architecture. Engineer 1 and 2 are accessing allowed services through the use of the Gateway Server, which acts as a service firewall preventing unauthorized access to services. Even though these services are remote, they appear local to the engineers. The enterprise is passing encrypted messages through the messaging service to the service provider, which contains files, jobs, or user information. These messages are recorded and tracked through the use of the Verification Server, which both the enterprise and service provider utilize for non-repudiation.

Each fundamental technology created for the system architecture provides certain features and benefits that incorporate security, traceability, and non-repudiation. Also, these technologies integrate seamlessly into current design environments, providing maximum efficiency during collaborative design efforts. The following sections give a brief overview of each technology within the integrated system architecture.

1.3 SECURE JAVA MESSAGING SERVICE

Messages sent between client enterprises and service providers within the system architecture need to be secure, traceable, and non-reputable. Also, these messages need to provide methods of transmitting META information, job requests or responses, and storage of file information, such as design or source code files. Without security, traceability, and non-reputability, messages sent within the system architecture would be completely vulnerable to malicious attacks. Messages would be unsecured and outside sources could view the contents of the message. Messages would also be non-traceable, preventing a record of message access to be stored by a third party sharing trust between two entities.

A method of sending secured JMS messages does not exist today within an API construct. "Secure" Java Messaging Systems have only utilized Secure Socket Layers for transport of messages within JMS frameworks. This is partially due to JMS not specifying a security context or an API for controlling message confidentiality and integrity. Also, JMS is controlled by a system administrator rather than implemented programmatically or by the J2EE server runtime. A new API had to be developed to implement message confidentiality and integrity to facilitate communication between client enterprises and service providers within the system architecture of the middleware solution. This is where Secure JMS applies its technologies.

This thesis introduces Secure Java Messaging Service, or Secure JMS, as a method of sending and receiving secure, encrypted JMS messages through standard JMS applications and frameworks. It essentially interconnects the other fundamental technologies together. It solves security issues by using Advanced Encryption Standards (AES) to encrypt each message that is sent over hostile networks, such as the Internet. It also solves many traceability and non-repudiation problems by storing transactional information, including encryption keys and hashes, in the Verification Server.

Secure JMS is modular in nature and integrates seamlessly into existing JMS frameworks, benefiting any client enterprise that already utilizes JMS in their infrastructures. This benefits client enterprises by alleviating integration issues for already existing source codes. To facilitate this, only one line of mode modification is needed to utilize Secure JMS's features, including database storage of transactional information.

When performance tests were executed on both Regular JMS and Secure JMS, the difference between the two was found to be minimal. Standard JMS had approximately seven hundred and four kilobytes per second total throughput when sending and receiving large files. In comparison, Secure JMS had only six hundred and sixty eight kilobytes per second total throughput when sending and receiving the same large files. This thirty-six kilobyte difference is negligible when compared to standard Internet up-stream connections that most enterprises have. This is described in detail within Chapter 3.

1.4 VERIFICATION SERVICES

Transactions are essential within distributed design environments. All information shared between enterprises and service providers must be recorded and traced to facilitate non-repudiation and integrity of information sent over the information infrastructure. A third party company that shares trust among enterprises and service providers must provide this transactional information storage. This company must also be available across multiple networks to anyone wishing to utilize its services of transactional verification.

These requirements accumulate together to form the basis for the verification services introduced in this thesis. The verification services act as a third party company that verifies which services are available to an enterprise. It also records all transactional information for any information that is exchanged throughout the system. Verification services can be used for any disputes over transactions between entities that utilize its services of transactional storage. When the system stores its transactions with the Verification Server, it is assumed that each party's identity has already been verified through existing means, such as Verisign. Verification Services ensure completion and accountability of transactions and provides a record of services accessed. This is integrated with Secure JMS and described in detail within Chapter 3.

1.5 GATEWAY AND DIRECTORY SERVICES

When enterprises, such as CAD providers, need to access the Internet and remote computational services, a method of central protection and control is necessary to protect their interests and intellectual property. Information cannot be freely accessible by outside sources, and computational services cannot be accessed without management's authorization.

Previous methods of network protection consisted of hardware implementations through network segregation and through the use of firewalls to block incoming network traffic. A system administrator needed to be notified when a "hole" or throughway needed to be made so that certain network enabled services could function properly. This can sometimes be cumbersome and may take days to finally get the appropriate service authorized. It also defeats the security the firewall was installed to enforce.

The Gateway Server is a solution that enables and controls the enterprise from a single point of control, allowing easy management of remote services and network protection through the use of a software firewall. Remote services are accessible to the enterprise through its use. Management has the ability to dynamically control which products and services are available by accessing the Gateway Server and making the necessary modifications to the enterprise policies. Since management is only concerned with what services are accessible within their enterprises, and which service providers can provide those services, the Gateway Server caters to management's needs, simplifying the access and control of the enterprise's property. Thus, the hole in the firewall is actively managed at the application layer and security is enforced.

At the same time as providing a software firewall, the server must also keep track of contractual information such as which enterprises are allowed to access what services, or what enterprises a service provider is allowed to serve. This results in the directory services portion of the Gateway Server. This is controlled by management based on contractual information and ties into management's dynamic control ability. Gateway and Directory Services are infrastructure services and more information about them is within Chapter 4.

1.6 SECURE FILE SHARING SYSTEM

When companies need to share and store large files within their distributed enterprises, many issues complicate simple file sharing techniques such as FTP (File Transfer Protocol) or Windows File Sharing. Many of these issues focus directly on the security and integrity of the files accessed. How can an enterprise's files be accessed remotely without arbitrarily punching holes through the enterprise's firewall? How can traceability occur on the files accessed to eliminate and prevent unauthorized access to those files? What about burdening file transfer clients that may only work within specific operating systems? In addition to these concerns, the file sharing system must be distributive to allow for globally enabled enterprises. Coordinating global firewall rules is a hassle no administrator wishes to deal with.

Some prior work on distributive systems includes works within academia and industry. Though these systems have many benefits, a common problem among them is that there is no built in encryption for file storage within distributive systems. At the same time, they conflict with cross-platform compatibility – the ability to run identical programs or source codes on many hardware platforms. There are e-mail and web based applications available on the market today, which share files remotely and offer security and authorized access control. However, those applications have systematic flaws that prevent them from being utilized within the system architecture this thesis presents. Only one complete solution incorporates all the necessary security and distributive features that compliment and facilitate the system architecture, Secure File Sharing System.

Secure File Sharing System provides methods of secured file transfer, file storage, and file access without the necessity of burdening file transfer clients. Files are shared with simple web browsers and uploaded via Secure Hypertext Transfer Protocol (HTTPS) to a series of distributive file servers. The distributive file servers employ a doubly encryption method that guarantees that even in the event of a server physically being broken into, the data cannot be extracted via hard disk extraction methodologies. The distributive nature of Secure FSS is useful for twenty-four hour design work that occurs over an enterprise's global network.

Integrating several different security features into an all-in-one inclusive package solves the major problems of secure file transfer and secure file storage. Through the use of standard secure protocols and methods (e.g. HTTPS, SSL, MD5, and GnuPG), the trustworthiness of the

system is very high and reliable. Secure FSS is distributive in nature, helping to make a globally available file sharing system to any enterprise that wishes to deploy it. The design and implementation of Secure FSS is shown within Chapter 5.

The remainder of this thesis separates into five chapters, each providing an individual look at the secure middleware solution. Chapter 2 lists prior work in all the technologies and their components within the secure middleware solution. Chapter 3 describes enabling secure and traceable communications, which includes the Secure Java Messaging Service and the Verification Services. Chapter 4 details the infrastructure services that utilize the system architecture to perform enterprise and provider functions. Chapter 5 describes the Secure File Sharing System and explains how it can be used in any environment. Chapter 6 is a demonstration of the capabilities and functionality of the integrated solution through the use of Eclipse. Finally, Chapter 7 concludes this thesis, summarizing each technology and providing future work capabilities for those technologies.

With knowledge of what fundamental technologies were necessary to facilitate the complete system architecture and information infrastructure, a more detailed look at prior work within the fundamental technologies will be explored in the next chapter. Many benefits of the prior work were improved upon and implemented within the system architecture.

2.0 PRIOR WORK AND RELATED TECHNOLOGY

A discussion of the various existing technologies, of which are within industry and academia, will be explored in this chapter to highlight the various features and benefits that were utilized when developing and implementing the system architecture and information infrastructure for the new middleware solution. The shortcomings that each of these technologies have will be highlighted as well to illustrate how the new solution improves on the existing technologies to make a more robust and secure design environment. Topics include dynamically updating software, network data transmission technologies (both secure and non-secure), application firewalls, distributive systems, and secure file systems.

2.1 DYNAMICALLY UPDATING SOFTWARE

Dynamically updating software has been an ongoing and involved research topic since the invention of software programming. Many dissertations in Computer Science within multiple institutions have all discussed the most efficient ways to implement dynamically updating software. Within Industry, two companies have been most notable for developing and deploying this technology, Microsoft Corporation and the Eclipse Foundation.

2.1.1 Dissertation on Dynamic Software Updating – Michael Hicks

Michael Hicks wrote a dissertation on dynamically updating software, in the department of Computer and Information Science at the University of Pennsylvania in 2001, which discussed the benefits of dynamic patches that contained both the transition code and the newly updated code. This allowed for easy transitioning from the old version of the software to the new version

[5]. He developed and implemented such techniques including building on dynamic linking, using verifiable native code, dynamic patching, and generating most module patches automatically.

Building dynamic linking involves starting with a dynamic linking approach and then building flexibility on top of it. This must be done while keeping a simple implementation, and developing an automated way that converts every program module into a plug-in.

Typed Assemble Language is used to define a framework in which native machine code is tied together with annotations to prove that the code is safe for execution. This aids in avoiding the performance cost of byte-code interpretation and the security cost of having a complier in the Trusted Computing Base [5].

Dynamic patching involves deploying patches that contain two pieces of information: the new version of the code module and the data needed to support updating that module dynamically. An important part of the dynamic patching process is a state transformer function that computes the new module's starting state from that of the old module [5].

Hicks also wrote a tool that identifies all changes to a program from one version to another and automatically generates the necessary patches. Changes were addressed either by generating some patch code, or by inserting a placeholder when generating code was not possible, so that the programmer may address the change. This tool ensures that patches are complete and makes the system easier to use.

Although these concepts apply mostly to software that usually runs continuously and without interruption, they can be applied to the design environment in the same manner. The concepts applied include flexibility, robustness, efficiency, and ease of use, all of which the middleware solution employs throughout its system architecture and information infrastructure. This is an alternative to services but without a computational server. It is a single source application solution.

2.1.2 Microsoft Component Object Model

Microsoft has developed and deployed their Component Object Model Technology (Microsoft COM). This technology performs a multitude of functions that facilitate the theories behind dynamically updating software. It enables interactive content from the Internet through the use

of ActiveX, allows for re-usable software components, and links software components together to build dynamic applications. The Internet today contains a rich source of multimedia and dynamic applications because of Microsoft COM [2].

While all these features and benefits would seem to work with this environment, Microsoft COM has some major problems that are not compatible with the design specifications of the system architecture. The most significant problem is that it lacks robust security. Microsoft COM is susceptible to many viruses, worms, and trojans that are manifesting on the Internet today. These security breaches cause arbitrary code to be executed on a user's system without authorization of that user. This can cause multiple, serious issues within the user's operating system like deletion of files, duplication of the viruses or worms that infected the system, or even unauthorized access to the user's files. This alone prevents Microsoft COM from being utilized as the software component for dynamically updating software within the new design environment.

Microsoft also deploys its "Windows Update" service that conflicts with many different software packages. This causes conflicts for simple tasks and can lead to user confusion, thus causing a system to possibly not be updated at all.

2.1.3 Eclipse

A robust and incredibly extensible application is the Eclipse development environment. Eclipse is an open source community whose projects are focused on providing a vendor-neutral open development platform and application frameworks for building software [3]. Several industry leaders formed Eclipse and still advocate it today. Companies like Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft, and Webgain all formed the initial Eclipse memberships and the list grows everyday [3].

Eclipse represents a programming framework that allows an incredible amount of extensible programming capabilities. Many of which are based within the Java programming language. From rich client applications (RCP), to dynamic code generation of various types, the options are endless as far as what can be implemented within Eclipse.

Eclipse is written in the Java programming language and comes with extensive plug-in construction toolkits available to any programmer who wishes to implement them. A plug-in is a

modular feature and/or tool that is downloaded via a drag and drop system and it is installed and accessed when Eclipse is first invoked. Eclipse searches its plug-in folders for any new additions that have occurred since the last startup and will automatically install any new plug-in that was copied into its plug-in directory. A great feature of Eclipse is that all plug-ins are fully extensible, even to other plug-ins. They can be extended upon and modified on the fly to incorporate new releases and downloadable updates via the Internet. Most of these concepts were the driving forces behind the creation of the middleware solution, but these concepts needed to drive one step further by reaching out to more complex design environments, Computer Aided Design environments.

```
<!-- Extension for drop down menu -->
   <extension
         point="org.eclipse.ui.actionSets">
      <actionSet
            id="Pegasus.actionSet01"
            label="Pegasus Action Set"
            visible="true">
         <menu
               id="Pegasus.menu01"
               label="Services">
            <separator name="Pegasus.separator01"/>
         </menu>
         <action
               class="pegasus.PegasusAction"
               enablesFor="10"
               id="Pegasus.action03"
               label="Process Computations"
               menubarPath="Pegasus.menu01/Pegasus.separator01"
               style="push"/>
         <action
               class="pegasus.PegasusAction"
               id="Pegasus.action01"
               label="Graphvis Conversion"
               menubarPath="Pegasus.menu01/Pegasus.separator01"
               style="push"/>
      </actionSet>
   </extension>
```

Figure 2. Code Block of Eclipse's Menu System

Figure 2 shows a code block of the extensibility within the Eclipse menu system. Through simple eXtensible Markup Language (XML) information contained within a configuration file,

the menus within Eclipse can easily be extended upon to include custom menus that perform specific functions contained within a custom plug-in. Figure 2 shows the extension used for the plug-in demonstration described within Chapter 6. The menu tag indicates a top-level menu item that would be listed next to the standard menu items such as File, Edit, View, Insert, etc. The action tag above represents options that can occur underneath the top-level menu item. This is equivalent to the New or Open options usually found under the File menu item. For the particular demonstration in Chapter 6, Graphvis Conversion is clickable and Process Computations is not. For further explanation, see Section 6.3.

2.2 NETWORK DATA TRANSMISSION

There are many different types of network data transmission protocols that are deployed and utilized throughout global networks today (such as the Internet). Many of them include methods of transmission control, error checking, transactional capabilities, and secure transport of data. Although these technologies and protocols are complete within their specific purposes, they are not complete solutions for a distributed design environment. They are merely individual components that need to be integrated together to form the complete middleware solution described within this thesis. A brief background in each technology used to formulate the system architecture will be presented in the next few subsections.

2.2.1 Java Messaging Service

The Java Messaging Service, or JMS, was developed by Sun Microsystems as an extension of the J2EE programming language. JMS provides a common way for Java programs to create, send, receive, and read messages among many entities. Furthermore, JMS is a set of interfaces and associated semantics that define how a JMS program, or client, accesses the facilities of an enterprise-messaging product (J2EE). Messages, as defined the JMS specifications, are asynchronous requests, reports or events that are consumed by enterprise applications [12].

Figure 3. Java Messaging Service – Messaging Mechanism

Figure 3 above displays the point-to-point (PTP) product or application that is built on the concept of message queues, senders, and receivers. Each message is addressed to a specific queue, and receiving clients extract messages from the queues established to hold their messages. Queues retain all messages sent to them until the messages are consumed or until the messages expire [12]. The receivers can fetch the message whether or not it was running when the client sent the message. The receiver also acknowledges the successful processing of a message [12].

JMS does not define an API for administering messaging products [4]. JMS also does not define a repository for storing message type definitions and it does not define a language for creating message type definitions [4]. JMS is a robust messaging service, but is not secure. Because of these limitations inherent within the standard JMS API, regular JMS cannot be used as the messaging system needed for the system architecture. A new JMS API that extends current JMS methods needs to be created to incorporate administration, security, and ease of use, while also integrating into existing JMS infrastructures.

2.2.2 Secure Java Messaging Service

Security enabled JMS integrates administration, security, and ease of use, while also integrating into existing JMS infrastructures. However, current secure JMS systems have only used Secure Socket Layer (SSL) for transport of messages and administering SSL requires knowledge of that particular protocol. No thorough secure JMS method has been developed that is programmed directly using API interfaces. This is due to JMS not specifying a security context or an API for

controlling message confidentiality and integrity [4]. JMS is controlled by a system administrator rather than implemented programmatically or by the J2EE server runtime [4]. So, the new Secure JMS API had to be developed to implement message confidentiality and integrity. This is where Secure JMS is applied.

2.2.3 TCP/IP

Transmission Control Protocol and Internet Protocol are the core protocols of the Inter-networks today. TCP and IP were developed by a Department of Defense research project to connect a number of different networks designed by different vendors into a network of networks (the "Internet") [7]. It provides basic services such as file transfer, electronic mail, and remote login across very large numbers of clients and server systems.

The IP component of TCP/IP provides routing from the enterprise network, to regional networks, and finally to the global Internet [7]. The DOD designed TCP/IP to be robust and automatically recover from any node or line failure. This design allows the construction of very large networks with less central management [7]. However, because of the automatic recovery, network problems can go undiagnosed and uncorrected for long periods of time [7].

Because TCP/IP provides only basic network functions, and leaves any security or authentication of clients and servers up to other technologies, it cannot be used alone within design environments. However, it can be and is used as the *transport* of the fundamental technologies purposed within this thesis. It is the underlying protocol that transports and provides communication between all functional components with the information infrastructure.

2.2.4 IP Security

IPSec is the IETF standard for Virtual Private Networks and is a technology specifically defined for the TCP/IP suite of protocols [13]. IPSec is usually tightly integrated with the TCP/IP stack, running in the operating system kernel instead as a separate software program [13]. IPSec is mostly used to setup VPNs between two entities across hostile networks.

IPSec consists of three major protocols: AH, ESP, and IKE. AH is a protocol that provides data origin authentication, data integrity, and replay protection. ESP is a protocol that

provides the same services as AH but also offers data privacy through the use of encryption. Lastly, IKE is a protocol that provides the all-important key-management function. The alternative to IKE is manual keying, which IPSec also supports [13].

Although IPSec can be beneficial when incorporating security within TCP/IP transmissions between two entities, the complexity of IPSec causes difficultly when configuring and managing it. This contributes to security weaknesses, of course, but it also means that IPSec VPNs require an expert to configure and maintain them. IPSec provides the secure concepts that the middleware solution utilizes within its system architecture.

2.2.5 HTTP / HTTPS

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, and hypermedia information systems [8]. It is a generic and stateless transmission protocol used for many tasks, through extension of its request methods, including distributing object management systems [8]. The HTTP protocol is widely used on the Internet today when serving clients with World Wide Web content, and can also be used for information transmission between clients and servers when both uploading and downloading data.

HTTPS is an extension of HTTP to allow for secured HTTP transmission over the Secure Socket Layer (SSL). SSL provides basic security services to various higher layer protocols and is mostly used to exchange higher layer data, such as HTTP traffic, as well as the data of various other SSL protocols. SSL also provides two services for each SSL connection, confidentiality and message authentication. Confidentiality utilizes a shared secret encryption key for encrypting the information sent between clients and servers. Message authentication uses a shared secret integrity key to create a Message Authentication Code for verifying whom the information is coming from.

HTTPS over SSL has many features and benefits that provide the mechanisms necessary to facilitate several technologies within the system architecture. The Secure File Sharing System utilizes HTTPS for uploading and downloading files through the use of web browsers, and SSL is utilized for establishing connections to the Gateway and Verification Servers.

2.3 APPLICATION FIREWALLS

When there is a need to transfer information between two networks such as corporate networks and external networks (e.g. the Internet), many companies wish to utilize hardware and software solutions to eliminate threats to their enterprises. Some solutions include proxy servers and hardware firewalls. These help protect internal intellectual property and help protect the enterprise from outside threats such as viruses, spyware, and improper network usage. Proxy servers can be implemented to prevent employees within an enterprise from conducting improper network usage while firewalls protect internal property and resources from outside threats.

A brief description of a few applications that perform these functions will be highlighted below. However, these applications and others that exist today do not offer solutions to support the information infrastructure this thesis defines; none offer service based protection and transaction verification and tracking. The infrastructure services found within Chapter 4 *do* support service based protection and transaction verification/tracking.

2.3.1 Squid Web Proxy Cache

Squid is a high-performance proxy caching server for web clients [21]. It supports the following protocols: File Transfer Protocol (FTP), a protocol for simple peer-to-peer file transfers; gopher, a distributed document search and retrieval network protocol designed for the Internet [22]; and Hypertext Transfer Protocol (HTTP) data objects used during standard web browsing sessions Unlike traditional caching software, Squid handles all requests in a single, non-blocking, I/O-driven process [21]. This is useful when large amounts of data are being transferred through the proxy server.

Squid supports Secure Socket Layers, extensive access controls, and full request logging. By using the lightweight Internet Cache Protocol, Squid caches can be arranged in a hierarchy or mesh for additional bandwidth savings [21]. Internet object caching is a way to store requested Internet objects (e.g. HTTP, FTP, and gopher protocols) on a system closer to the requesting site than to the source. Web browsers can then use the local Squid cache as a proxy HTTP server, reducing access time as well as bandwidth consumption [21].

2.3.2 Cisco PIX 525 – Hardware Firewall

The Cisco® PIX® 525 Security Appliance delivers a wealth of advanced security and networking services for medium-to-large enterprise networks, in a reliable, purpose-built appliance [23]. Some of its features include robust peer-to-peer and remote access virtual private network connectivity, flexible management solutions, protocol anomaly detection, and URL deobfuscation [23].

Although this device may seem to be the ideal solution, this still does not satisfy the requirements of the infrastructure services within the system architecture defined in this thesis. A method for service management and service protection is needed that integrates seamlessly within the design environment, and this device does not offer such solutions.

