

California State University, San Bernardino

**CSUSB ScholarWorks**

---

Theses Digitization Project

John M. Pfau Library

---

2004

## **A secure lightweight currency service provider**

Chih-Wen Hsiao

David Turner

Keith Ross

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Information Security Commons](#)

---

### **Recommended Citation**

Hsiao, Chih-Wen; Turner, David; and Ross, Keith, "A secure lightweight currency service provider" (2004). *Theses Digitization Project*. 2594.

<https://scholarworks.lib.csusb.edu/etd-project/2594>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact [scholarworks@csusb.edu](mailto:scholarworks@csusb.edu).

A SECURE LIGHTWEIGHT CURRENCY SERVICE PROVIDER

---

A Project  
Presented to the  
Faculty of  
California State University,  
San Bernardino

---

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

---

by  
Chih-Wen Hsiao  
March 2004

A SECURE LIGHTWEIGHT CURRENCY SERVICE PROVIDER

---

A Project  
Presented to the  
Faculty of  
California State University,  
San Bernardino

---

by  
Chih-Wen Hsiao  
March 2004

Approved by:

  
\_\_\_\_\_  
Dr. David Turner, Chair, Computer Science

2/5/2004  
Date

  
\_\_\_\_\_  
Dr Richard John Botting

  
\_\_\_\_\_  
Dr. Ernesto Gomez )

© 2004 Chih-Wen Hsiao

## ABSTRACT

The Lightweight Currency Protocol (LCP) was originally proposed to solve the problem of fairness in resource cooperatives. However, there are other possible applications of the protocol, including the control of spam and as a general-purpose medium of exchange for low-value transactions. Therefore, LCP currency can be used for both automated agent-based trading, and also for trading performed directly by humans. This project investigates the implementation issues of the LCP, and also investigates LCP bank services to provide human interface to currency operations.

This project provides two prototype systems: a prototype of a currency issuer, and a prototype of a bank service. The currency issuer is a web service that provides only machine-accessible interface, while the bank service is a web application that allows users to register for bank services, and begin transacting in lightweight currencies.

To make LCP transactions secure, issuers and holders of currency must possess public/private key pairs, which they use for authentication and establishment of secure communication channels. Unfortunately, typical users are not capable of securely managing a public/private key pair

in their local machines. To solve this problem, this project demonstrates how a bank service can be provided so that users only need password access to their currency accounts. In this manner, a local hardware failure does not result in the user losing access to her accumulated wealth.

## ACKNOWLEDGMENTS

I would like to thank Dr. Richard John Botting and Dr. Ernesto Gomez for being my committees and their insightful suggestions on my proposal and report. I am particularly thankful to my project advisor Dr. David Turner. Without his suggestions, guidance, and corrections of design, implementation, and report I wouldn't even get a chance to complete the project.

Special gratitude goes to my mother Mrs. Chen who gives me full support to help me go abroad to continue my education without fear.

The support of the National Science Foundation under award 9810708 is gratefully acknowledged.

## TABLE OF CONTENTS

ABSTRACT .....	iii
ACKNOWLEDGMENTS .....	v
LIST OF TABLES .....	ix
LIST OF FIGURES .....	x
CHAPTER ONE: INTRODUCTION	
1.1 The Lightweight Currency Protocol .....	1
1.1.1 Transfer Messages .....	2
1.1.2 Verification Messages .....	2
1.1.3 Message Counter .....	3
1.1.4 Bank System .....	3
1.2 Purpose of Project .....	4
1.3 Scope of Project .....	4
1.3.1 Phase I .....	4
1.3.2 Phase II .....	5
1.4 Organization of Chapters .....	5
CHAPTER TWO: TECHNOLOGIES	
2.1 Java .....	6
2.2 Java Servlet and JSP .....	7
2.2.1 Java Servlet .....	7
2.2.2 JSP .....	8
2.3 Java Applet .....	8
2.4 Struts Framework .....	9
2.4.1 Model-View-Controller .....	9
2.4.2 Overview of Struts .....	9