2.4 DISTRIBUTED FILE SYSTEMS

Much work has been performed on distributive systems including work in academia and industry. Those distributive systems include the AFS Distributive File System [17], Coda File System [25], Kiwi [18], and Microsoft's Distributive File System [6]. Each system has their strengths and weaknesses as described in the following subsections. The Secure File Sharing System introduced within this thesis utilizes the strengths of each of the following systems and eliminates the weaknesses each system possesses.

2.4.1 AFS Distributive File System

Since 1983, the AFS Distributive File System has been under constant development at Carnegie Mellon University. It is a distributed computing environment that is a location-transparent distributed file system. File system resources can be shared across both local and wide area networks, and the file system is transparent to the end user. However, there is no built in encryption for file storage. Also, AFS is UNIX based and will not work in a Windows environment without the use of third party clients or plug-ins.

2.4.2 Coda File System

Coda is a distributed file system developed at Carnegie Mellon University since 1987. It has its origin within the second version of AFS and has many features that are desirable for network file systems. Currently, Coda's features not found in other network file systems are its continued operation during partial network failures and its high performance through client side persistent caching. It has a security model for authentication, encryption, and access control while also allowing for server replication. It has good scalability and adapts to network bandwidth [25]. Even though this system entails several key features desired within a distributed file system, it does not have transactional capabilities for recording access to files within the system.

2.4.3 Kiwi

Kiwi is a Linux based system that uses HTTPS to implement a distributive file system over global networks. This file system allows for secure access to all files, regardless of what network the files reside in, by simply using a web browser. This eliminates the need for any burdening clients, firewalls, or network policies and also allows for scalability within the global infrastructure [18]. Again, this may seem like the ideal solution for the Secure File System, but Kiwi does not provide transactional capabilities to record all accesses to files. This is one of the core ideas that the middleware solution is built around.

2.4.4 Microsoft Distributive File System

Microsoft's Distributive File System has recently been renamed to Distributive File System Namespaces, or DFS Namespaces, in 2003. Although new features have been added to Microsoft's DFS, such as a new replication engine that replaces the old File Replication Service (FRS) and a reduction of bandwidth used when replicating new files, the same basic principles apply from DFS. Like AFS, you can share file system resources across both local and wide area networks, and the file system is transparent to the end user. The problems of DFS are still

apparent in the new DFS Namespace; there is no built in encryption for file storage, and it is Windows based only and will not work in a UNIX environment.

2.5 SECURE FILE SYSTEMS

Prior work on secure file systems has been completed to address certain security issues when the need to transfer and store large files was required. They include e-mail based infrastructures, web-based infrastructures, and client based infrastructures. Some already existing systems listed below include many of these infrastructures, and each one has supportive benefits that this thesis extracts and uses, along with systematic flaws that prevent them from being used independently as the Secure File Sharing System component of the system architecture.

2.5.1 CINDI Secure File System

CINDI Secure File System (CSFS) is based off of two secure systems: a Secure Database Management System (SGDBM) and a Secure File System that uses SGDBM to store passwords for encrypted files. Combining the implementation of these two systems, CSFS allows for data and metadata to be stored in an encrypted form. The command API is similar to other basic Linux commands, but a GUI application is necessary for those secure file commands to be issued [19].

2.5.2 Mailonator

Mailonator [14] is a solution for sending large files via an e-mail based infrastructure. Some features and benefits of the system are that it integrates into already existing e-mail clients and servers, it stores attachments via a secure URL, it is transparent to end-users, and it provides usage statistics to track how the system is being utilized. Some of its problems however make it impractical for a true secure file sharing system. First, it is only usable via e-mail. E-mail is inherently insecure, so the files would not be transferred as securely as desired within this new

system. Secondly, the files are not encrypted when they are stored on the server or servers. Access to them may be encrypted, but not the files themselves. If the server were to be compromised, full access to the unencrypted files would be possible.

2.5.3 Xythos

Xythos [15] is a solution for web-based file sharing and storage. A feature of it is that it is 100% compatible with WebDAV applications. It also includes features such as bandwidth controls, enterprise level security, platform-independence, and 100% Java-based code. However, even this system has some problems. One problem of Xythos is that it is web-based only and can only be used via web-based applications. No other scripting or low-level programs will work with it. Another problem is that the files are not encrypted or secured on the file system; only the transmission of the files is secure. This opens the same hole as with the e-mail system; if the server became compromised, full access to the files would be possible.

2.5.4 ZipLip Large File Transfer

ZipLip Large File Transfer (ZL LFT) [16] is another secure file system that is used over vast public networks. Some features and benefits include secure methods for remote file access and file sharing, policy control for access to the files, audit trail capabilities, scalable file repositories (with optional encryption), redundancy over multiple servers, and it is 100% Java-based. This system may look like the all-inclusive package to use, but it does have some downfalls. It is a closed source program with no community support and it uses its own proprietary encryption. The trust-worthiness of the encryption is not very high, and some flaws could be found in it.

2.6 CONCLUSIONS

All of these prior works briefly describe what work with Industry and academia has been done to facilitate the creation of the fundamental technologies that are necessary to make the middleware

solution a reality. As was mentioned before, the new solution improves upon and integrates the features and benefits of all the prior works into the comprehensive middleware solution.

The development and implementation of the fundamental technologies, which make up the system architecture, complete the middleware solution for design environments. During formation of the system architecture, each component was developed and implemented on an individual basis. This included testing and evaluating each system component separately before integrating them together. After each technology was completed, an integration effort took place, bringing each component together to form the completed system architecture. The following chapters describe the details of each technology developed to formulate the system architecture.

3.0 ENABLING SECURE AND TRACEABLE COMMUNICATIONS

When transmitting information between the components of the system architecture, the communications between these components must be secure, traceable, and allow for non-repudiation. This "information highway" must also be able to encapsulate and transmit the different data types utilized within the architecture such as transactional information and job requests and results. The idea of objects sent as messages appealed to the specification requirements of the system architecture. Thus, messaging services that could send data objects were researched to find which service best satisfied those requirements.

After review of the services available, the Java Messaging Service was found to be the best fit for the requirements specified. JMS is a robust and integrated messaging solution that provides several key features that are utilized for secure communications within the system architecture. A more detailed look at the various features, downfalls, and improvements made on JMS occurs in Section 3.1.

To implement security, traceability, and non-repudiation, all information shared between components of the system architecture must be recorded. These transactions are essential to maintaining integrity between enterprises and service providers. A third party verification mechanism is necessary to record these transactions and this mechanism must be available to anyone wishing to use it. These necessities come together to create the verification services introduced in Section 3.2.

3.1 SECURE JAVA MESSAGING SERVICE

Messages sent within the system architecture of the middleware solution need to maintain security, traceability, and non-reputability. Without these necessary requirements, messages sent

between enterprises and service providers would be vulnerable to malicious attacks from the networks they are transferred over. At the same time, messages need the capability to transfer all data objects and not just textual information. This is why a "secure" messaging service had to be developed that incorporates security, traceability, and non-reputability, and has the ability to transfer all types of data objects.

A major messaging service widely used within Industry today is the Java Messaging Service. JMS utilizes the java enterprise environment and has many features that would satisfy the requirements of this architecture. JMS has built-in persistent messaging capabilities which are incredibly useful when JMS messages are sent to unavailable receiving servers. JMS will continually try to send the messages until a specified time has elapsed, or the servers came back online and were able to receive the messages. JMS also has transactional capabilities that include commit and rollback functions.

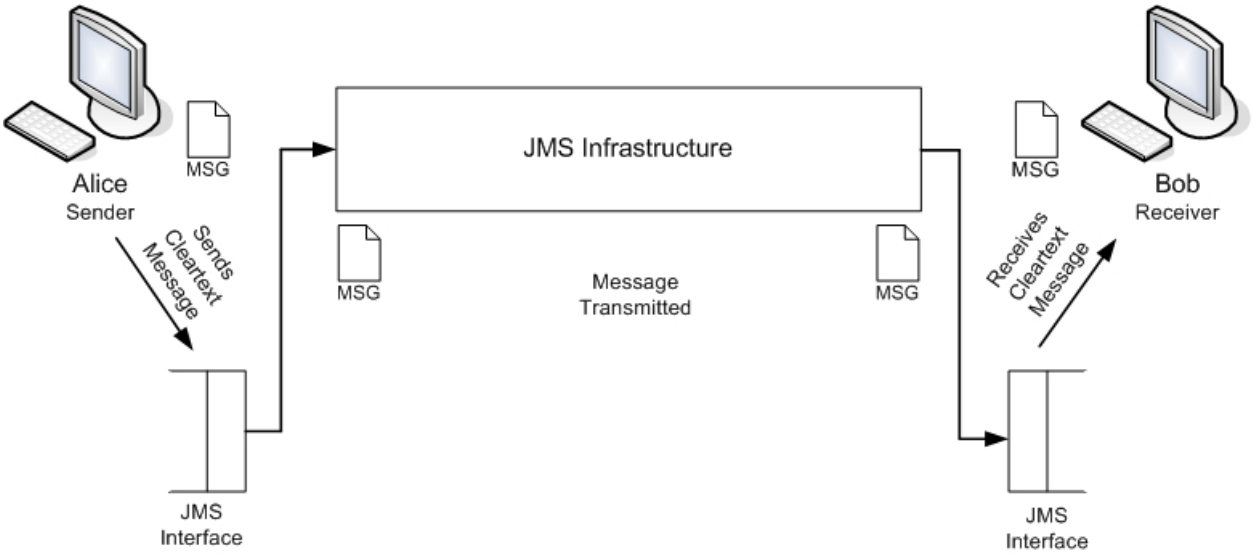


Figure 4. Sending a Standard Message Over the Java Messaging Service

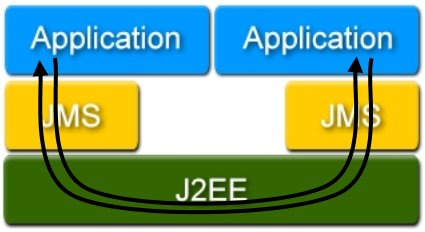


Figure 5. Regular JMS Data Flow

Figures 4 and 5 display how the java messaging service functions within a messaging environment. In Figure 4, Alice sends a message from her client to the JMS interface within her network. This interface then packages the message into a JMS message and sends it over the JMS infrastructure. The message is then received by Bob's JMS interface and the original message Alice sent is extracted from the JMS message. Finally, the message is sent to Bob's client and displayed on his screen. The data flow through these technologies is shown in Figure 5.

As displayed in the example above, several problems present themselves when transferring a message between two entities; standard JMS is missing key features that are necessary to meet the full requirements of the system architecture. One problem is that JMS does not support built in encryption standards. Messages sent over the JMS infrastructure are unsecured and viewable via any network monitoring software.

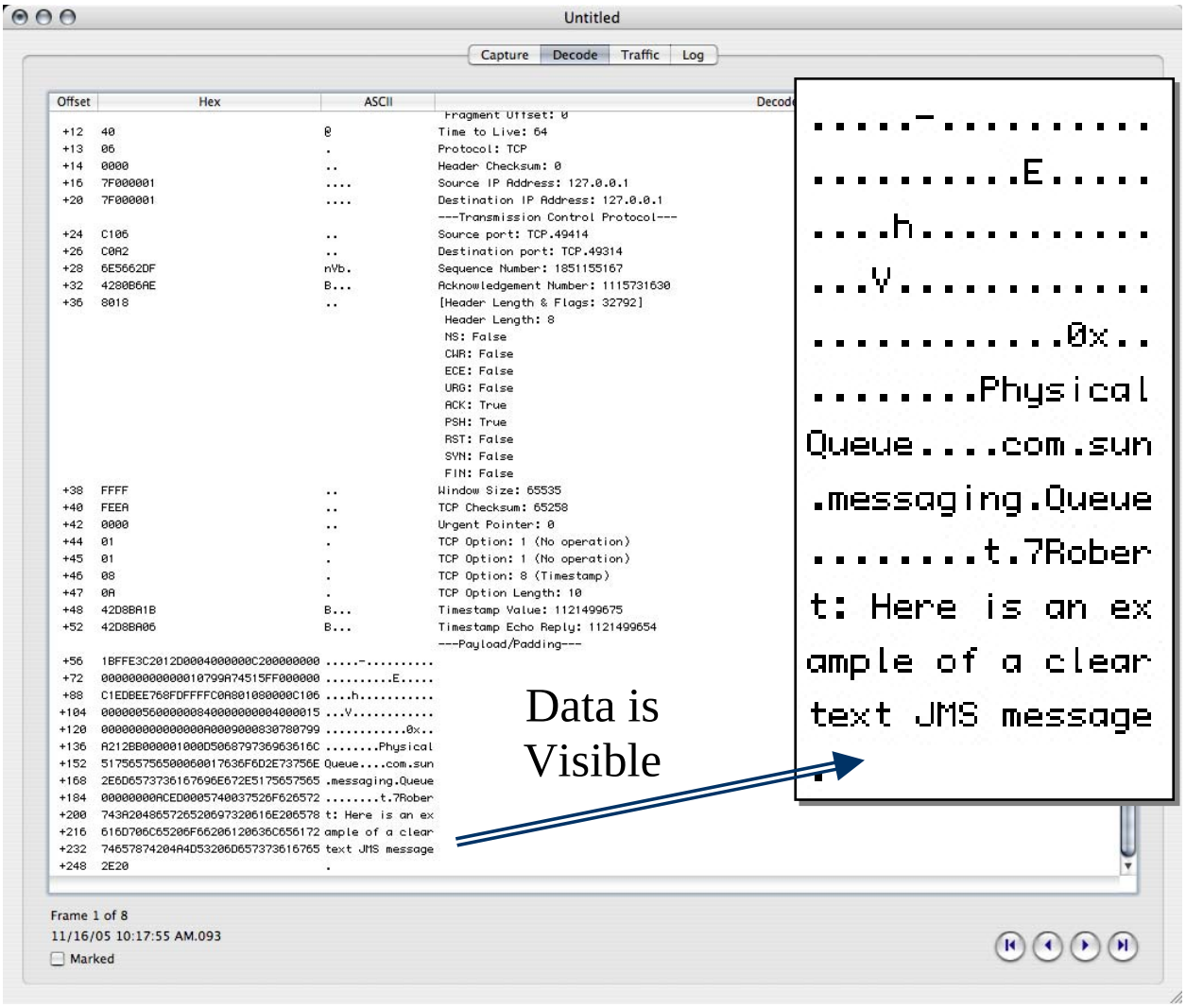


Figure 6. Display of an Unencrypted Java Message Service Message

Figure 6 shows the JMS text message that was sent from Alice to Bob in Figure 4. Clearly, anyone monitoring network traffic over the infrastructure can easily view the message. If the message contained sensitive information such as intellectual property, then that intellectual property is now exposed to the outside world without authorization. Also, there was no storage of transactional information for traceability and non-repudiation, such as demographic information (e.g. user information, IP address information, and company information), META information, keys, and hashes. How can this message be traced and non-reputable without

verification services? Secure JMS utilizes the verification services introduced in Section 3.2 for traceability and non-repudiation.

Secure Socket Layer has been the only technology used for Secure JMS systems today. SSL is used to transport the messages over the network layer, with JMS messages being encapsulated within SSL packets. An API construct method of sending secured JMS messages does not exist, partially due to JMS not specifying a security context or API for controlling message confidentiality and integrity [12]. Therefore, a new API had to be developed to implement message confidentiality and integrity for the communications between client enterprises and service providers within the system architecture. This is where Secure JMS applies its security and transactional technologies.

3.1.1 Design of Secure Java Messaging Service

Secure JMS is one of the core technologies fundamental to the development of the new middleware solution. It is derived out of the necessity to send large messages and files through the JMS infrastructure, encrypted, traceable, and non-reputable. Secure JMS is used for the communications between client enterprises and service providers within the system, instead of using Java communication sockets. This creates a method of information transfer and includes the added features and benefits of JMS as mentioned above. However, Secure JMS extends the current JMS technology as a result of standard JMS not supporting encryption standards or added transactional capabilities. Secure JMS encrypts the messages sent over its infrastructure so that they cannot be viewed, not even by network monitoring software, and utilizes the verification services as described in Section 3.2. The data flow of Secure JMS is like the data flow in Figure 5. Figure 7 shows how Secure JMS plugs-in to the current JMS infrastructure to allow for encrypted messages.

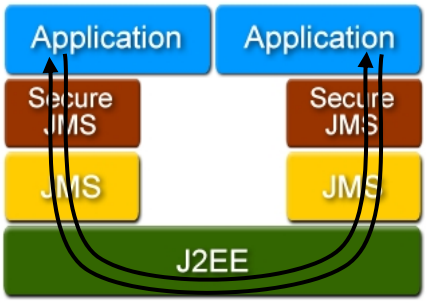


Figure 7. Secure JMS Data Flow

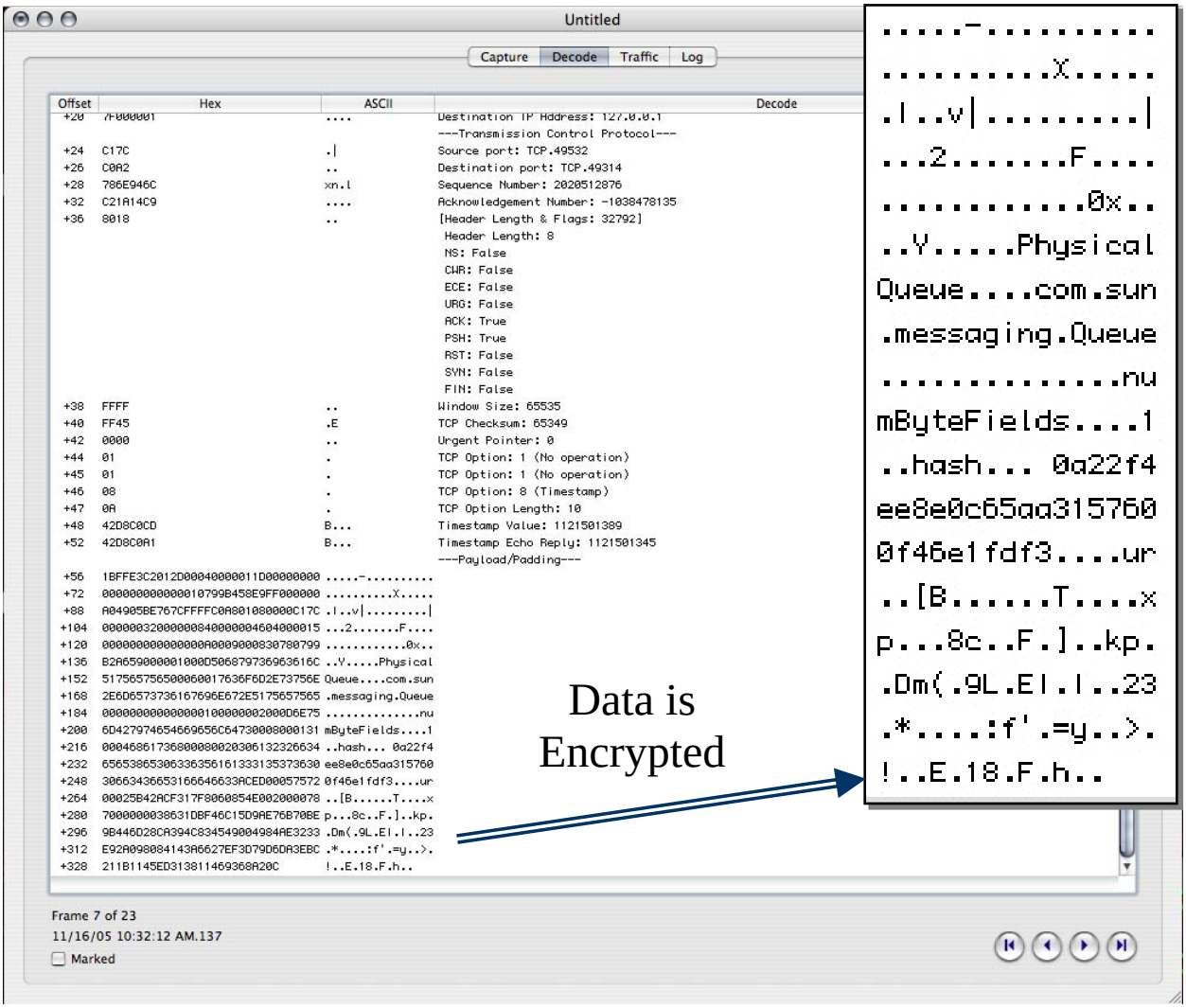


Figure 8. Display of a Secure Java Messaging Service Message

Figure 8 shows an example of a Secure JMS message being sent from one person to another, but this time, encrypted. As is shown, the message is completely illegible to anyone monitoring the network traffic. This encrypted message can contain text messages or data objects. Any type of information sent over the secured JMS infrastructure will be encrypted as shown.

A major benefit of Secure JMS is that it is modular, allowing for seamless integration into already existing JMS infrastructures. This is beneficial to any enterprise that already uses JMS in their current design environments. Because Secure JMS was created and implemented with existing JMS frameworks in mind, integration issues are alleviated for the programmer. Current JMS implementations can be utilized and minimal code modification is required; only one line of code modification is needed as shown in Figure 9.

```
// Standard JMS
StreamMessage.writeString(CurrentMessage);producer.send(StreamMessage);
// Secure JMS
sJMS.send(CurrentMessage, producer, StreamMessage);
```

Figure 9. Code Modification Example from Standard JMS to Secure JMS

A programmer simply replaces their current JMS code with a modified function call to access the security and transactional features of Secure JMS; the message is sent through the Secure JMS module, encrypted, recorded, and secured.

Secure JMS utilizes the Advanced Encryption Standard [9] for its encryption algorithm. The messages are encrypted and sent using a random key that is generated for each message as it is sent. This was derived from the concepts behind the Vernom Cipher, which is a cipher that is only used once and then destroyed, and it helps to prevent key-guessing attacks against the encrypted JMS message. An MD5 [20] hash value of the encrypted message is also used to verify that the receiver of the message received it correctly and in tact. This also verifies that the message has not been altered during transmission, much like a digital fingerprint.

Secure JMS attaches META information to its messages and stores the information in the Verification Server, which utilizes the verification services as described in Section 3.2. This solves the traceability and non-repudiation issues mentioned throughout this thesis. The META

information describes who the sender of the message was, who the receiver of the message was, their respective IP addresses, the encryption key of the message, the hash of the message, and any other data associated with the message (files, images, etc).

3.1.2 Requirements

Secure JMS has very few requirements and most of the essential requirements are already met within a java enterprise infrastructure. Secure JMS only needs a current JMS infrastructure within an enterprise to be set up and administered for complete functionality to occur. Also, a method of transaction storage must be available as well (e.g. verification services). Once these requirements are satisfied, Secure JMS's functionality will be available to any service wishing to utilize it.

Secure JMS does require the Java Runtime Environment in order to operate. It can run on any platform that supports Java and enterprise server applications, such as J2EE, including the Mac OS/X, Microsoft Windows, and Linux operating systems.

3.1.3 Performance Results

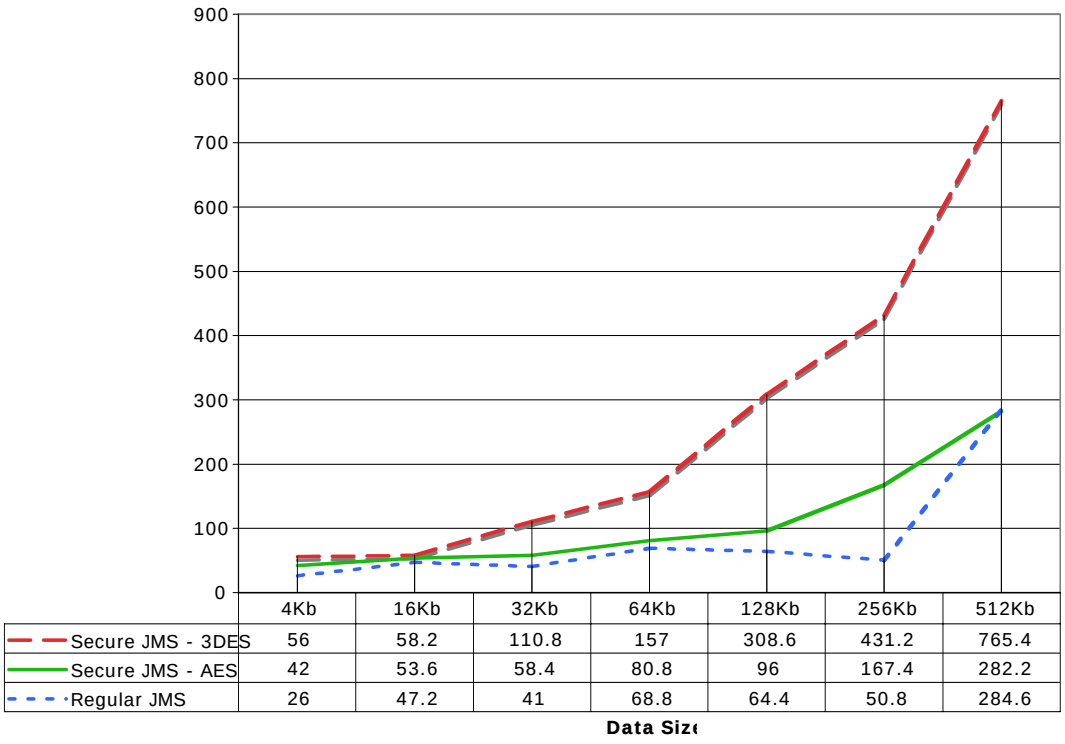
When the secured JMS infrastructure was first being developed, it was thought that encrypting messages and sending them would take more time than just sending clear text messages alone. The 3DES (Triple Data Encryption Standard) encryption algorithm was the first one used during development, and implemented in the first tests of Secure JMS. Transmission time increased about twenty-five percent when using 3DES as the encryption algorithm, and this was found to be satisfactory for small files. However, when large files were transferred, the difference proved to be substantial. During a discussion about the encryption algorithms, another algorithm was mentioned as being much faster.

AES (Advanced Encryption Standard) [9] is a newer encryption algorithm designed and published by the Federal Information Processing Standards Publications (FIPS PUBS) and issued by the National Institute of Standards and Technology (NIST). AES is a symmetric block cipher that can process data blocks of 128 bits and use many cipher key lengths such as 128, 192, and

256 bits [9]. This was found to be the best encryption algorithm to suit the needs of Secure JMS, both in efficiency and implementation.

Secure JMS was then implemented with AES, with a 128 bit cipher key length, and tested again with the same files. When using AES, the difference between sending regular JMS messages and encrypted JMS messages reduced significantly, when compared with the 3DES algorithm. This reduction was enough to negate a difference when transmitting the messages over large networks such as the Internet.

Figures 10 and 11 compare laboratory test results of sending small and large JMS messages for regular JMS messages, 3DES encrypted messages, and AES encrypted messages. These experiments were performed on an Apple G4 Laptop running at 1 GHz with 1 GB of ram. Each data point represents the average of five identical runs for the specified file size.



	4Kb	16Kb	32Kb	64Kb	128Kb	256Kb	512Kb
Secure JMS - 3DES	56	58.2	110.8	157	308.6	431.2	765.4
Secure JMS - AES	42	53.6	58.4	80.8	96	167.4	282.2
Regular JMS	26	47.2	41	68.8	64.4	50.8	284.6

Figure 10. Sending Small Java Messaging Service Messages

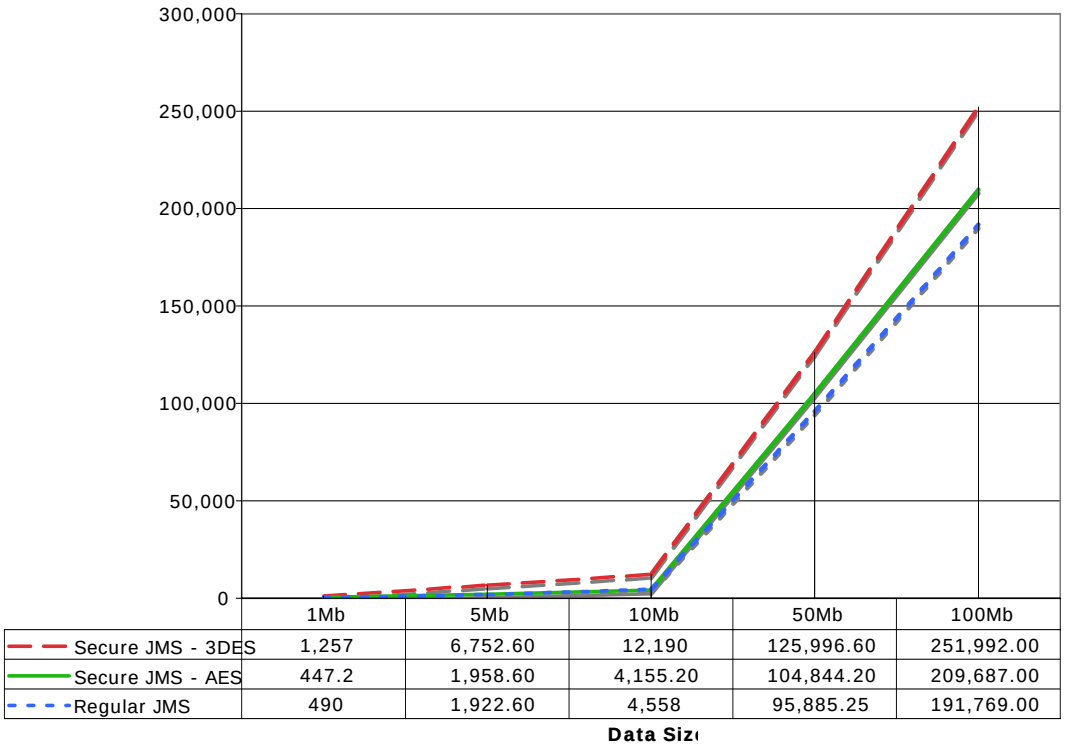


Figure 11. Sending Large Java Messaging Service Messages

When sending a large file through a standard JMS infrastructure with a size of one hundred megabytes, an unencrypted JMS message has an approximate throughput of five hundred and forty seven kilobytes per second. When sending the same file using Secure JMS with AES encryption enabled, the message has an approximate throughput of five hundred kilobytes per second through the same JMS infrastructure. This forty-seven kilobyte difference is negligible when compared to the standard Internet up-stream connections that most enterprises have within their corporate networks. Also, it is worth the additional eighteen seconds of transfer time to ensure security, traceability, and non-repudiation for JMS messages sent between enterprises and service providers. A detailed look into how the verification services utilize Secure JMS will be described within the next section, Section 3.2

3.2 VERIFICATION SERVICES

Verification services are part of the fundamental technologies developed for the system architecture. This technology is necessary because it provides a method of verifying operations within the infrastructure by storing transactional and meta-data (data that describes other data) information into a database. The Verification Server represents the verification services utilized within the middleware solution this thesis defines. This server plays a key role when establishing connections to the outside world from within an enterprise; it will record all transactions that occur within the design environment. This includes all communications to and from the Service Provider, such as Secure JMS message information, service request information, and any other communication that requires traceability and non-repudiation. The Verification Server shares trust among enterprises and service providers in order to assure proper traceability and non-repudiation.

Secure JMS was the original method of data transmission between entities using the verification services and the Verification Server. However, this method was unsuccessful because there was no way to transport the encryption keys of the encrypted JMS messages without putting the keys directly in the JMS header. The JMS header is not encrypted and thus, viewable by anyone on the network. Instead, a different approach was taken so that keys could be transmitted without being viewed. The transmission of data between entities using the verification services and the Verification Server is carried out over Java SSL sockets. This allows for encrypted communication, and allows the encryption keys and hashes to be sent securely. Also, before usage of the verification services, each entity's identity is assumed through previous authentication methods. The current design of the verification services does not incorporate authentication, but plans for providing authentication methods are set for future work.

3.2.1 Requirements

Verification services are mechanisms that store transactional information for data sent and received within the design environment. These operations facilitate traceability and non-repudiation. The verification services introduced within this thesis were designed and

implemented through the use of the Verification Server. The Verification Server utilized within the system architecture requires the Java Runtime Environment to run the actual server. It also requires a database application for the back-end storage capabilities and setup/administration of that database application. For this particular implementation, the MySQL database application [24] was used. Setup and administration of the MySQL server used will be explained in Section 3.2.5. The Verification Server also requires an open network port to allow for incoming and outgoing requests from anyone wishing to utilize its services of verification. For this thesis, the entities that request verification services are the enterprise and service provider.

The Verification Server interacts with client enterprises and service providers in the following ways. Client enterprises send all transactional information relating to jobs to the Verification Server for storage including encryption keys, hashes, the service requested, and the service provider that performs the service requested. These processes are described in detail in Section 3.2.3. Service Providers use the Verification Server to send receipt confirmation of jobs, while also sending and receiving encryption keys of Secure JMS messages.

3.2.2 Processing Secure and Traceable Messages

The implementation, requirements, and performance of a secured and traceable message have been explored in the previous sections. Now, details of how a secured message is made and processed as it goes through the JMS infrastructure will be described. Figure 12 below represents the top-level overview of the secured and traceable system. It also displays a visual representation of each major step a message takes as it is sent from a sender to a receiver. A messaging client is displayed as a simple way of sending text messages to another client. Alice – the sender – can represent a client enterprise sending some files to a service provider, Bob – the receiver. The communications between each user's Secure JMS module and the Verification Server are performed over SSL pipes so that the encryption keys can be sent for storage. Oscar represents a malicious user trying to view the message on the network.

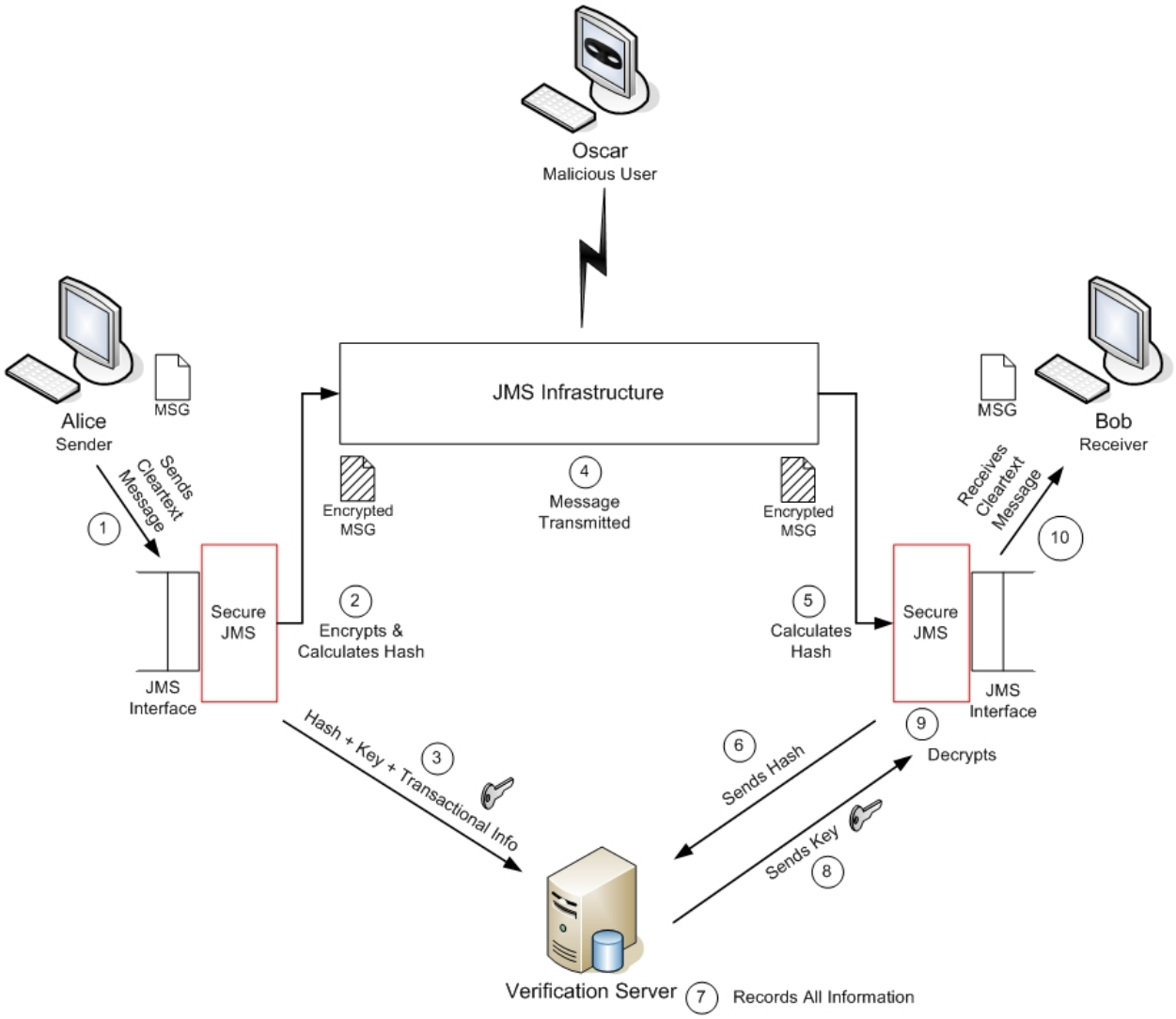


Figure 12. Secure Java Messaging Service Overview

- Step 1 – Alice types a message in her software and clicks “send”. The message is sent in the clear to her regular JMS API through a software function call.
- Step 2 – This JMS API passes the function call and text to the Secure JMS module. Secure JMS then encrypts the message and calculates a hash value while the encryption process is occurring. This hash is utilized as a digital fingerprint to verify that the message has not been changed during transit.
- Step 3 – After the encryption process is completed, the encryption key and hash values are sent to the Verification Server for storage within its database. This begins a transaction for the message, storing information such as Alice’s username, IP address, date sent, and

message ID. When Bob is notified that a message is ready to be received, he can process the message using the transactional information that has been stored in the Verification Server.

- Step 4 – The encrypted message is transmitted over the enterprise JMS infrastructure. Since it is encrypted, it will not be viewable on the network by Oscar (a malicious user), or by any other network monitoring software as shown in Figure 8.
- Step 5 – The message is received by Bob's Secure JMS module. The message hash is calculated as the encrypted data is received.
- Step 6 – When the hash calculation is complete, the hash value is sent to the Verification Server with a request for the encryption key. This effectively ends the transaction in progress for the message received.
- Step 7 – The Verification Server records everything as it occurs during the process of sending and receiving messages, including all transactional information. In addition to Alice's information that is already stored, the server stores Bob's username, IP address, and date received, according to the message ID received and sent by Bob.
- Step 8 – The Verification Server verifies that the hash matches the one on file and sends the encryption key back to Bob. The server knows that Bob is indeed Bob through an authentication process that occurred earlier when Bob first connected to the Verification Server.
- Step 9 – Bob's Secure JMS module decrypts the message using the received encryption key.
- Step 10 – Secure JMS sends the message in clear text to Bob's client via a software function call. This completes the process of sending and receiving a secured message over the system architecture.

Because messages are now secured and traceable, all three requirements of security, traceability, and non-repudiation have been satisfied within the system architecture. Enterprises and service providers can now exchange messages with confidence that those messages will be sent and received with automatic integrity.

3.2.3 Processing of Incoming and Outgoing Information

Incoming information to the Verification Server includes Secure JMS transactional information and job related information. Outgoing information includes transaction IDs and encryption keys. These transmissions occur over the SSL connections established with the enterprise's Gateway Server and the Server Provider.

The Verification Server processes all incoming information in the following manner. The Verification Server listens on its connection port (5525) for any new incoming connections. Once a connection is established, the connecting party transmits its purpose for connecting, and the Verification Server responds by performing one of the following actions:

1) Sending the requested information, depending on which entity connected.
2) Accessing database methods, depending on what the transmitting party requests; then returns any values necessary.

The Gateway Server and Service Provider can both access the database methods within the Verification Server. The Gateway Server and Service Provider can send requests to add transactions for Secure JMS messages including encryption keys, and update transactions already in progress. The Gateway Server can also send requests to add jobs, and update jobs already in progress. The Verification Server will return a transaction ID whenever a new transaction is created, either by the Gateway Server or the Service Provider. The returned transaction ID is used by the receiver to update the transaction whenever they receive the message successfully.

3.2.4 Transaction Security

Through the use of the verification services, transaction security is established. Transaction security protects the several components of the system architecture from each other in the following ways. The hash exchange mechanism protects the service, and prevents the senders (enterprises) and receivers (service providers) from being dishonest to each other. If the hash value of a received message is calculated and sent to the Verification Server for retrieval of the encryption key, then denial of ever receiving that message cannot occur. At the same time, if a

hash value is not received for a message claiming to have been sent, then that message did not pass through the information infrastructure and denial of ever sending that message cannot occur.

Transaction security also protects against snooping verification servers. The Verification Server only sees the hash and key value of a message sent over the infrastructure. It does not see actual message data. Therefore, the company that houses the verification services does not have access to sensitive information intended for authorized entities only.

The verification server does have some weaknesses. One of them is collusion. The verification services implemented within this thesis do not protect against the client and verification server from deceiving the service provider, or from the service provider and verification server from deceiving the client. Another weakness is database security. If the database within a verification server is breached, then information pertaining to messages is exposed and the mechanisms established to protect the messages would not longer be active.

A solution to these weaknesses would be to implement multiple verification servers. This would allow for K-1 collusions or database breaches to occur for every K verification server utilized. Ideally, only one trusted verification server is needed, provided that none of the above collusions or database breaches occurs.

3.2.5 Verification Service Database Structure

As was mentioned in Section 3.2.1, MySQL was the database application used for the Verification Server. MySQL version 4.0.20 was installed and set up in a Linux operating system environment for the demonstration found in Chapter 6.

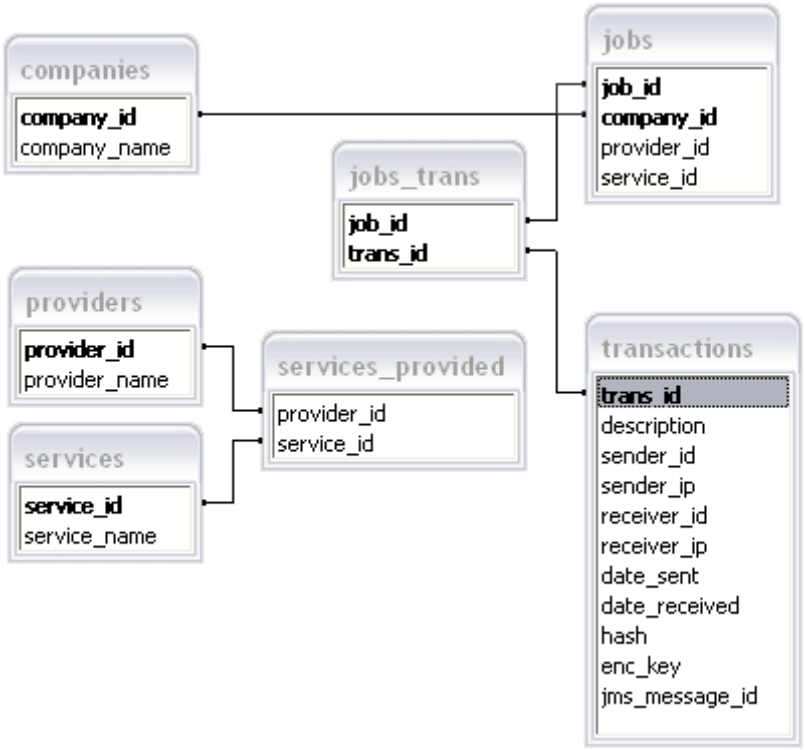


Figure 13. Entity Relationship Diagram of Table Relationships

Figure 13 displays an entity relationship diagram representing tables 1 through 8 below, and the figure is a visual representation of each table description that follows the table listings. Figure 13 shows the relationships between the various tables and data fields within the database. The bold field names within a table represent the primary key of that table. Those primary keys relate to other fields as shown by the drawn lines, connecting them to other data fields within the other tables. The following tables describe the database setup necessary to allow the Verification Server to function.