2.5 Database Connection Pooling .....	11
2.6 iBATIS Database Layer Framework .....	12
2.6.1 SQL Maps .....	13
2.6.2 Conclusion .....	13
2.7 PostgreSQL .....	14
CHAPTER THREE: SECURITY	
3.1 Public/Private Key Mechanism .....	15
3.2 Secure Socket Layer Protocol .....	17
3.2.1 SSL Handshake Type One .....	18
3.2.2 SSL Handshake Type Two .....	19
3.3 HTTPS Protocol .....	20
3.4 Login Authentication .....	21
3.5 Applet Security .....	21
3.6 Database Security .....	22
CHAPTER FOUR: PROJECT DESIGN	
4.1 Overall Description .....	23
4.1.1 Project Perspective .....	23
4.2 Project Architecture Design .....	25
4.2.1 The Architecture .....	25
4.2.2 Description .....	26
4.3 Bank System Design .....	28
4.3.1 Use Case Diagram .....	28
4.3.2 Graphical User Interface and Description .....	28
4.3.3 Database Design .....	36

4.4 Currency Issuer Design .....	37
4.4.1 Request and Response Message Format Design .....	37
4.4.2 Database Design .....	42
CHAPTER FIVE: PROJECT IMPLEMENTATION	
5.1 Project Class Design .....	44
5.1.1 Bank System Class Design .....	44
5.1.2 Lightweight Currency Client Module Class Design .....	48
5.1.3 The Currency Issuer Classes Design .....	50
5.2 Project Classes Implementation .....	51
5.2.1 Bank System .....	52
5.2.2 Currency Client Module Classes Implementation .....	67
5.2.3 The Currency Issuer Classes Implementation .....	70
CHAPTER SIX: FURTHER ENHANCEMENT AND CONCLUSION	
6.1 Related Projects .....	73
6.2 Further Enhancement .....	74
6.3 Conclusion .....	75
APPENDIX A: XML FILE .....	76
APPENDIX B: JSP FILE .....	88
APPENDIX C: JAVA SOURCE CODE .....	98
APPENDIX D: ACRONYMS AND ABBREVIATIONS .....	154
REFERENCES .....	156

LIST OF TABLES

Table 4.1. Software Interfaces ..... 24

## LIST OF FIGURES

Figure 2.1.	The Difference between CGI and Servlet .....	7
Figure 2.2.	Struts Architecture .....	10
Figure 2.3.	JDBC and Database Connection Pooling .....	12
Figure 2.4.	Flow of SQL Maps .....	14
Figure 3.1.	Flow of Public/Private Key Mechanism .....	16
Figure 3.2.	Flow of SSL Handshake Type One .....	18
Figure 3.3.	Flow of SSL Handshake Type Two .....	20
Figure 4.1.	Architecture of the Project .....	26
Figure 4.2.	Bank System Use Case Diagram .....	29
Figure 4.3.	Login Page .....	30
Figure 4.4.	Registration Page .....	31
Figure 4.5.	Default Main Page .....	32
Figure 4.6.	Profile Page .....	33
Figure 4.7.	Account Balances Page .....	34
Figure 4.8.	External Transfer Fund Page .....	35
Figure 4.9.	Verification Request Page .....	36
Figure 4.10.	Bank System ER Diagram .....	37
Figure 4.11.	Transfer Request Message Format .....	39
Figure 4.12.	Verification Request Message Format .....	39
Figure 4.13.	Transfer Response Message Format .....	40
Figure 4.14.	Success Verification Response Message .....	41