Table 1. Database Information Table

Username	Password	Database Name	Database Port
pegasus	peDesign	pegasus	3306

Table 2. `companies` Table

companies					
Field	**Type**	**Null**	**Key**	**Default**	**Extra**
company_id company_name	int(11) varchar(64)	 YES	PRI	0 NULL	

Table 3. `providers` Table

providers					
Field	**Type**	**Null**	**Key**	**Default**	**Extra**
provider_id provider_name	int(11) varchar(64)	 YES	PRI	0 NULL	

Table 4. `services` Table

services					
Field	**Type**	**Null**	**Key**	**Default**	**Extra**
service_id service_name	int(11) varchar(64)	 YES	PRI	0 NULL	

Table 5. `services_provided` Table

services_provided					
Field	**Type**	**Null**	**Key**	**Default**	**Extra**
provider_id service_id	int(11) int(11)	YES YES		NULL NULL	

Table 6. jobs Table

jobs					
Field	**Type**	**Null**	**Key**	**Default**	**Extra**
job_id	int(11)		PRI	0	
company_id	int(11)		PRI	0	
provider_id	Int(11)	YES		NULL	
service_id	Int(11)	YES		NULL	

Table 7. transactions Table

transactions					
Field	**Type**	**Null**	**Key**	**Default**	**Extra**
trans_id	int(11)		PRI	NULL	auto_increment
description	varchar(255)	YES		NULL	
sender_id	varchar(32)	YES		NULL	
sender_ip	varchar(16)	YES		NULL	
receiver_id	varchar(32)	YES		NULL	
receiver_ip	varchar(16)	YES		NULL	
date_sent	datetime	YES		NULL	
date_received	datetime	YES		NULL	
hash	varchar(32)	YES		NULL	
enc_key	varchar(32)	YES		NULL	
jms_message_id	varchar(64)	YES		NULL	

Table 8. jobs_trans Table

jobs_trans					
Field	**Type**	**Null**	**Key**	**Default**	**Extra**
job_id	int(11)		PRI	0	
trans_name	int(11)		PRI	0	

Table 1 describes the database information necessary to allow the Verification Server to function properly. A database with the name “pegasus” needs to be setup within MySQL, along with a username of “pegasus” and a password of “peDesign”. Also, the database must be setup to accept socket connections on the local port 3306 (default for MySQL), so the Verification Server can establish a JDBC connection with MySQL.

Table 2 describes the `companies` table. This table stores a company's identification number and name within the system. The primary key is `company_id`, so there cannot be two companies with the same ID number.

Table 3 describes the `providers` table. This table stores a service provider's identification number and name within the system. The primary key is `provider_id`, so there cannot be two service providers with the same ID number.

Table 4 describes the `services` table. This table stores a service's identification number and name within the system. The primary key is `service_id`, so there cannot be two services with the same ID number.

Table 5 describes the `services_provided` table. This table stores which services are provided by what providers. There are no primary keys in this table because service providers may offer the same type of services as other service providers and vice versa.

Table 6 describes the `jobs` table. This table stores which enterprises have a current job with what service providers, and what service that provider is providing. The `job_id` is assigned by the Gateway Server from the enterprise that submitted the job. The combined primary keys are `job_id` and `company_id`. They combine to make up the primary key, so there cannot be the same job from the same company submitted twice, but there can be multiple duplicate job's from different company's submitted at the same time.

Table 7 describes the `transactions` table. This is by far the most important table within the database because it tracks every step a job takes from being submitted, processed, and then returned. The `trans_id` is an auto-incrementing primary key, meaning that each new transaction is assigned a new `trans_id` by the Verification Server that is one more than the previous `trans_id` in the table. The `sender_id`, `sender_ip`, `date_sent`, `hash`, and `enc_key` fields are filled in by the sender of a JMS message when a job is submitted or finished. Then, the receiver fills in the `receiver_id`, `receiver_ip`, and `date_received` when it has successfully received the message. An explanation on how it looks up the `trans_id` information from the message is explained in Section 3.2.3.

Table 8 describes the `jobs_trans` table. This table relates job information to transactional information. In the usual case, each send or receive will consist of at least two transactions. One to signify the sent action and one to signify the receive action. The `job_id` and `trans_id` are combined to create the primary key. This is so there cannot be an entry in the

database with the same transaction ID and the same job ID. On the other hand, a job may have several different transaction IDs, such as the usual case of two. This table is used to look up which transaction is currently awaiting receipt confirmation from the receiver. When the receiver of a message parses the `job_id` from the message received, it sends it to the Verification Server, along with its information, so that the Verification Server can update the appropriate transaction with the receiver's information.

3.3 CONCLUSIONS

The information infrastructure within the middleware solution is now secured, traceable, and non-reputable through the use of Secure JMS and verification services. Secure JMS is necessary because it provides high levels of encryption while also interacting with the verification services for storage of transactional information. Verification Services are necessary to store information for traceability and non-repudiation. Enterprises can now deploy this critical information infrastructure within their design environments to enable distributed and robust design environments. The information infrastructure can now be utilized by the infrastructure services listed in the next chapter to formulate the complete system architecture.

4.0 INFRASTRUCTURE SERVICES

Enterprises usually contain one or more entities within their corporate networks that provide access to the Internet and remote facilities, and offer a central point of network contact to any other external entity. They block out viruses, spyware, and other malicious software from infiltrating the corporate networks. But how are they controlled? How is the enterprise's intellectual property protected, even from malicious users within the network? This information cannot be freely accessible to outside sources without proper authorization and traceability.

Service oriented architectures require mechanisms that provide services for those architectures. Since this new design environment extensively utilizes services, services can replace the standard software features most design software packages employ today. Services provide collaborative, distributed design environments, such as what is defined in this thesis. Ultimately, services allow any number of remote features to be accessed on the fly, and they can be shared among entities within an enterprise.

Previous methods of network protection consisted of hardware implementations through network segregation such as virtual area networks and through the use of firewalls, all to block incoming network traffic and control network traffic leaving the enterprise. When certain network enabled services are desired within the enterprise, the system administrator must be notified to punch a "hole" or throughway so that these services can function properly. This can sometimes be cumbersome and take many days to finally get the appropriate service authorized. It also defeats the reason the security was set up in the first place. What is needed within the design environment is an all inclusive gateway server solution that provides a method of central control and protection.

The Gateway Server and Service Provider provide the infrastructure services needed to facilitate the secured middleware solution. These architectural components utilize the information infrastructure for secured and traceable communications as described in Chapter 3.

A detailed look at these system components will be discussed within the next two subsections, Section 4.1 and Section 4.2.

4.1 GATEWAY SERVER

The Gateway Server is another fundamental technology created for the new middleware solution. It acts as a proxy server and is necessary for service oriented architectures. It is a solution that enables and controls the enterprise's network and information from a single point of control, allowing easy management among external services and providers, along with network protection. Storage of contractual agreements between the enterprise and service providers is also necessary for accounting purposes. Enterprise employees require knowledge of which services they are allowed to access via management approval.

To attain this knowledge, the Gateway Server provides a directory of services available to the enterprise. It can be thought of as a service or software firewall, preventing clients from accessing unauthorized services and keeping network data in check. The Gateway Server is a key technology within the system architecture because it centralizes all points of contact to the outside world, and can be easily managed by management within an enterprise. Client enterprises can use the Gateway Server as a directory server to retrieve a list of available services authorized by management, so that their designers can access the services from within their design programs. This occurs when the design software is first invoked.

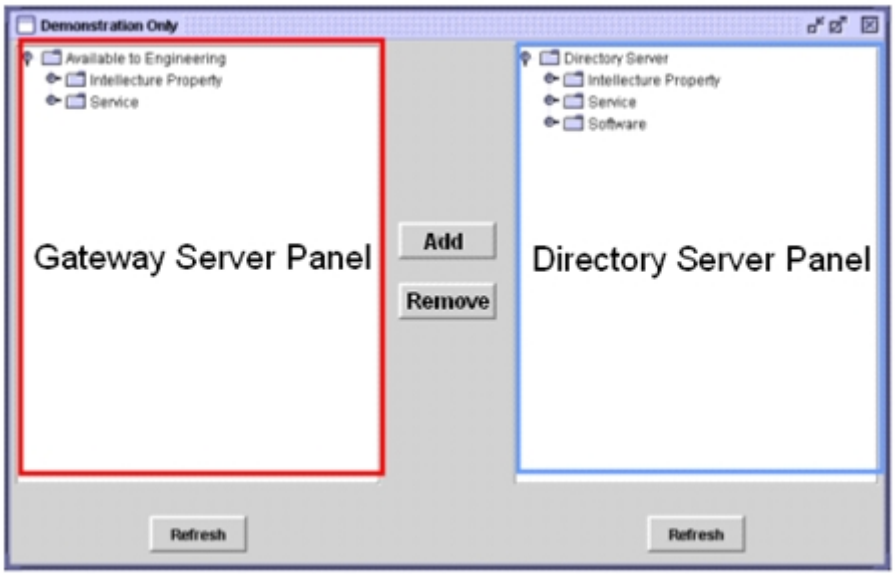


Figure 14. Management Control of Services

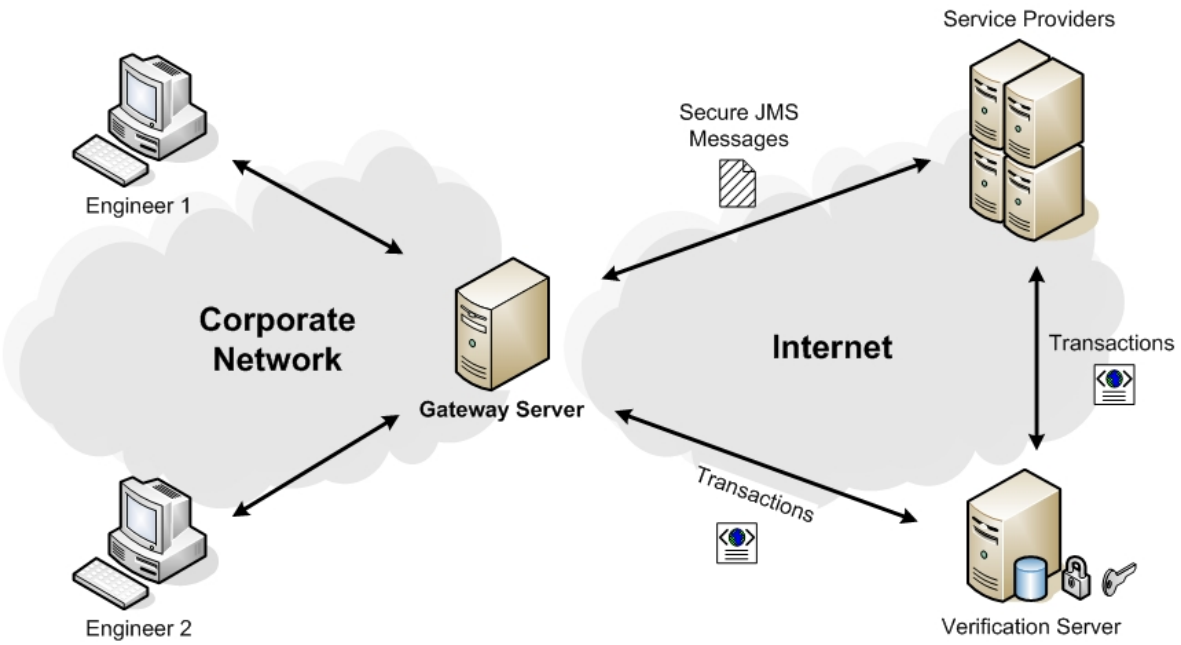


Figure 15. Management's Viewpoint of System Architecture

Figure 14 displays how a Management Control GUI (Graphic User Interface) might look. The GUI displays services in the left and right panel of the window. On the left panel are the services available to engineers within an enterprise. On the right panel are services listed within the

directory server portion of the Gateway Server. Management would simply highlight the service or services they want available to the engineers, and click the "Add" button. They can also remove available services if contracts expire, or they no longer want a service to be available to the engineers. Through this simple service management, services can be added or removed with little effort to upgrade existing services, install newer services, and remove old, deprecated services.

Figure 15 displays management's viewpoint within the design environment. Management is only concerned with what services are authorized to their enterprises from the different service providers. Management is also concerned about what contracts they have with the service providers as well. They see the Verification Server as a method of tracking the services that are accessed from the corporate network. At the same time, the Service Provider utilizes the Verification Server to verify that only authorized enterprises access their services.

4.1.1 Design of the Gateway Server

The Gateway Server provides methods of transmitting and receiving JMS messages to external service providers. It is required within the system architecture to limit the access of certain services and allow management control of available services. At the same time, it provides traceability and security for any information that leaves the enterprise's network.

The Gateway Server, or GWS, is implemented as a software firewall that controls access to external services. It runs on a java enterprise server to send and receive JMS messages to and from the Service Provider. The messages can be intended for the GWS itself, or the internal clients within the enterprise. The GWS automatically makes the decisions on where to forward the information.

Table 9. Gateway Server Java Messaging Service Queue Subscriptions

Queue	Communication Method
pegasus/gatewayQueue	Receive
pegasus/serviceQueue	Send

Table 9 describes the JMS queues that the GWS subscribes to, along with the communication method utilized for each queue. The GWS runs within the enterprise and listens for any new clients to connect. It then determines if the clients wish to retrieve the list of available services, or submit a job to an external service provider. It will respond in the appropriate manner.

Table 10. Gateway Server Ports

Service	Port
Data Communication	5523
Text Communication	5524
Verification Server Communication	5525

Table 10 describes the necessary ports that must be available for the Gateway Server to perform its functions properly. Port 5523 is used when establishing a SSL connection with client computers so that the clients can upload their data to the GWS. Port 5524 is used for sending function calls back and forth from the client to the GWS. This port is also encrypted over SSL. Finally, port 5525 is used to establish the Verification Server encrypted SSL connection.

The GWS must establish a SSL connection with the Verification Server before it can begin sending and receiving JMS messages. This is to record transactional information about the JMS messages and jobs submitted by its internal clients. Also, it needs to receive encryption keys before it can decrypt the JMS message containing the results.

4.1.2 Operation

When the Gateway Server is first invoked, it connects to the java enterprise server (J2EE) and subscribes to the JMS queues for sending and receiving Secure JMS messages to and from the Service Provider. It then establishes a SSL connection with the Verification Server for transmission of JMS transactional information. It retrieves a list of currently available services from within its own directory service and stores this list into memory so when a client computer connects to the GWS from within the enterprise, the list of services can be sent to the client and displayed in their software.

After retrieval of available services from the list of approved services, the GWS opens and listens for SSL connections on ports 5523 and 5524. This is the data and text port that clients use to submit jobs and request access to the outside world. When a client first invokes his/her software program, the program will connect to the GWS and retrieve an XML file of the services currently available to the enterprise. After that, it resumes listening for any number of clients to connect to receive the list of available services, or to submit a job to a service provider.

Processing of Incoming and Outgoing Data

After the GWS completes its initial runtime setup including connecting to the J2EE server, subscribing to the JMS queues, opening its ports, and transmission of available services to clients, it awaits for a job submission connection to occur. When a client connects to the GWS to submit a job, the GWS establishes a SSL connection with the client and receives the service request information, temporarily storing any files on its file system for later processing. The GWS then looks up the client's demographic information, such as their internal IP address, username, and e-mail address and creates a job, internal to the enterprise, which contains the following information: job id, username who submitted it, IP address, the service provider that the file is being sent to, and the service that the service provider is providing. This is similar to Table 6, which describes the `jobs` table within the Verification Server.

The GWS then packages any files related to the job and any arbitrary META information into a Secure JMS message, publishing the packaged message to the service provider message queue. This informs the Service Provider that there is a pending job waiting. The GWS then sends JMS transactional information to the Verification Server for storage in its database. This includes all the information listed in Table 6. When it is finished publishing the job the client requested, it resumes listening for any new jobs.

When the service provider completes the job, it publishes the results to the GWS message queue. This notifies the GWS that a completed job has arrived. The GWS begins to receive the JMS message. The GWS calculates a hash value for the received Secure JMS message, and sends that to the Verification Server with a request for the encryption key. The Verification Server verifies the hash, and then sends the encryption key to the GWS so it can decrypt the received message.

The GWS decrypts the message and retrieves the user information that is attached to the complete job. It uses this information for location lookup, so that it knows where to send the completed results. Also, the GWS sends final transactional information to the Verification Server to record successful retrieval of the completed job.

After locating the original sending client, the GWS establishes a connection to that client and sends the processed file back to the client. It then closes its connections with that client and begins to listen for any new requests.

4.2 SERVICE PROVIDER

The Service Provider is a main technology accessed on a regular basis within the system architecture. It provides the services for the design environment through the use of a software program, which initiates J2EE communications through JMS services and shares a connection to the Verification Server via a SSL connection. The software program acts a service provider company and calls external programs that execute and perform functions on the incoming data. Through the use of this technology, execution of services, for enterprises within the new design environment, can be performed.

The Service Provider implemented within this architecture can apply to any executable desired. The program chosen to execute was “dot”. Dot makes “hierarchical” or layered drawings of directed graphs. The layout algorithm aims edges in the same direction (top to bottom, or left to right) and then attempts to avoid edge crossings and reduce edge length [10]. Dot is the Linux implementation of the Graphviz project [10].

4.2.1 Requirements

This particular server provides the JMS system for service providers to send and receive JMS messages within the system architecture. Although Graphviz can run on different operating systems, this particular implementation requires a Linux operating system environment to perform properly.

The main requirement that the service provider satisfies is the ability to execute arbitrary services and provide a mechanism for access to those services. The concepts behind the service provider can be applied to any provider wishing to offer services for service oriented architectures.

4.2.2 Design of the Service Provider

The Service Provider subscribes to JMS queues for receiving and sending information. Table 11 describes the queues the Service Provider subscribes to and the communication method of that queue.

Table 11. Service Provider Java Messaging Service Queue Subscriptions

Queue	Communication Method
`pegasus/gatewayQueue`	`Send`
`pegasus/serviceQueue`	`Receive`

The Service Provider will listen for any new messages that are published to the service provider queue. The Service Provider is notified of a new message by the JMS API specified within J2EE. It will also establish a SSL connection with the Verification Server before it can begin receiving messages. This is to record transactional information about JMS messages and jobs. Also, it needs to receive encryptions keys before it can begin processing a received job.

4.2.3 Operation

Operation of the service provider will now be described. Figure 16 shows the complete operation of the Service Provider, from receiving the file, processing it, and returning the file to the enterprise that submitted it.

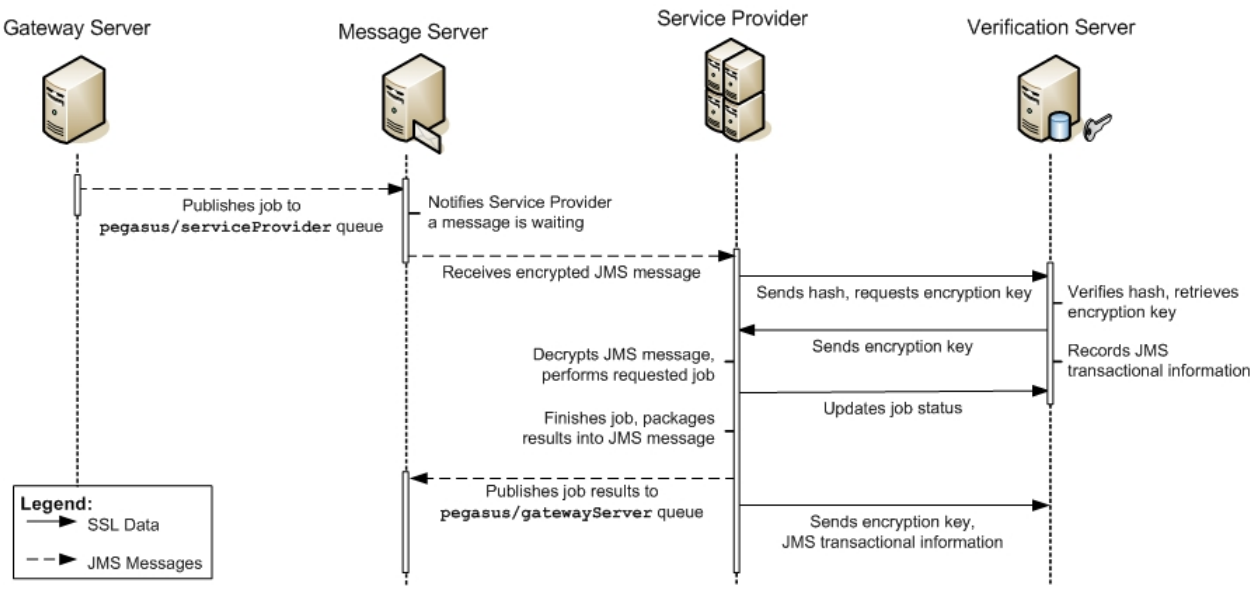


Figure 16. Service Provider – Job Execution Process

When the Service Provider is first loaded, it connects to the J2EE server and subscribes to the queues for sending and receiving JMS messages as shown in Table 11. It also establishes a SSL connection with the Verification Server. After the connecting and loading process, the Service Provider listens for any new messages that are published to its queue.

Receiving Job Request

An enterprise's gateway server publishes a job to the Service Provider's queue as shown in Figure 16. The JMS server then notifies the Service Provider that a message has been received and the Service Provider begins receiving the encrypted JMS message. After receipt of the message, it calculates the hash of the message and sends it to the Verification Server, along with a request for the encryption key. Included in this transmission is the JMS transactional information (the Service Provider's demographic information), to update the JMS transaction in progress. The Verification Server verifies the hash matches the one on file (to make sure the message was not altered during transmission), retrieves the corresponding encryption key, and sends the key to the Service Provider. Then, the end of the transaction is recorded for the in-progress JMS transaction. The Service Provider receives the key and decrypts the JMS message,

sending a job status report to the Verification Server stating that it received and decrypted the JMS message successfully, and that the job has begun.

Sending Results of Completed Job

The Service Provider processes the job according to what service the job is requesting and stores the result so that it can send it back to the enterprise. When it finishes the job, it packages the results into a JMS message and publishes the message to the `pegasus/gatewayQueue` queue. After that, the Service Provider sends the encryption key of the published message, along with the JMS transactional information, to the Verification Server for storage. After completion of the entire process of executing a job, the Service Provider returns to an idle state, awaiting the next job to be published.

4.3 CONCLUSIONS

Utilizing the information infrastructure for secured and traceable communications are infrastructure services. They complete the system architecture of the middleware solution. These entities provide mechanisms for controlling access to services provided by service providers. The Gateway Server offers control for the enterprise and the Service Provider offers services for the enterprise to access.

The Gateway Server is another core contribution that enables and controls the enterprise from a single point of control. It also allows easy management of remote services and network protection through the use of a software firewall. Remote services are accessible to the enterprise through the Gateway Server and management has the ability to dynamically control which products and services are available.

The Service Provider provides services to the enterprise over the information infrastructure detailed in Chapter 3. Security, traceability, and non-repudiation occur for any service accessed as explained within this chapter. Since this new design environment extensively utilizes services and service providers, the service providers can provide services that replace the standard software features most design software packages employ today.

5.0 SECURE FILE SHARING SYSTEM

There are many types of secure file systems in the computer industry today. Most of them do not include a comprehensive collection of features necessary to secure the files, secure access to the files, and hold people accountable for accessing the files. It is relatively easy to send small files to one another using e-mail, but when the files are large, 100–400 Megabytes or even Gigabytes, what happens? How does one guarantee that the files arrived successfully while maintaining file integrity?

The proposed secure file sharing system plans on addressing those issues, along with many others, without burdening clients or complicated front-ends. Simplicity to the end user was the driving force behind development of this secure file sharing system. While maintaining a high level of security on the file system itself, transport of those files over potential unsecured networks is possible as well. This secure file sharing system is a core contribution and was designed as an additional technology within the system architecture. It can be applied to any environment, not just design environments. The technology is available for many different uses within Industry today.

Transferring Large Files

When implementing a distributive, secure file sharing system, several of the following issues arise. How do you transfer a large file securely while minimizing man-in-the-middle attacks? Once the file is transferred, how do you store the file securely; and once the file is stored, how do you protect the server the files are stored on? What would happen if the server was physically compromised? How do you prevent DDoS'ing and spoofing of identities? These issues are of utmost importance when it comes to securely transferring and storing large files within a system. A secure system is only as strong as its weakest link [1]. If any part of this system is weak, the

entire system as a whole becomes weak. Details of how strong this system is, and the security integration and operation are explained further in this chapter.

Redundancy

Redundancy is another issue that arises when developing a secure file sharing system. What happens when a server goes down, or an entire network goes down? How do you guarantee that the files will still be accessible? Do you replicate files manually, or have some type of automated replication process occurring according to policies? Automation of file replication ensures that data loss is curved down to a very small probability, and eliminates the possibility of human error.

Accessibility

Accessibility is an important issue when cross platform compatibility is desired. How do you allow for multiple operations to occur on several different hardware and software architectures? How to do you control access and prevent unauthorized access to the servers? While maintaining a certain level of accessibility, the need for accountability is also evident within the system. How do you maintain repudiation? How do you track where a file has been, and who has viewed or accessed the file? Common uses of databases allow for such tracking, but take that one step further. The ability to backup and restore database entities to encrypted XML files on the file system would allow for temporary operations on the file system while a database server was down or inoperable. It would also allow backups of the database to be maintained.

Secure FSS integrates secure file transfer, secure file storage, accessibility, extensibility, cross platform compatibility, the ability to store an arbitrary amount of META information with each file, accountability, server protection, and redundancy into one complete software package.

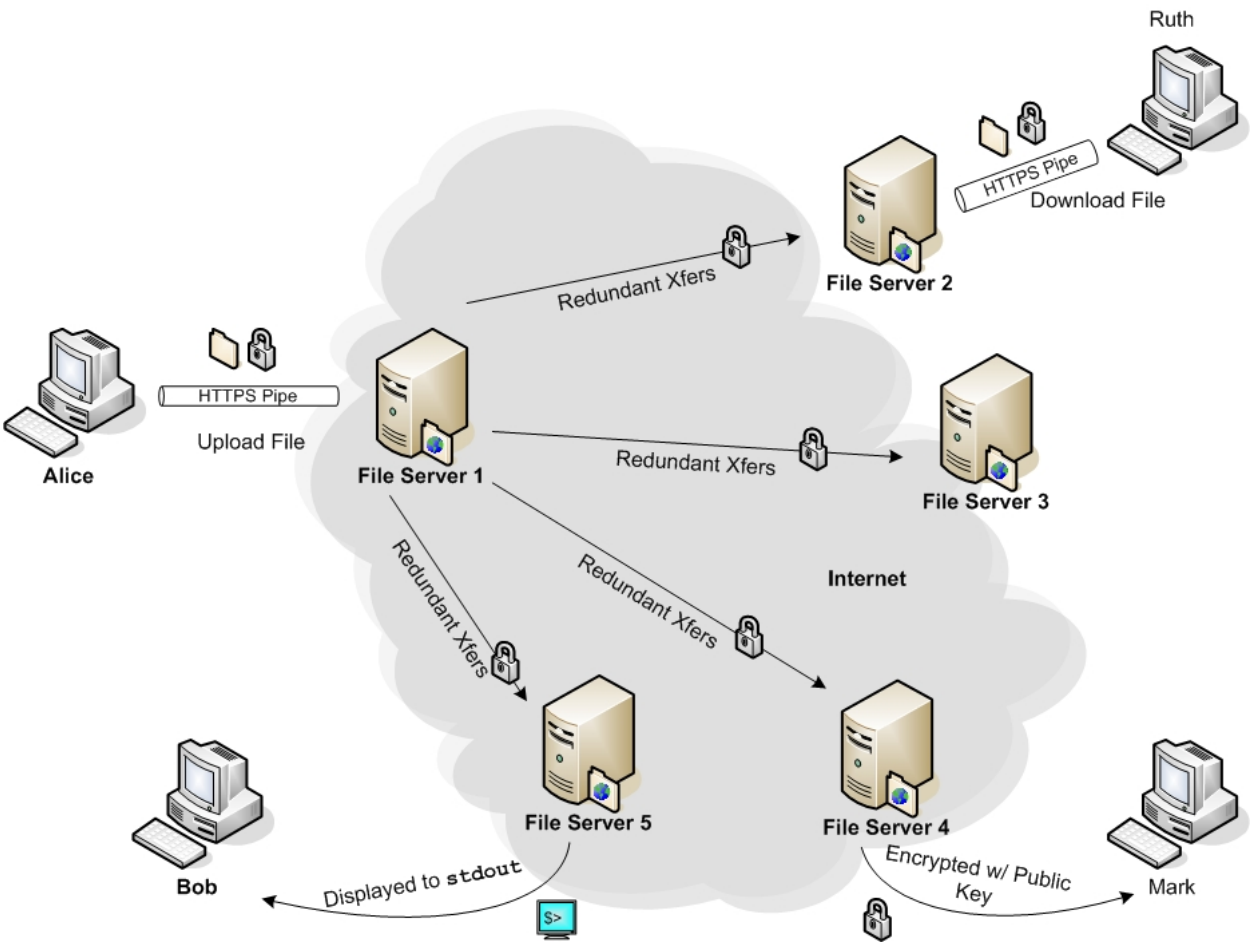


Figure 17. Overview of Secure File Sharing System

5.1 OVERVIEW

Figure 17 above describes an overview of the Secure File Sharing System, complete with the several options of input and output available to the system. For example, Alice uploads a file via the HTTPS mechanism built into Secure FSS to File Server 1. After completion of the upload, the server encrypts the file and transmits it, over regular non-encrypted sockets, to the other redundant servers as shown in the figure. It is not necessary to send the redundant transfers over an encrypted connection because the files being transferred are already encrypted.

Users can download the files in a number of ways. Ruth demonstrates downloading the file via HTTPS through her web browser. Bob demonstrates downloading the file with the contents displayed or piped to `stdout`. This allows Bob to process and utilize the file's contents

without the file existing on his file system. Mark demonstrates the file being download to his file system, but encrypted with his public key. This option is for users that wish to use their public key for encryption, rather than Secure FSS's public encryption key. These options are further explained in the following subsections.

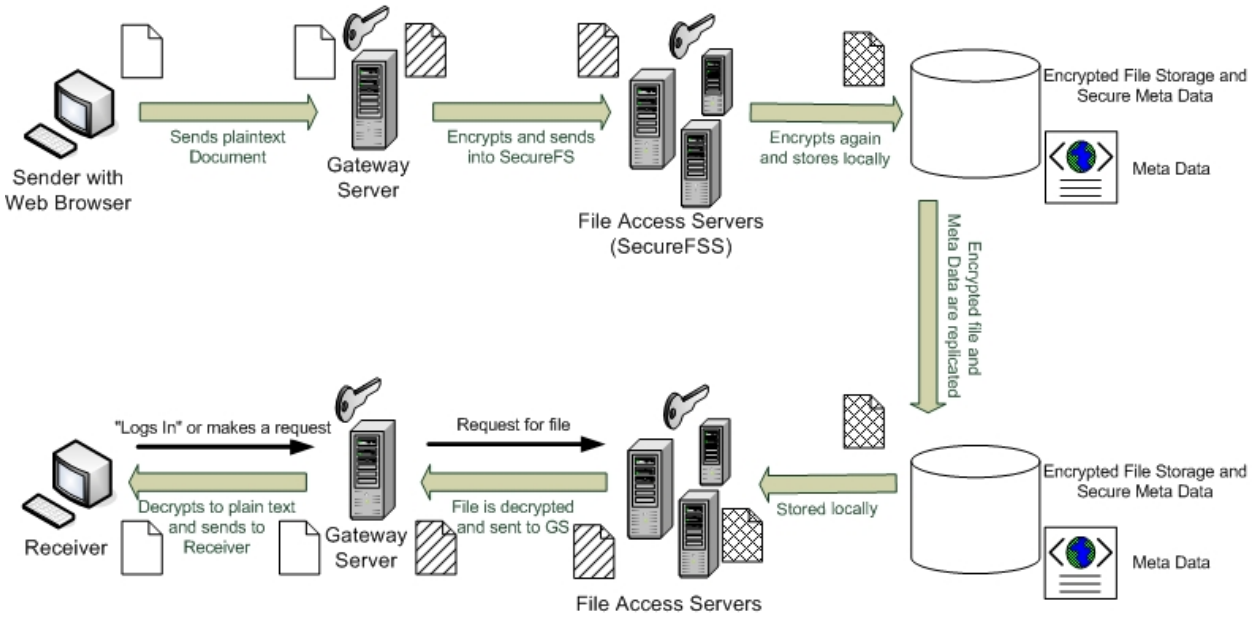


Figure 18. Multiple levels of encryption and security within Secure File Sharing System

5.1.1 Example of File Transfer within Secure File Sharing System

Figure 18 shows the different components of the secure, distributed file sharing system, Secure FSS. The following text describes how a file gets from one user to another through the use of Secure FSS. A user with a web browser sends a plaintext document to the Gateway Server. This server encrypts the file and sends it into the Secure FSS. The Secure FSS encrypts it again and stores it locally, along with updating its META data information. The encrypted file and META data are replicated to redundant servers. Receivers would then "Log In" and ask for a specific file. The Gateway Server requests the file from the File Access Servers. The Secure FSS decrypts the file and sends it to the Gateway Server. The Gateway Server then decrypts it to plaintext and sends it to the receiver, thus completing the transfer process.

5.1.2 File Security during Transmission and Storage

To safely transfer files between the end user and the server, Java SSL sockets or HTTPS (Hyper Text Transfer Protocol Secure) within web browsers has been selected as the protocols of choice. All encryption of files and META data is performed by GnuPG, an open source implementation of the OpenPGP standard as defined by RFC2440 [11].

Authentication and Tracking

Authentication is performed with digital signatures via GnuPG. MD5 hashing is used to verify the successful transfer of all files and acts as a digital fingerprint for each file. The hash for each file is also stored for file integrity purposes. A database stores all tracking information for Secure FSS.

5.1.3 Design of the Secure File Sharing System

Let's start describing the inner workings of this new system, answering all the questions suggested within the introduction of this chapter. Once the files have been transferred, they are stored within the Secure FSS infrastructure. It includes MD5 hashing the directory structure so that if physical compromising of the server occurred, the directory structure has meaningless information and sensitive information such as filenames would not be available. Figure 19 shows how this directory structure might look.

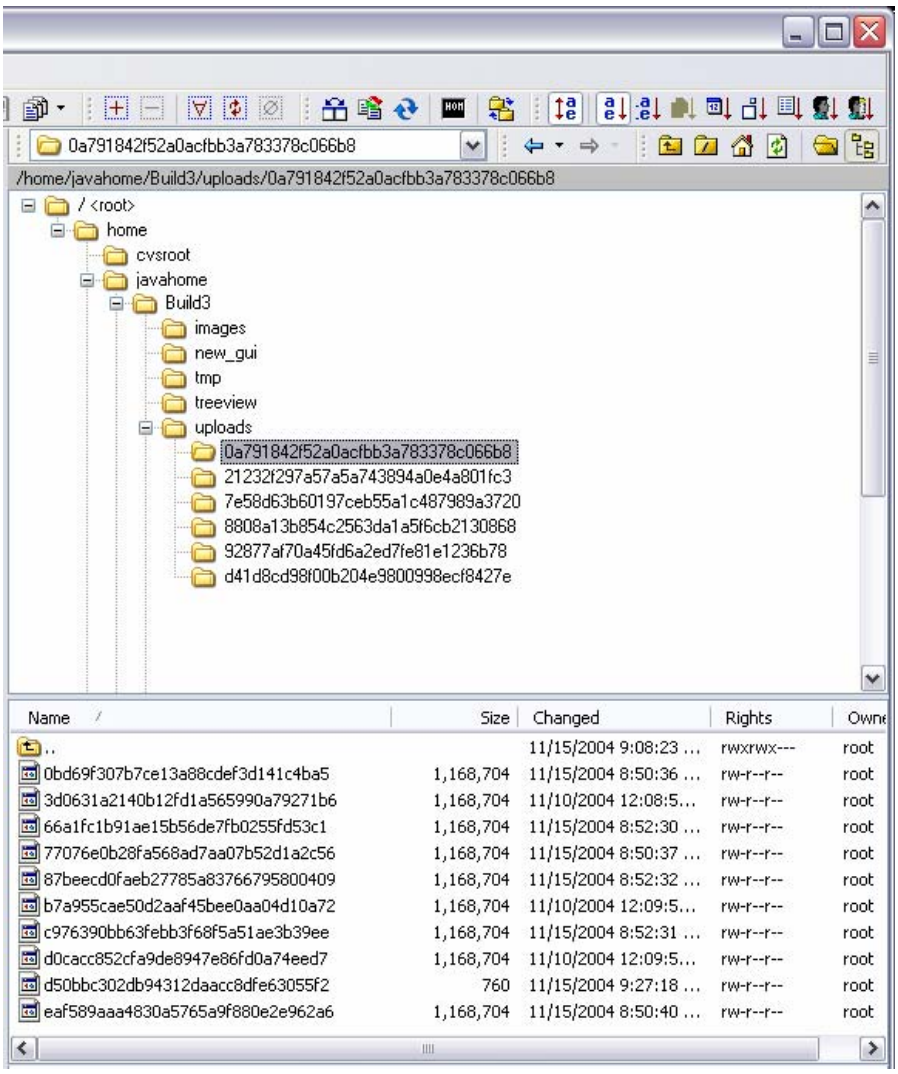


Figure 19. Directory Structure of Hashed Directories and Filenames

The files themselves are then encrypted using the server's private key and GnuPG, the standard for strong cryptography. Any amount of META information can also be sent with the files and stored via XML files. The XML files would also be encrypted using the server's key. Signing the encrypted files allows for verification and integrity checks of those files. If a user wishes to upload an already encrypted file, then double encryption will be present, for when the most extreme sensitivity cases occur.

To protect the server from various remote threats, digital signatures of files and requests are implemented. This also helps prevent spoofing of identities when accessing the server. Quotas and limits for storage amounts and capabilities prevent a DDoS attack on the server, and

secure admin access via GnuPG pass phrases and key-rings ensure that no unauthorized access occurs during administration. The server maintains its key-ring and public/private keys in encrypted form on the file system. A startup pass-phrase is necessary to decrypt this information and start/run the server. This is the master key so a strong pass-phrase is recommended.

Repository redundancy is a must in the industry of shared file system resources. Replication on this system would be controlled by a series of stored policies within the servers. The ability to replicate to multiple groups is available as well; any additions or subtractions to this group server list are transparent to the end user. They just need to know the group name to replicate to and the server takes care of the rest. The automated replication also provides methods of controlling bandwidth and connections between servers. This allows scheduling of redundant processes so that QOS is maintained within a network. A queuing mechanism is implemented for fairness during the replication process as well as a search feature for when files don't exist on a local server (because they have not been replicated yet).

Access to the server is controlled by access policies to prevent unauthorized access. Public key signatures would be used to provide a means of digital authentication. 100% Java-based code provides cross platform compatibility between operating systems and hardware.

The use of transactions provides accountability for all data moved within this system. The server records when a file is uploaded or downloaded, system commands issued to add or delete access, policy additions and subtractions within the servers, and any errors that may occur so that tracking down problems and bug reporting is easier. This allows for an individual verification of files sent, received, and accessed.

Secure FSS allows for extensibility to other programs by being a Java API interface. This can be either command line based, or parameter based (with parameters stored in a digitally signed XML file). This allows it to integrate seamlessly into already existing software packages such as PHP, Perl, or other scripting languages. Integration into programmable clients is possible as well.

5.2 OPERATION

Secure File Sharing System employs methods of uploading files, searching for files that are not locally stored on a redundant server, and downloading files. These methods are explored below with different use-cases that step through each scenario. Figures 20 and 21 illustrate these steps.

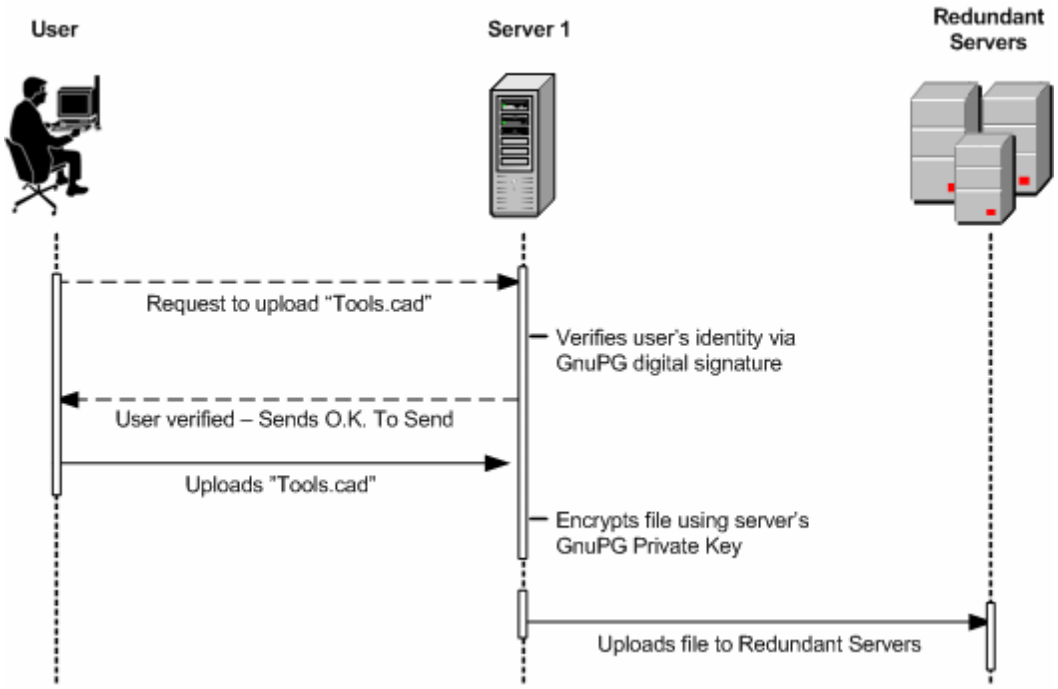


Figure 20. Sequence Diagram of File Upload within Secure File Sharing System

Use-case of File Upload

A request to upload a file is done from the user to the server, as shown in Figure 20. Prior to this request, the user has stored parameters such as filenames, MD5 hashes, timestamps, and authorized receivers in an XML file. This XML file is then digitally signed using the user's private key. The server verifies the user's identity by checking the signature against its key-ring database (the server has the user's public key). After verification, the XML file is then used as the "request" and the server extracts the information within the XML file. When the extraction is complete and the server is ready to receive the file(s), it sends an O.K. to send signal to the

user. The user then uploads the file. During the transfer process, the server encrypts each received byte stream using its own private key. This eliminates the need for temporary files on the file system and prevents the file from being stored anywhere on the file system in clear text form. After a successful file upload, the file is distributed to redundant servers maintained in a list by a policy file.

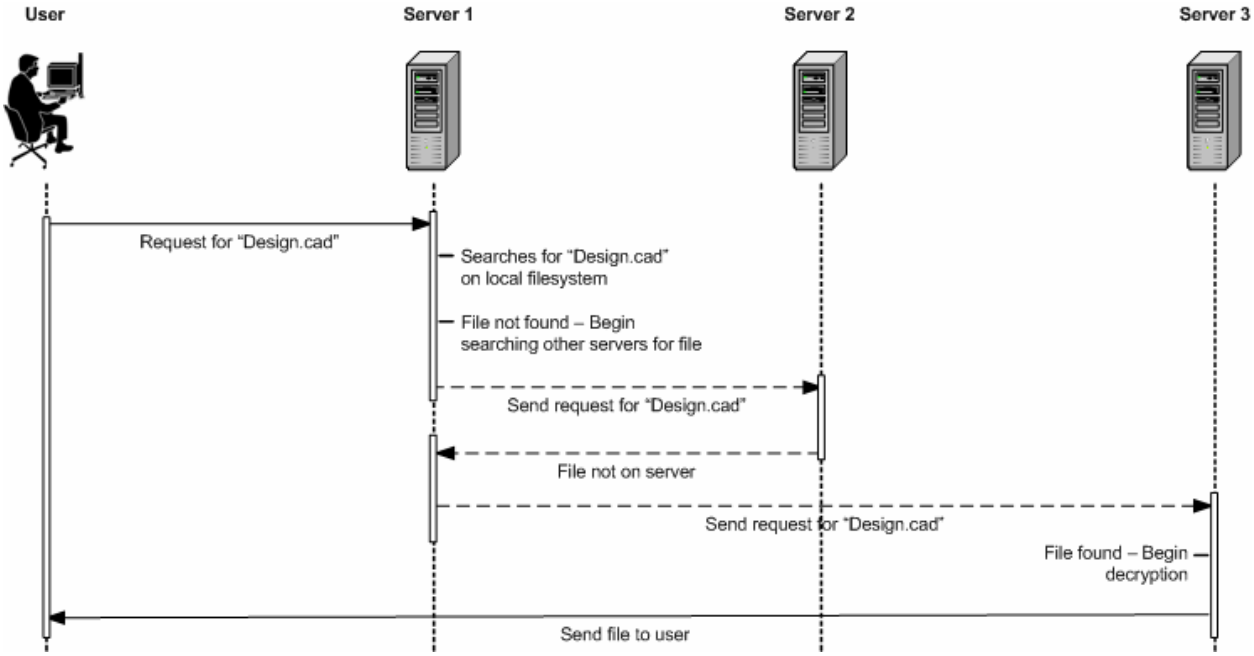


Figure 21. Sequence Diagram of User Searching for a File within Secure File Sharing System

Use-case of File Search between Redundant Servers

Figure 21 above describes the process by which the search feature is implemented (when a file requested has not been replicated yet to the server the user is connected to). A user requests "Design.cad" from the Secure FSS server local to his location. The server searches for the file and determines that the file is not found. It then begins requesting the file systematically from each redundant server within the Secure FSS infrastructure. Once the file location is found, the file is returned back to the user.