Figure 4.15.	Failure Verification Response Message .....	41
Figure 4.16.	Currency Issuer ER Diagram .....	43
Figure 5.1.	Client Module Class Diagram .....	49
Figure 5.2.	Currency Issuer Class Diagram .....	51
Figure 5.3.	The Account Class .....	52
Figure 5.4.	The AccountDAO Class .....	53
Figure 5.5.	The BaseDAO Class .....	53
Figure 5.6.	The ChatServer Class .....	54
Figure 5.7.	The CheckAccountBalanceAction Class .....	55
Figure 5.8.	The DAOException Class .....	55
Figure 5.9.	The EditProfileAction Class .....	55
Figure 5.10.	The ExternalTransferAction Class .....	56
Figure 5.11.	The LoginAction Class .....	56
Figure 5.12.	The LogoutAction Class .....	56
Figure 5.13.	The LogoutHandler Class .....	57
Figure 5.14.	The RegistrationAction Class .....	57
Figure 5.15.	The RequestHandleServlet Class .....	58
Figure 5.16.	The RequestType Class .....	59
Figure 5.17.	The TransferFundAction Class .....	59
Figure 5.18.	The TransferFundHandler Class .....	60
Figure 5.19.	The TransferRequestBean Class .....	61
Figure 5.20.	The TransferResultBean Class .....	62
Figure 5.21.	The TrustPartner Class .....	62
Figure 5.22.	The TrustPartnerDAO Class .....	63

Figure 5.23. The User Class .....	63
Figure 5.24. The UserDAO Class .....	64
Figure 5.25. The UserService Class .....	65
Figure 5.26. The VerificationRequestAction Class .....	65
Figure 5.27. The VerificationRequestBean Class .....	66
Figure 5.28. The VerificationResultBean Class .....	66
Figure 5.29. The ParserResult Class .....	67
Figure 5.30. The RequestHandler Class .....	67
Figure 5.31. The ServerConnectionHandler Class .....	68
Figure 5.32. The TransferResult Class .....	68
Figure 5.33. The VerificationResult Class .....	69
Figure 5.34. The XMLHandler Class .....	69
Figure 5.35. The Client Class .....	70
Figure 5.36. The ClientConnectionHandler Class .....	71
Figure 5.37. The ParserResult Class .....	71
Figure 5.38. The SSLSimpleServer Class .....	72
Figure 5.39. The XMLHandler Class .....	72

## CHAPTER ONE

### INTRODUCTION

#### 1.1 The Lightweight Currency Protocol

A peer-to-peer (P2P) resource market is a market where computer systems can trade their surplus resources such as network bandwidth, storage and computer computation. Lightweight Currency was designed as a trading means for the market; furthermore, compared to real world currencies, Lightweight Currency has less cost, risk, and complexity.

The Lightweight Currency Protocol (LCP), created by Dr. David A. Turner in CSU San Bernardino and Dr. Keith W. Ross in Polytechnic University, is a simple, secure protocol that enables entities with network connections to exchange their Lightweight Currencies through public key identifiers. In a P2P resource market, each entity acts in at least one of these two roles -- currency issuer or currency holder. A currency issuer is a web service that allows connecting clients to authenticate as their public keys, and then to transfer currency they hold to another public key. A currency holder is a seller or buyer who sells his/her resources to get more currencies or

purchases other users' resources by spending his/her currencies.

The protocol simply contains two request and two response messages.

#### 1.1.1 Transfer Messages

1.1.1.1 Transfer Request Message. A resource buyer sends a transfer request message to the currency issuer. The message consists of the public key of the resource seller to receive the funds, a unique transaction ID, the amount to be transferred, and a message counter.

1.1.1.2 Transfer Response Message. After the currency issuer completes the process of the transfer request, it will send a transfer response message back to the applicant. The message contains the result and the fee for processing this request.

#### 1.1.2 Verification Messages

1.1.2.1 Verification Request Message. A resource seller sends a verification request message to the currency issuer to check if the buyer deposited agreed upon funds in the seller's account. The message contains a transaction ID to specify a unique deposit record.

1.1.2.2 Verification Response Message. After the currency issuer gets a verification request, it sends a verification response message back to the client. For a

success case, the message contains the amount of the specific deposit and the fee charged to process this request. For a failure case, the message contains the failure reason and the fee for the verification request.