5.3 CONCLUSIONS

Integrating several different security features into an all-in-one inclusive package can solve the major problems of secure file transfer and secure file storage. Through the use of standard secure protocols and methods (HTTPS, SSL, MD5, and GnuPG), the trustworthiness of the system is very high and reliable. If needed, this system allows for double encryption; if the user encrypts the file before sending it, the server will encrypt the encrypted file after it is received thus providing double encryption on the file. This provides maximum security for file storage on the server. Redundancy and automated replication allow multiple servers to be utilized across several different networks and several different corporate infrastructures. Accountability for every action within the system provides for non-reputability and for tracking information to be extracted for each file deposited on the repository.

The benefits of this new Secure File Sharing System over other already established systems include being open source, the use of Java as the programming API, and extensibility to many different applications. Integration of several different security issues, redundancy, accessibility, and accountability within one system allow ease of use and simple source and server management.

6.0 ECLIPSE PLUG-IN AS A DEMONSTRATION

To demonstrate the practicality and versatility of the new middleware solution for distributed design environments, a plug-in was developed within Eclipse to simulate a CAD design environment. Eclipse's extensibility and community support provides the best environment for this type of demonstration. With Eclipse, dynamic retrieval of services is possible through the use of XML, along with submission of jobs to remote services. Since Eclipse is written and programmed in the Java programming language, development of the Eclipse plug-in was relatively simple, and integrated very well with the design architecture and specifications of the system.

Each individual technology of the system architecture was developed with a stand-alone design approach. Throughout this process, features were analyzed and curtailed to make them more efficient. After completion, tests were performed in a stand-alone environment to ensure the technologies performed as expected. After testing, each technology was integrated into the comprehensive solution, which incorporated and provided the benefits of the distributed design environment's concepts and theories.

Although the integration process did not take very long, a retrospective look back at the architectural design indicates that a few optimal solutions were overlooked. Slowly designing each technology all at once and integrating the pieces together from the beginning of development would have provided an optimal development process. When integrating multiple components into one, many problems are encountered that prevents full integration within a specified time period, even though this did not occur during the integration of the system architecture.

Throughout this demonstration, the following will be assumed. The client enterprise is a CAD design firm with engineers that use Eclipse to program CAD designs. The enterprise will need to request remote services because they do not have the proper software or hardware to compute the results needed. These requests will be sent to the Service Provider, which will be a company that provides super computing services. The design firm will house the Gateway Server and the engineers will use Eclipse. The Verification Server and Service Provider will be

separate companies distributed across the Internet. The Verification Server will act as a separate, third party company that shares trust between the service provider and the enterprise. The participating parties, the enterprise and the service provider, will assume authentication with the Verification Server.

6.1 SYSTEM REQUIREMENTS

Although the system architecture defines four individual computers, for this demonstration, only two computers were necessary to represent the four components of the architecture. The first computer represents the enterprise: the client or engineer, and the Gateway Server. This computer required the installation of a J2EE server along with the Eclipse Development Platform. Any operating system that supports both Eclipse and J2EE can be used. For this demonstration, the Mac OS/X operating system was used. Setup and administration of the J2EE server as described in Section 4.2.2 was also required.

The second computer represents the Service Provider and the Verification Server. This computer required the installation of a J2EE server and the "dot" visualization program. Linux was the operating system used for the second computer because of the implementation of Graphviz utilized ("dot"). This computer also required the installation and setup of the MySQL database application so that the Verification Server could function properly. See Section 6.2.3 for details regarding the MySQL setup.

6.2 SETUP

There are four main setups necessary to operate and demonstrate the system's functionality. They include setup and installation of the Eclipse Platform and system plug-in, setup of the J2EE server for the client and Gateway Server, setup of the J2EE server for the Service Provider and Verification Server, and setup of the MySQL database application. Each J2EE server installed was set up with the Sun Java System Application Server Platform Edition 8.1_02 as shown in Figure 22.

Figure 22. J2EE Application Server Description and Version

When administering the objects to facilitate JMS functionality, JMS requires two parts for JMS applications to function; connection factories and destination resources. JMS requires the use of connection factories in order for a client computer to connect to a JMS provider. A connection factory encapsulates a set of connection configuration parameters that has been defined by an administrator [12]. The overview of the connection factories set up and administered for the system is shown in the Appendix, Figure 28. JMS also requires the setup of physical destinations to store the actual queues that the middleware solution uses. A destination is the object a client uses to specify the target of messages it produces and the source of messages it consumes [12]. The overview of physical destinations set up and administered for this demonstration is shows in the Appendix, Figure 31.

MySQL version 4.0.20 was used for the database application. This was downloaded from http://dev.mysql.com/downloads/mysql/4.1.html and administratively setup according to the installation instructions.

For the remainder of this setup section, refer to the Appendix at the end of this document that contains the figures and screen shots of each of the setups explained for the Gateway Server and Service Provider.

6.2.1 Client and Gateway Server

Eclipse was installed by downloading the latest version from http://www.eclipse.org/downloads/ and following the installation instructions as listed. After that was completed, the middleware solution plug-in was installed by the drag and drop method, placing the solution plug-in into the `plugins` folder located within Eclipse's installation directory. This will enable the solution whenever the Eclipse platform is first invoked.

The J2EE server was installed by downloading the latest version from Sun's website at http://java.sun.com/j2ee/1.4/download.html. The installation instructions on installing the server were followed as listed, setting up the default administrator and domain information, and starting and stopping the application server instances. Once those processes were complete, setup of the JMS queues could now occur.

The (pegasus/EngimaConnectionFactory) is the Connection Factory within the JMS Resources that represents the Gateway Server. Figure 30 displays the properties necessary for configuration. The `AddressList` property as shown contains the public IP address of the enterprise's GWS. All other properties can be left at their default values. Because this computer will be accessing a remote service provider, the (pegasus/J9ConnectionFactory) needs to contain the service provider's IP address for the `AddressList` property. This is displayed in Figure 29.

After entry of the physical destinations as listed in Figure 31, the setup of the JMS queue for the GWS can occur. The queue for the client enterprise is the (pegasus/gatewayQueue) as shown in Figure 33. Note the value for the property "Name". This is the physical destination created for the GWS. Once these steps are completed, the client enterprise is ready for the demonstration.

6.2.2 Service Provider

Like in Section 6.2.1, the Service Provider requires the download and installation of the J2EE Application Server. Everything was performed the same, until administration of the JMS objects occurred. Since this particular administrative instance of the J2EE server is running on the computer that houses the Service Provider, it is necessary to enter the remote enterprise computer's (client and GWS) IP address for the `AddressList` property. This is displayed in Figure 30. This allows access to the enterprise's messaging queues for sending completed jobs.

After entry of the physical destinations as listed in Figure 31, the setup of the JMS queue for the SP can occur. The queue for the Service Provider is the (pegasus/serviceQueue) as shown in Figure 34. Note the value for the property "Name". This is the physical destination created for the Service Provider. Once these steps are completed, the Service Provider is ready for the demonstration.

6.2.3 Verification Server – MySQL Database

MySQL was installed by downloading version 4.1, to ensure compatibility with the JDBC implementation in the system, from http://dev.mysql.com/downloads/mysql/4.1.html. Follow the instructions for installation and setup the database according to the tables listed in 3.2.5. This includes the username, password, database name, and creation of Table 2 through 8. Once that is complete, the database will be set up and ready for operation.

6.3 OPERATION

Now comes the fun part, running the new middleware solution! This section will describe the complete process of designing a file, submitting it to a service provider, receiving a processed result, and displaying the results directly within the design program. A series of screen shots will accompany each major step throughout this process to aid in visualization. The comprehensive middleware solution will now be highlighted.

Figure 23 describes the engineer's viewpoint within the enterprise. They are only aware of connecting to the Gateway Server and when they request a service from within their design software, they will be unaware that the service they requested is remote and not on their system. This eliminates the complications encountered when using third party software for submission of services, and also allows the engineer to work without worrying about whether they packaged the file correctly or submitted it to the right provider. It provides a hassle free workflow.

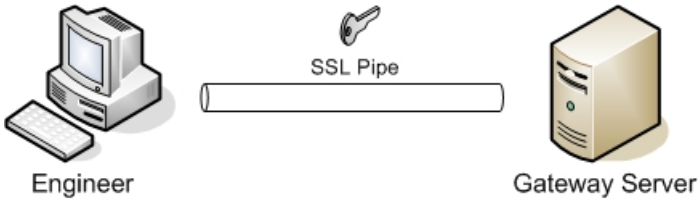


Figure 23. Engineer's Viewpoint

The individual parts that were implemented for this demonstration (from Figure 1 in Section 1.2) are Engineer number 1, the Gateway Server, the Verification Server, and the Service Provider. Though these entities represent four different computers, each entity ran within its own instance on two computers, one housing the engineer and Gateway Server and the other housing the Service Provider and Verification Server. Future demonstrations will separate each part into individual computers as shown in Figure 1.

After Eclipse is opened, the engineer would create a new text document to create some source code. After completion of the source code, the engineer would request that the source code be converted into a graphical representation. Since the concept of the system is for several different remote services to be available to the engineers working within a development environment, this engineer would access the services as if they were currently installed on their system. In Figure 24, notice the grayed out selection below "Graphviz Conversion." This option is currently not available, because the enterprise's management decided that this service would not be available to the engineers. This list of services is updated every time the engineers first invoke Eclipse via XML.

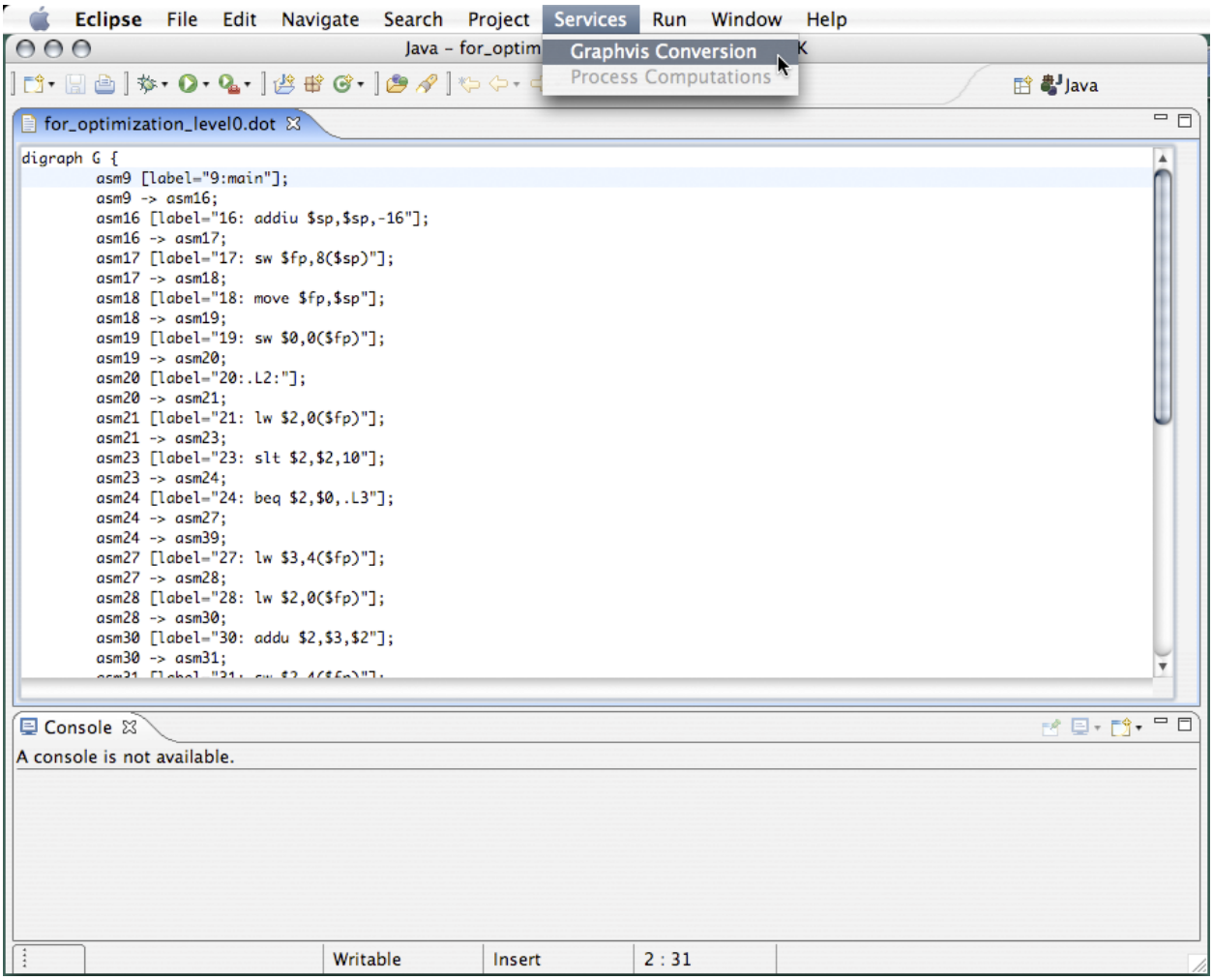


Figure 24. Service Selection within Eclipse

After this service has been selected, the following list of steps occurs; sending the job to the service provider, receiving the job status, and finally receiving the completed job.

- **CAD Engineer**
  1) The file that is currently opened and selected within Eclipse is submitted to the Gateway Server for processing and submission.
  2) A window showing the status of the submitted request is displayed to the engineer. This updates at each stage during job execution. After a job is successfully submitted, the job identification number is returned to the user through this window as shown in Figure 25.

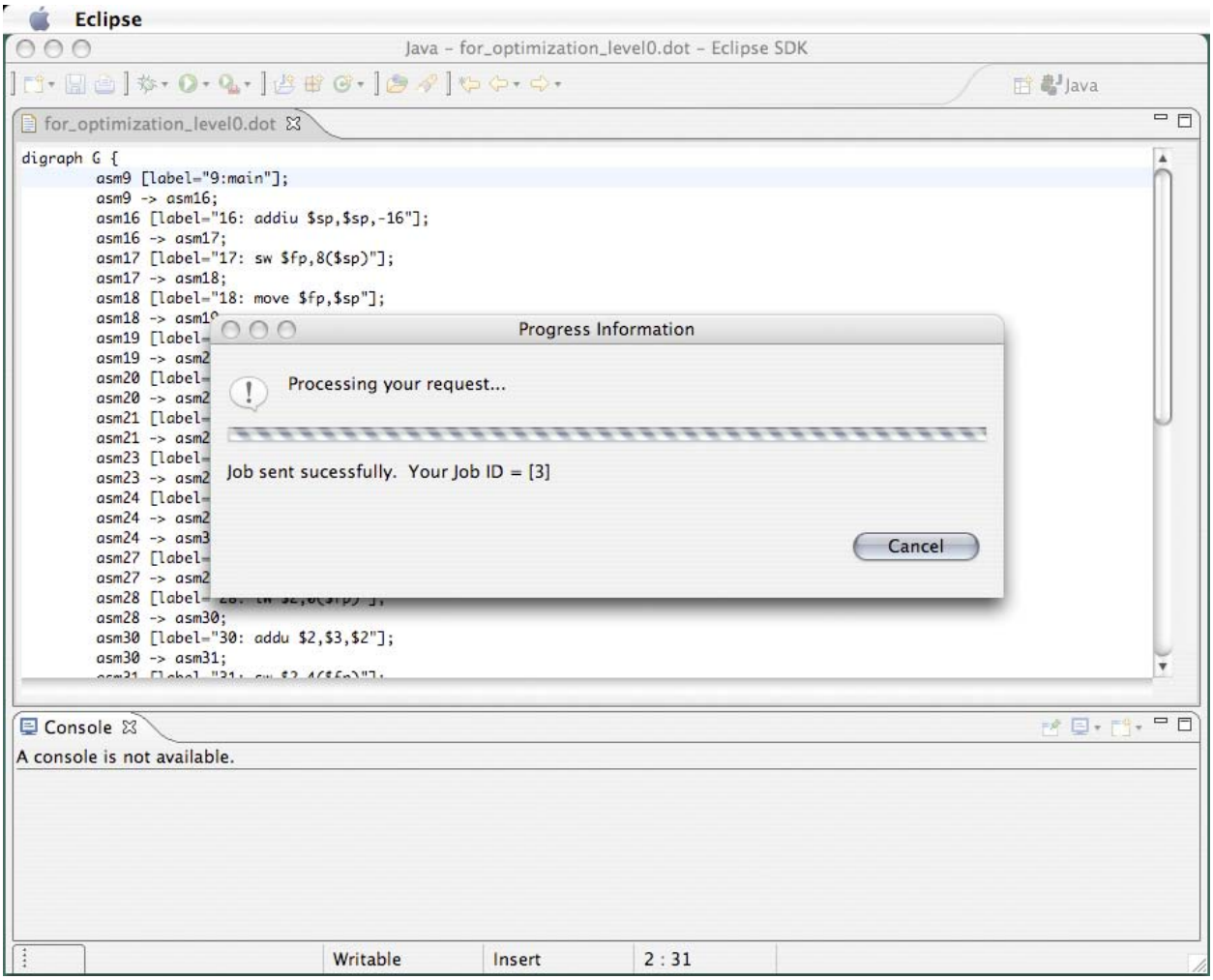


Figure 25. Job Status Window Displayed to the Engineer

- **Gateway Server**

3) Receives the file from the engineer and creates an internal job.
4) Packages the job request and any META information available into a Secure JMS message.
5) Sends the Secure JMS message to the Service Provider.
6) Sends the transactional information to the Verification Server to record the process of message exchange between itself and the Service Provider. This includes information about the job and the Secure JMS message transmitted.
7) Returns the job ID to the engineer so that they can record it for tracking purposes.

- **Service Provider**

8) Receives the JMS message from the Gateway Server and assigns it an internal job number.
9) Retrieves the message's encryption key, decrypts the message, and retrieves the job's META information. Then sends a receipt of the job, job number related to the job, and any META information to the Verification Server.
10) Processes job.
11) Packages the completed results into a Secure JMS message, and sends the message back to the Gateway Server through the Gateway Server's messaging queue.
12) A message indicating completion of the job and transactional information is then sent to the Verification Server.

- **Gateway Server**

13) Receives the completed job and processes the META information to locate which engineer submitted the job.
14) Sends a message to the Verification Server indicating successful receipt of the results.
15) After locating the engineer, it establishes a new connection and sends the results.

- **CAD Engineer**

16) Receives the finished job and Eclipse displays it directly within the program as shown in Figure 26.

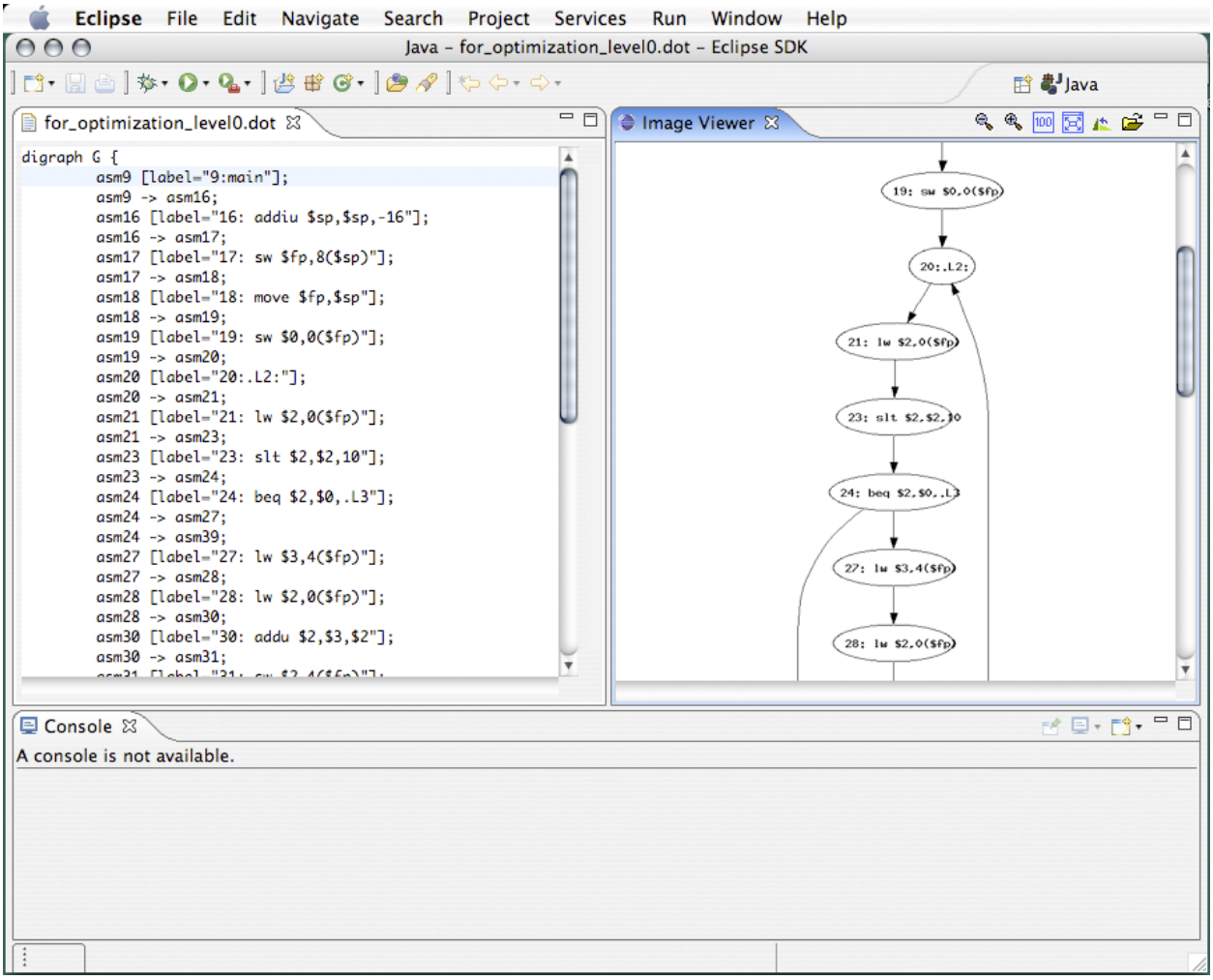


Figure 26. Display of the Completed Job

6.4 RESULTS

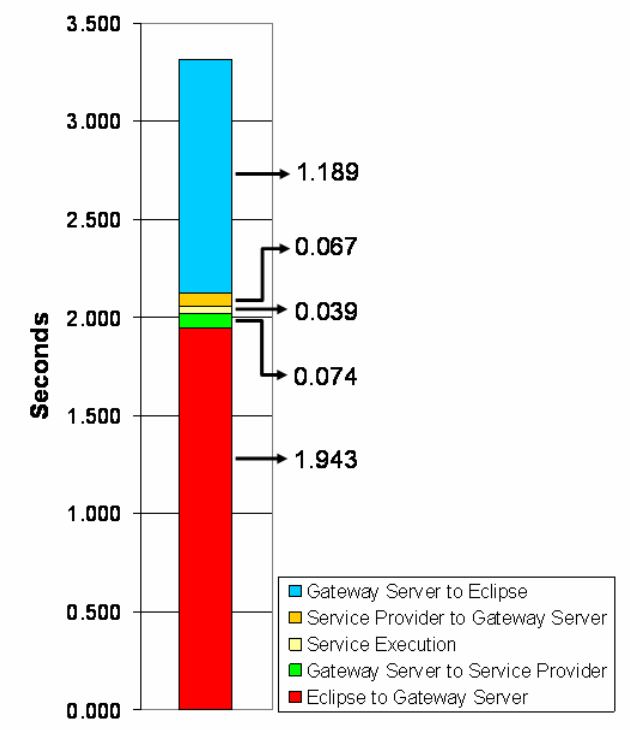


Figure 27. Performance Results of Middleware Solution

Figure 27 describes the performance results from execution of the middleware solution. These results indicate that the overhead for utilizing the middleware solution is only 0.180 seconds. This is a very small timeframe considering all that is occurring to maintain security, traceability, and non-repudiation. The large bars on the graph are not a result of the infrastructure being used but rather a result of a non-efficient Eclipse plug-in. This demonstration was meant only as a proof of concept, not a practical implementation.

Throughout this demonstration, the complete system architecture was displayed through the use of Eclipse to write some source code, submit it through the information infrastructure, and receive an image back representing that source code. This look at how the middleware solution operates gives insightful information on how this system can improve service oriented architectures.

The major stakeholders within the new design environments are satisfied. The engineer is satisfied because they can access remote computational services directly within their design software package. Management is happy because they can control these services with simple ease and without upgrading or modifying existing software packages, many of which are located on several different machines. Service Providers are happy because their services can be utilized without complex management, and they can process jobs in an efficient and effective manner. All parties are happy because this system provides the ability for security, traceability, and non-repudiation. These features allow for disputes to be settled quickly between enterprises and service providers, if a job gets lost, or a service provider violates its contract with the enterprise, or vice versa.

Finally, the most important benefit of this integrated architecture was that it allowed seamless integration into the engineer's everyday workflow. There is no need to modify existing software (aside from the menu system for selecting services) or purchase additional software to execute services in house. Services can be added on a pay-per-view basis by management and management has control over which services are offered.

7.0 CONCLUSIONS

When we think about dynamically updating software components and distributive environments for collaborative design, service oriented architectures that provide many benefits over software packages alone come to mind. With globally expanding networks and constantly changing software packages, the need for these types of architectures within industry is growing rapidly. New design environments are desired that incorporate security, traceability, and non-repudiation.

When developing these new design environments, fundamental technologies such as system architectures and information infrastructures must be formed to satisfy the needs of security, traceability, and non-repudiation. Security is a must that provides encryption and transmission methods over the information infrastructure, without compromising accountability and integrity. Traceability provides the necessary methods to determine if any unauthorized access to intellectual property has occurred. Non-repudiation protects the interests, both intellectually and financially, of the enterprises and service providers within the design environment. Information sharing of arbitrary data is needed as well to facilitate interoperability between enterprises and server providers.

This middleware solution solved the many problems associated with distributed design environments. It provided a system architecture that included the ability to dynamically and seamlessly incorporate design environments and remote providers for value added services and features. It also provided an information infrastructure that transmitted data and software over the Internet to remote providers, without composing security and traceability.

When solving the aforementioned problems, four core contributions were developed to support the system architecture and information infrastructure. These technologies include:

(1) **Secure JMS**: a secure and robust messaging service

(2) **Verification Services**: a transaction verification and tracking mechanism

(3) **Gateway Services**: a service firewall

(4) **Secure FSS**: a system for secure file sharing and distribution

Each of these technologies was built upon a core java enterprise technology, J2EE. J2EE provided the programming language that facilitated the overall system architecture of the middleware solution. These concepts can be applied to any messaging technology through the use of that technology's programming API. Java was chosen so that JMS could be utilized to its fullest.

The middleware solution was created with the designer in mind. It allowed the designer to access their files from anywhere, at any time, while having access to any features they desired. There were no more dependencies on specific software packages as third party providers now provide the services and features implemented within those packages. The technologies created for the system architecture integrate seamlessly into the designer's current environments, providing maximum efficiency during transitions from the old technology to new ones. There is no longer a need to modify existing software (aside from the menu system for selecting services) or purchase additional software to execute services in house. Services can be added on a pay-per-use basis by management and controlled by that management as well. As a result, features are only paid for when they are needed.

Through development and integration of the individual technologies, a complete and robust design environment solution was designed and implemented. A secure and innovative method of transferring service requests to remote service providers was created, along with providing traceability and accountability within that architecture. A method of dynamic service allocation was implemented so that management could control which services are accessed within an enterprise, from a single point of control.

A brief summarization of each of the fundamental technologies will be listed next, explaining their roles and importance within the system architecture, along with a description of their limitations currently in place. Section 7.2 describes the future work on the solution and technologies, and will go over the necessary items to further each technology and solve some of the limitations that they currently have.

7.1 SECURE JMS: ENABLING SECURE MESSAGING TRANSMISSION

When the need to transfer messages between client enterprises and service providers had to be secured, traceable, and non-reputable, Secure JMS satisfied those requirements by providing methods of transmitting META information and design files within the information infrastructure. To alleviate integration issues and possibly extend other JMS implementations, the operation of Secure JMS occurs through the standard JMS infrastructures already built directly into J2EE. Through the use of AES for encryption, Secure JMS messages are secured to anybody monitoring the network. When using the Verification Server for storage of transactional information, it solved the traceability and non-repudiation issues at hand. Secure JMS is modular in nature and integrates seamlessly into existing JMS frameworks, with only one line of code modification needed.

Limitations

Secure JMS currently has some limitations. One is that the configuration of Secure JMS is restricted to only the values hard coded into the program. These values include database destination, encryption algorithm used, and key length when initializing the algorithm. Also, Secure JMS will not work without verification services and the Verification Server accepting connections, since it uses the Verification Server to store JMS transactional information. Secure JMS has no recovery methods for when network or server failures occur that would prevent a message from being published to a JMS queue. These limitations could render the system impractical for real-time usage. However, addressing these issues is trivial. Section 7.5.1 suggests the future work necessary to solve these limitations.

7.2 VERIFICATION SERVICES: ENABLING TRACEABILITY AND NON-REPUDIATION

Verification Services and the Verification Server played an important role within the system architecture of the middleware solution. It provided storage methods of transactions for traceability and non-repudiation. Through the use of MySQL, an easy JDBC implementation was possible, allowing any type of database command to be executed within the program. Also, all communications with the other technologies were performed over encrypted channels to ensure the highest level of security.

Limitations

There are a few Verification Server limitations that cause it not to perform optimally. The Verification Server uses statically defined configuration values designed specifically for this thesis. Like Secure JMS, the Verification Server has no recovery or rollback methods implemented in the event of a network failure or server error. Section 7.5.2 suggests the future work necessary to solve these limitations.

7.3 GATEWAY SERVICE: ENABLING A SERVICE FIREWALL AND DIRECTORY SERVICE

The Gateway Server provided a method of central protection and control of service access through the use of a software firewall. It also provided a directory mechanism for enterprises to use when authorizing services to their engineers. The Gateway Server was a key technology within the design environment because it centralized all points of contact to the outside world, and was easily managed by management within an enterprise. It enabled the enterprise to have secure access to the Internet and remote providers, from a single point of contact.

Limitations

The Gateway Server too has some limitations limiting its functionality. Currently, the directory portion of the GWS has static directory services, meaning that management cannot edit the list of available services because they are hard coded within the GWS program. Like the previous technologies, all configuration values are statically defined. The GWS also has poor job management, meaning that its internal job assignments are not efficient and they are not meaningful to the programmer or application. Also, its data exchange with the client is not as efficient as it could be. If server errors were to occur, or an interruption of network traffic occurred, the GWS could not recover and the system would hang. There are no program timeouts implemented with the GWS. Section 7.5.3 suggests the future work necessary to solve these limitations.

7.4 SECURE FILE SHARING SYSTEM: ENABLING A SECURE AND DISTRIBUTED FILE SHARING SYSTEM

Secure FSS solved the majority of the problems relating to secure file transfer and secure file storage through integration of several different security features. It also provided many benefits for the designer with regards to the any file, any time, anywhere concepts.

Secure FSS allows the designer to access any file they upload to the system. File integrity and security is maintained through the use of HTTPS to upload the file, and GnuPG to encrypt the file once it is uploaded. The designer can upload their files at any time and from anywhere because Secure FSS is distributive in nature. Secure FSS creates a globally available file sharing system to anyone wishing to deploy it.

The benefits of this new secure file sharing system over other already established systems include being open source, the use of Java as the programming API, and extensibility to many different applications. Integration of several different security issues, redundancy, accessibility, and traceability within one system allow ease of use and simple source and server management.

Limitations

This technology has no limitations, making it the most complete technology created for the middleware solution. There are very minor improvements that could be made, most being added features or improved processes. Unlike the other technologies, Secure FSS does have built in timeouts for server errors or network disruption. However, they could be more efficiently implemented. Section 7.5.4 suggests sore future work available for the Secure FSS.

7.5 FUTURE WORK

Future work on the fundamental technologies created for the system architecture includes adding improvements and additions that were left out of version 1.0. These are insightful to future commerce, and for improvements to the original project requirements and specifications. Ideally, the system needs to be stress tested and deployed in a real life situation to seek out weaknesses and flaws that could cripple the system. Through the stress tests and production environments, the level of security, traceability, and non-repudiation can truly be assessed. In the final subsections below, future work on each individual technology is listed on how it could improve to make the system architecture even more robust with regards to security, traceability, and non-repudiation.

7.5.1 Secure Java Messaging Service

Future work on Secure JMS includes several improvements to its operation to make Secure JMS run more efficiently. One improvement is to implement a way of reading in program configuration through the use of XML files. This will allow Secure JMS to span across several different uses, not just for this middleware solution within design environments. It would also allow dynamic configurations to be used so that the program can initialize according to those configurations.

Another improvement would allow Secure JMS to function without the use of transactions. This can be useful if the security and integrity of JMS messages are desired, but without the use of traceability and accountability. Lastly, Secure JMS needs to implement recovery methods in the event of network failure or server errors. Not all networks, hardware, and software are reliable. Secure JMS needs to account for the few times these technologies become unstable and unreliable.

7.5.2 Verification Server

Like Secure JMS, the Verification Server needs to be able to read in system configuration through XML files. This will allow the technology to be utilized outside the scope of this thesis. The Verification Server also needs to implement recovery methods, similar to Secure JMS. Finally, a more expansive method of verification services could be implemented that would provide services to all users, regardless if they are part of a distributed design environment or not.

7.5.3 Gateway Server

Again, an improvement to the GWS would be to have the ability to read in configuration values through the use of XML files. Like the previous technologies, this would allow for dynamic allocation of resources, depending on how enterprise's infrastructures change.

A method of dynamic editing of allowed services for an enterprise would be ideal; such as a web-based interface that management could log into and perform the appropriate authorizations. The use of XML files could also be used; management could upload an updated XML file into a specified directory that the GWS would read from when initialized, causing the GWS to store the updated list of authorized services. Also, the directory portion of the GWS could be separated into an individual computer. This would allow even more modularity within the system architecture.

A better method of internal job assignments would be a vast improvement, along with statistical reporting. Statistics are a great way to analyze performance and investment returns,

especially when accessing remote services that are paid for as used. The data exchange between engineers and the GWS within the corporate network needs to be improved as well to better handle network or server failures. Implementing timeouts is a good starting point, but the overall protocol should be rewritten to be as efficient and effective as possible.

7.5.4 Secure File Sharing System

Although this technology is the most complete of them all, some improvements could be made. The option to choose different types of encryption methods would be good for companies that wish to use their own encryption, or perhaps another type of standard encryption algorithm. Also, the redundancy within Secure FSS could be improved. Redundancy is currently implemented with a queuing system that could be designed more efficiently.

7.6 FINAL REMARKS

A secure information infrastructure for service oriented architectures was researched, designed, and implemented to provide high levels of security, a robust amount of traceability, and non-repudiation. This complete and integrated system can be implemented and utilized in today's distributed design environments through the use of such systems as CAD systems and Eclipse. This system can also apply to future distributive, collaborative design efforts by extending its technologies through further development.

The fundamental technologies created within the middleware solution have the potential to reach out to other programming constructs and be used by any field that utilizes security, JMS for messaging, and file systems for distributed storage. The potential of the problems associated with security, traceability, and non-repudiation can be overcome through the use of these technologies developed and implemented by the middleware solution defined within this thesis.

APPENDIX

SUN JAVA SYSTEM APPLICATION SERVER SCREEN SHOTS

In this appendix are several screen shots that display the different JMS connection factories, resources, and destinations used within the demonstration of the system architecture. Refer to these figures for setup and administration as described in Section 6.2.