#### 1.1.3 Message Counter

Both transfer and verification request messages contain message counters. The purpose of the message counter is to allow retransmission of the message in the case that network failure or a man-in-the-middle attack results in the sender's inability to determine whether a request message has arrived or not. If a sender didn't receive the response message, the sender can resend the same request message without fear that the currency issuer processes the same request message twice or more. In this case, the currency issuer will recognize repeat request messages, and simply return the correct response again, but will not take any other action.

#### 1.1.4 Bank System

Basically, a user can communicate with the currency issuer by a standalone application with LCP protocol. However, a user could not manage his/her currencies by using any computer except his/her own pc. Therefore, a web application that acts as a bank system is a solution for this problem. A bank system is a secure and trusted

intermediary between currency holders and issuers. Usually, the system offers similar functionalities as a standalone application. A user only needs a browser like Microsoft IE or Netscape Navigator to handle all his currencies. However, all data are transmitted under a private and secure channel between the user and her bank system.

### 1.2 Purpose of Project

The main purpose of this project is to build a bank system that offers a friendly and simple interface to let users easily manage their lightweight currencies. Also, the bank system constructs a highly secure environment, therefore, a user can process any request without fear of outside attacks.

### 1.3 Scope of Project

The project was separated into two phases:

#### 1.3.1 Phase I

The first phase is related to the interface between the user and the bank system. The bank system offers several functions for users. First, a user can manage his/her own profile such as modify his/her password and e-mail address. Second, the system integrates a secure chat system. On the system, a user can chat with other

users or negotiate a deal with someone else. Third, a user can check his/her account balances and process an internal transfer operation to send funds to any other user in the same bank system. Fourth, a user can transfer currency to entities outside of her bank, and also receive currency from them.

### 1.3.2 Phase II

The second phase is related to the interface between the bank system and a currency issuer. The bank system communicates with a currency issuer using LCP protocol. When a user wants to transfer currency to another user on different bank system, the system needs to contact the currency issuer to complete the transaction. After receiving the reply from the currency issuer, the system will send an appropriate message to the user.

## 1.4 Organization of Chapters

Chapter Two introduces and summarizes technologies used in this project. Chapter Three describes the security mechanisms and ways to make the system more secure. Chapter Four discusses the details of project design. Chapter Five focuses on the project implementation. The last chapter presents the related projects and future work.

## CHAPTER TWO

### TECHNOLOGIES

Web application is an alternative to installing applications on users' computers. A user doesn't need to cope with any software and hardware conflicts. The only thing a user needs is just a browser like Microsoft IE or Netscape Navigator. A browser acts like a magician, which displays different application interfaces. In this case, a user can achieve his/her wants without any troubleshooting and installation on his/her computer. The main issues are: how to reduce development time, improve performance, and build a flexible web application. Next, I will briefly introduce the technologies I used in this project.

#### 2.1 Java

Java is one of the most popular programming languages in the world. Programmers use Java to ensure their applications can migrate onto different operating systems such as Microsoft Windows, Linux family, etc. without re-writing much of the code.

## 2.2 Java Servlet and JSP

### 2.2.1 Java Servlet

A CGI program is stored and executed on the web server in response to a request from a user. Servlets are the Java Technologies' answer to CGI programming. They are Java programs which run on the server side JVM (JAVA Virtual Machine) and generate dynamic content. Figure 2.1 shows the difference between Java Servlet and CGI. In CGI, the system creates a new child process for each request to run a CGI program. This method wastes finite server resources and limits the number of requests that server