JMS CONNECTION FACTORIES

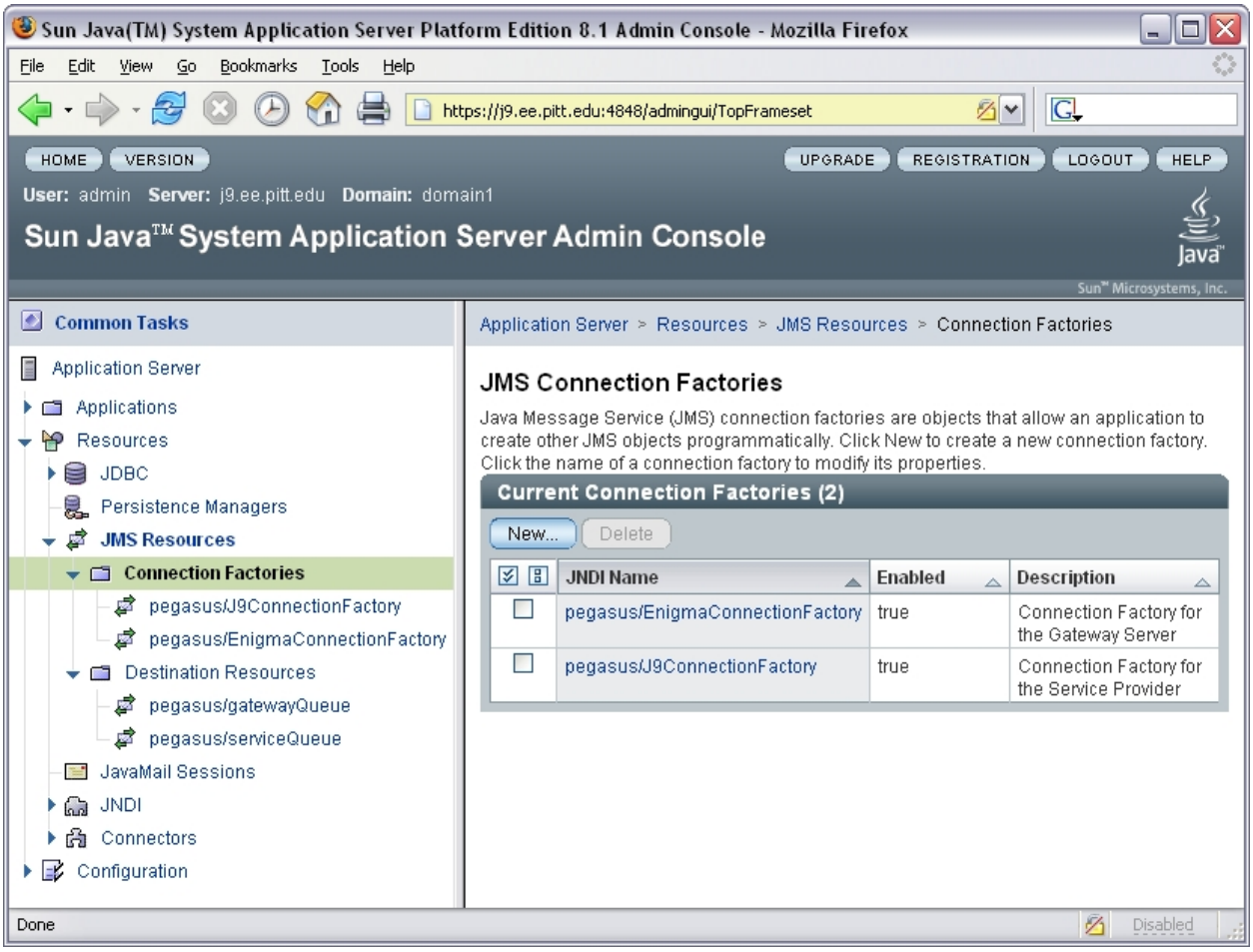


Figure 28. JMS Connection Factories

Service Provider

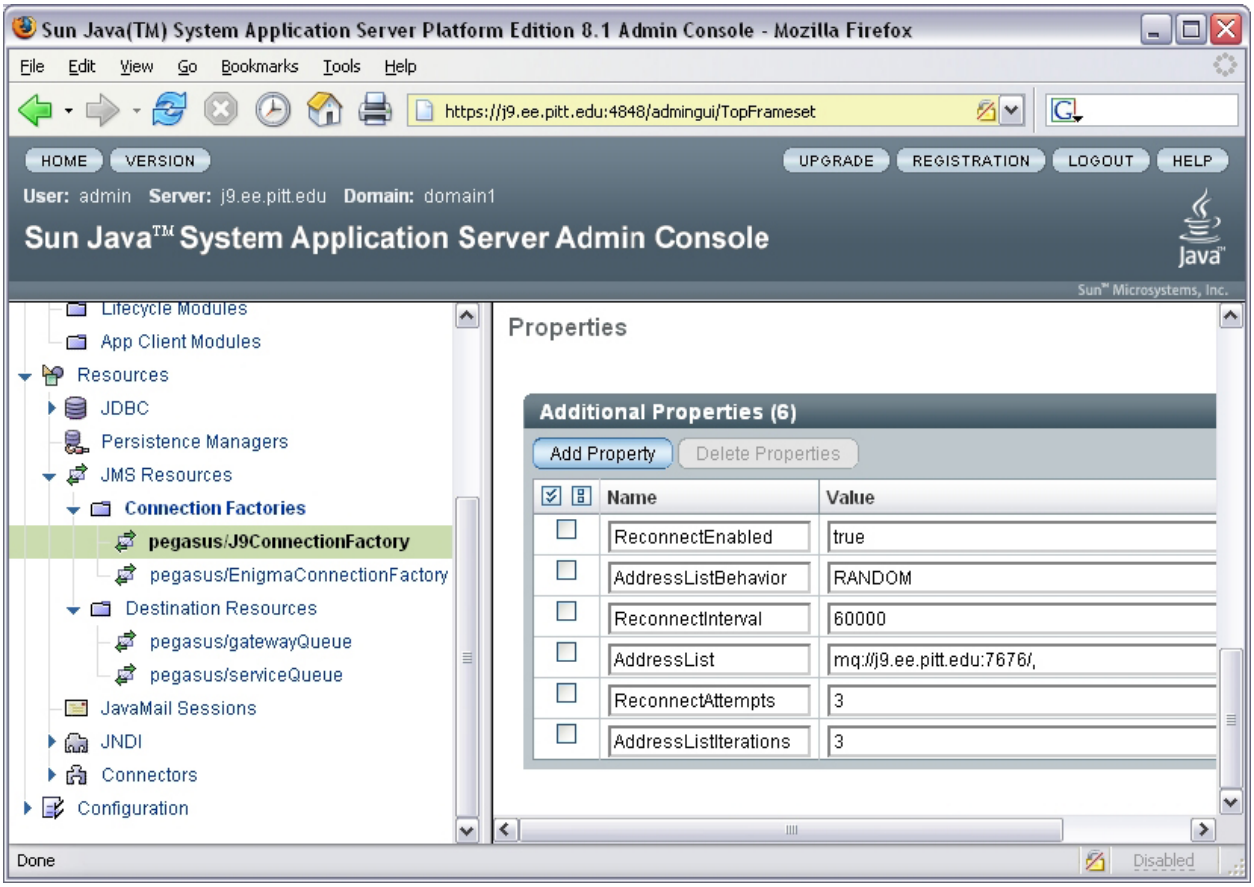


Figure 29. Service Provider Connection Factory Setup

Gateway Server

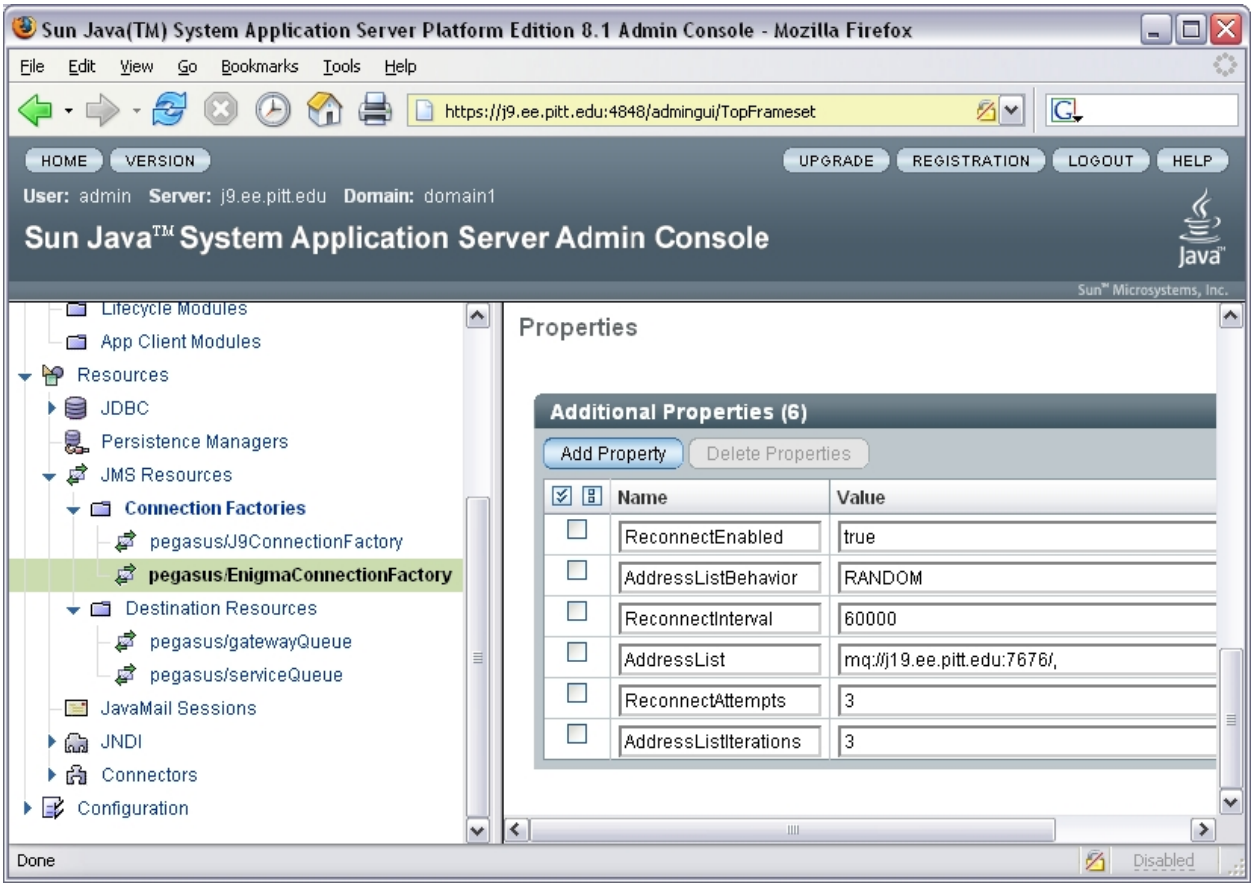


Figure 30. Gateway Server Connection Factory Setup

JMS PHYSICAL DESINTATIONS

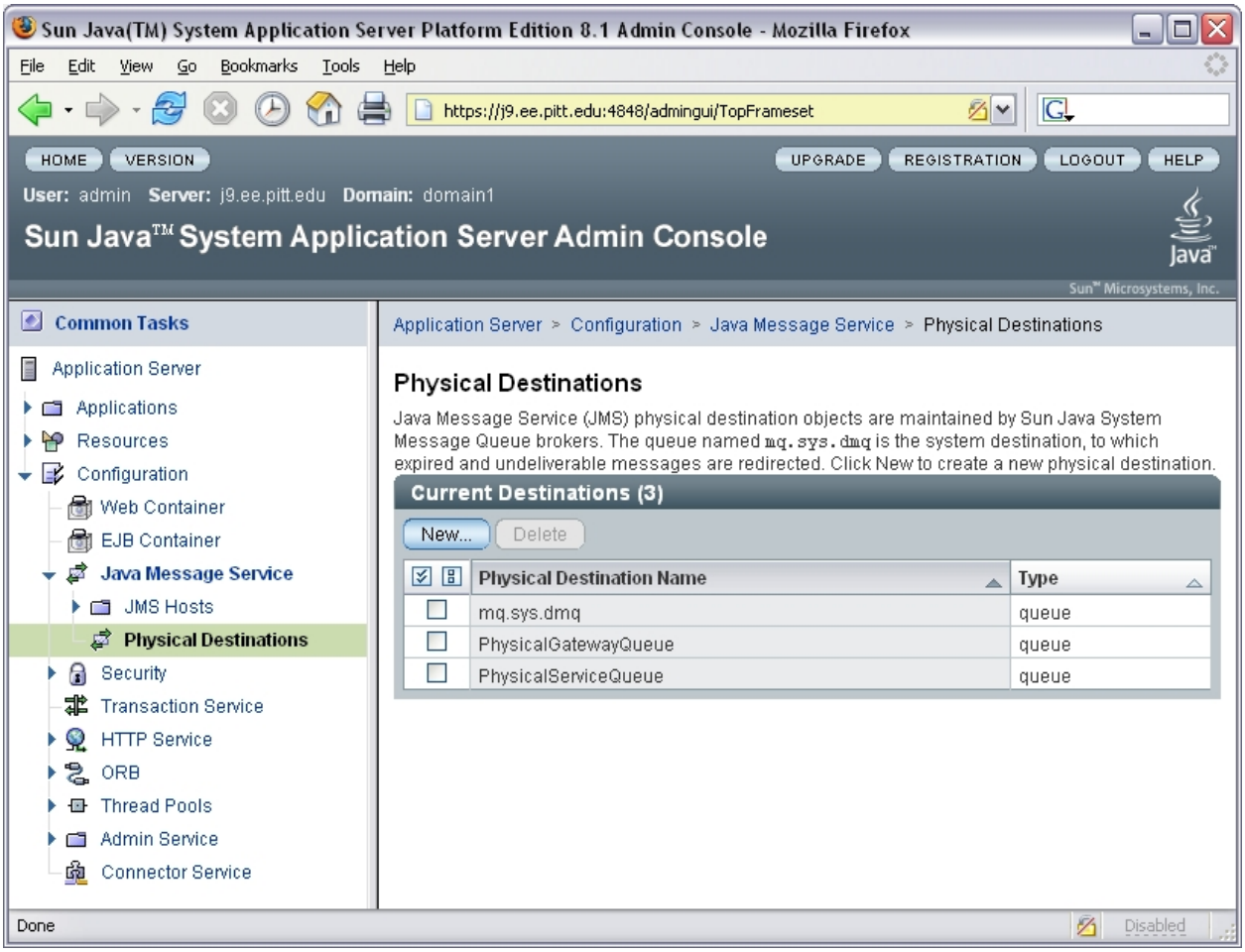


Figure 31. JMS Physical Destinations

JMS DESTINATION RESOURCES

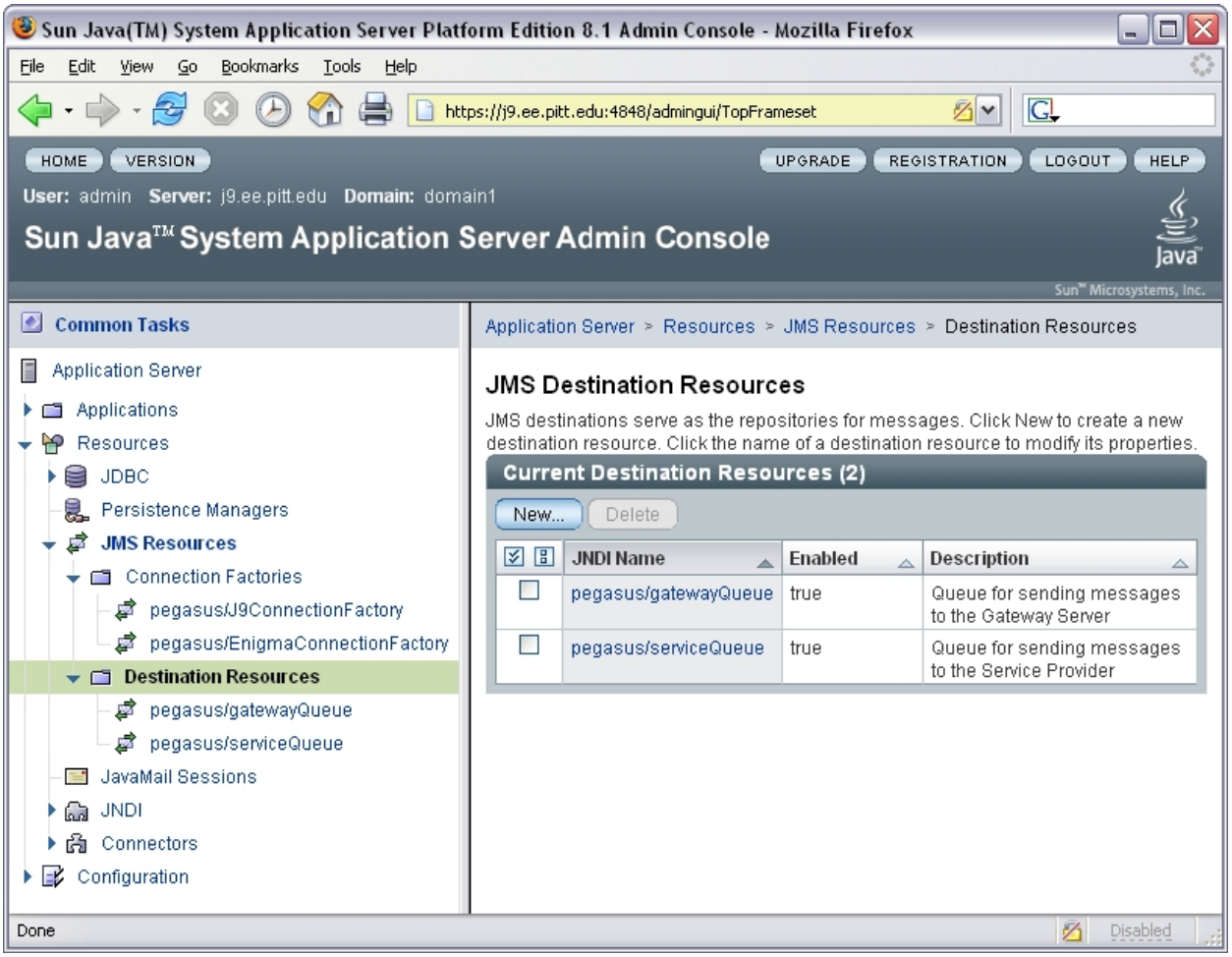


Figure 32. JMS Destination Resources

Gateway Server

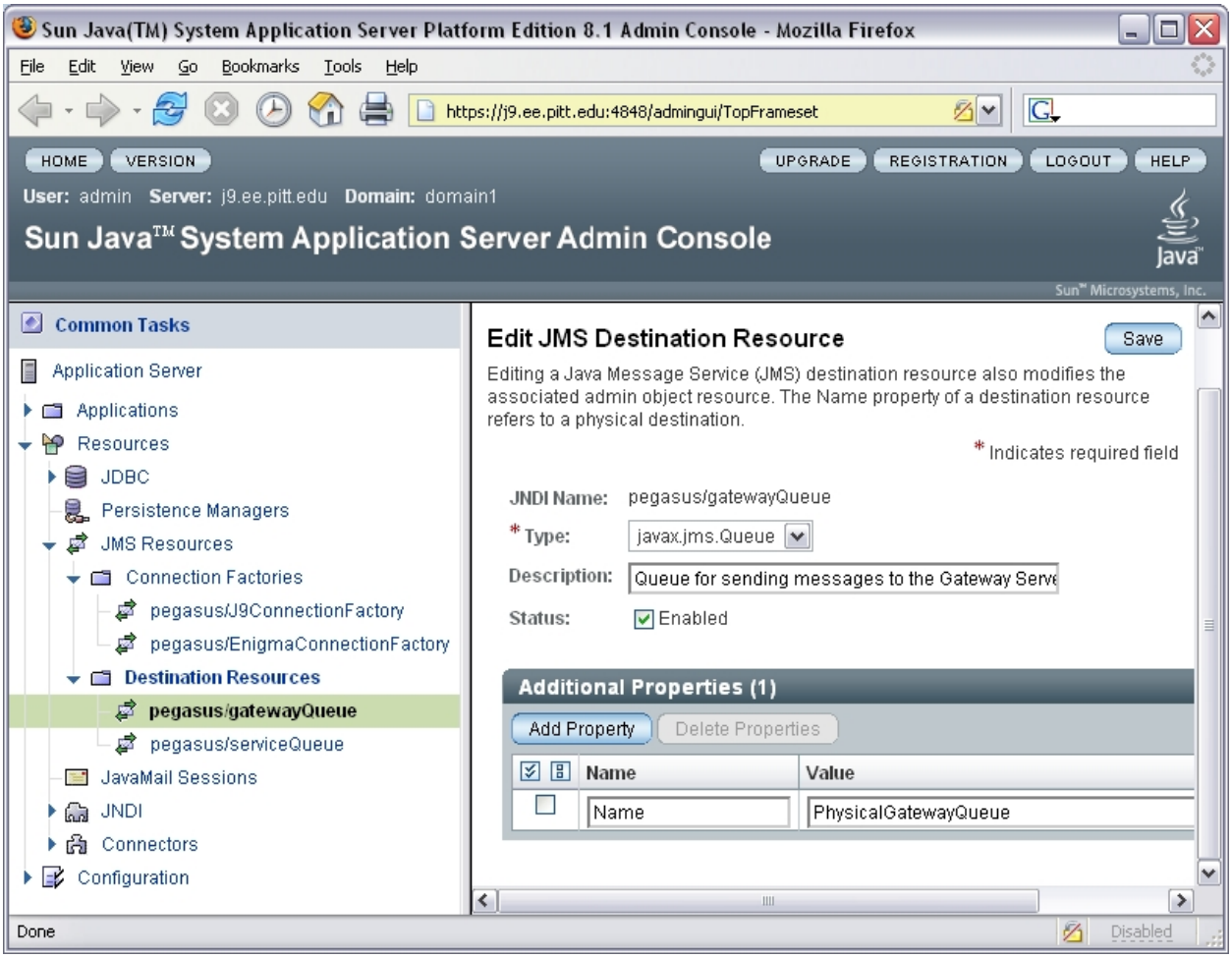


Figure 33. Gateway Server Queue Setup

Service Provider

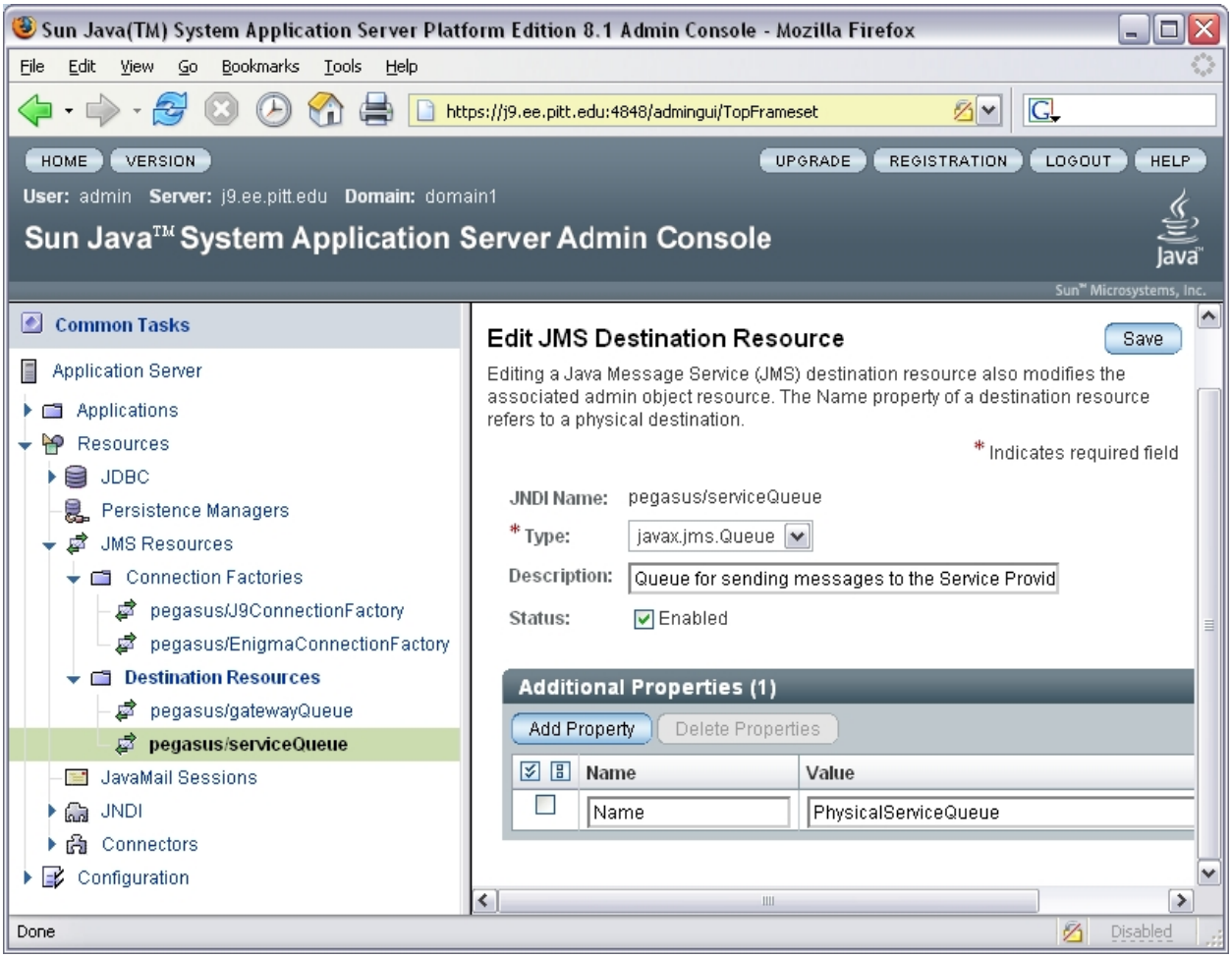


Figure 34. Service Provider Queue Setup

BIBLIOGRAPHY

[1] Niels Ferguson and Bruce Schneier, “Practical Cryptography”, Indianapolis: Wiley Publishing, Inc., 2003.

[2] “COM: Component Object Model Technologies”, http://www.microsoft.com/com/, Accessed March, 2006.

[3] “About Eclipse”, http://www.eclipse.org/org/, Accessed March, 2006.

[4] L. Koved, A. Nadalin, N. Nagaratnam, M. Pistoia, “Enterprise Java™ Security: Building Secure J2EE™ Applications”, Addison Wesley Professional, 2004.

[5] Hicks, Michael, “DYNAMIC SOFTWARE UPDATING”, Dissertation in Computer and Information Science, http://www.cis.upenn.edu/~mwh/papers/thesis.pdf, Accessed March, 2006.

[6] “Overview of the Distributed File System Solution in Microsoft Windows Server 2003 R2”, http://technet2.microsoft.com/WindowsServer/en/Library/d3afe6ee-3083-4950-a093-8ab748651b761033.mspx, Accessed March, 2006.

[7] Hall, Eric, “Internet Core Protocols: The Definitive Guide“, O'Reilly, 2000.

[8] “RFC 2616 – Hypertext Transfer Protocol -- HTTP/1.1”, ftp://ftp.isi.edu/in-notes/rfc2616.txt, Accessed March, 2006.

[9] “Specification for the ADVANCED ENCRYPTION STANDARD (AES)”, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, Accessed March, 2006.

[10] “Graphviz - Graph Visualization Software”, http://www.graphviz.org/, Accessed March, 2006.

[11] “RFC 2440 – OpenPGP Message Format”, http://www.ietf.org/rfc/rfc2440.txt, Accessed March, 2006.

[12] “The J2EE 1.4 Tutorial”, http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html, Accessed March, 2006.

[13] Snader, Jon, “VPNs Illustrated: Tunnels, VPNs, and IPsec”, Addison Wesley Professional, 2005.

[14] Castify Networks SA. “Product Information Sheet” Available on-line: www.mailonator.com

[15] Xythos Software Inc. “Xythos WebFile Server Data Sheet” Available on-line: www.xythos.com

[16] ZipLip, Inc. “ZipLip: Secure File Collaboration R4 Data Sheet” Available on-line: www.ziplip.com

[17] J. Howard et. al., "Scale and Performance in a Distributed File System," ACM Transactions on Computer Systems, Vol. 6. Issue 1, 1988, pp. 51-81.

[18] Austin Che, “Designing and Implementing Kiwi: A Secure Distributed File System over HTTPS”, Computer Science Department, Stanford University, May 2001.

[19] Ying Liu, “CSFS - A Secure File System”, Masters Thesis, Computer Science Department, Concordia University, Montreal, Quebec, Canada, September 2003.

[20] “RFC 1321 – The MD5 Message-Digest Algorithm“, http://rfc.net/rfc1321.html, Accessed March, 2006.

[21] Squid Web Proxy Cache, FAQ, http://www.squid-cache.org/Doc/FAQ/FAQ_long.html, Last Accessed March, 2006.

[22] “RFC 1436 – The Internet Gopher Protocol (a distributed document search and retrieval protocol“, http://www.rfc-archive.org/getrfc.php?rfc=1436, Last Accessed March, 2006.

[23] “CISCO PIX 525 SECURITY APPLIANCE – Data Sheet”, http://www.cisco.com/en/US/products/hw/vpndevc/ps2030/products_data_sheet09186a0080091b09.html, Last Accessed March, 2006.

[24] MySQL Database Application, http://www.mysql.com/, Last Accessed March, 2006.

[25] Code File System, http://www.coda.cs.cmu.edu/, Last Access March, 2006.