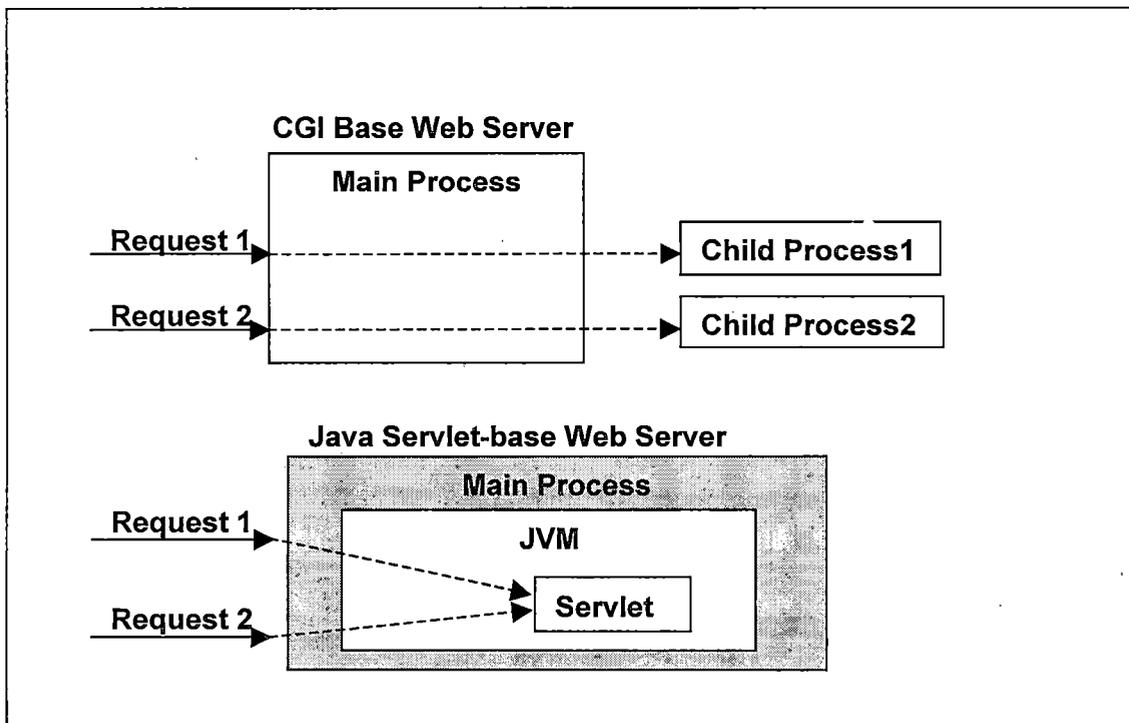


Figure 2.1. The Difference between CGI and Servlet

resources and limits the number of requests that server can handle simultaneously. Also, it causes some performance loss because the server has to reload the CGI program again and again. On the other hand, the server only generates a Java Servlet program once. Each request starts a new thread in the running Servlet. Compared to CGI, Java Servlets are more powerful, efficient, and scalable.

#### 2.2.2 JSP

JSP is an extension of Java Servlets technology. Typically, a Java Servlet can do the same tasks as JSP, however, JSP makes it easy to mix static HTML with Java code.

### 2.3 Java Applet

Applets are Java programs that run inside browsers, which provide visual display inside the browser and have the ability to interact with server. An applet is downloaded on a user's machine when the network connection is established between the client and server. Compared to Servlets, an applet runs on client side machine not on the server. In this project, I use an applet to be a chat system interface and a message box to accept messages from the server.

## 2.4 Struts Framework

### 2.4.1 Model-View-Controller

The Model-View-Controller design paradigm [1] breaks an interactive object into three main components: the model, the view, and the controller. A controller accepts input or data from users then sends it to the model or view components for processing. A model is a component that manages information, typically, the model stands data and maintain state. The model passes results to the view component. The only task for the view component is to generate the visual display representation. Because of the independence of each component, the controller or the view can be modified without any change to the model component.

### 2.4.2 Overview of Struts

Struts [1] is one of the best Java open source web application developing frameworks. It is based on the MVC design paradigm. The architecture of Struts is shown in Figure 2.2. Basically, Struts uses ActionServlet to be its controller component and JSP (Java Server Page) to generate the dynamic response output. In the model component, Struts doesn't offer much because many other frameworks including Enterprise JavaBeans [7] and Java Data Objects [1] already deal well with business logics. Therefore, Struts doesn't limit developers to any

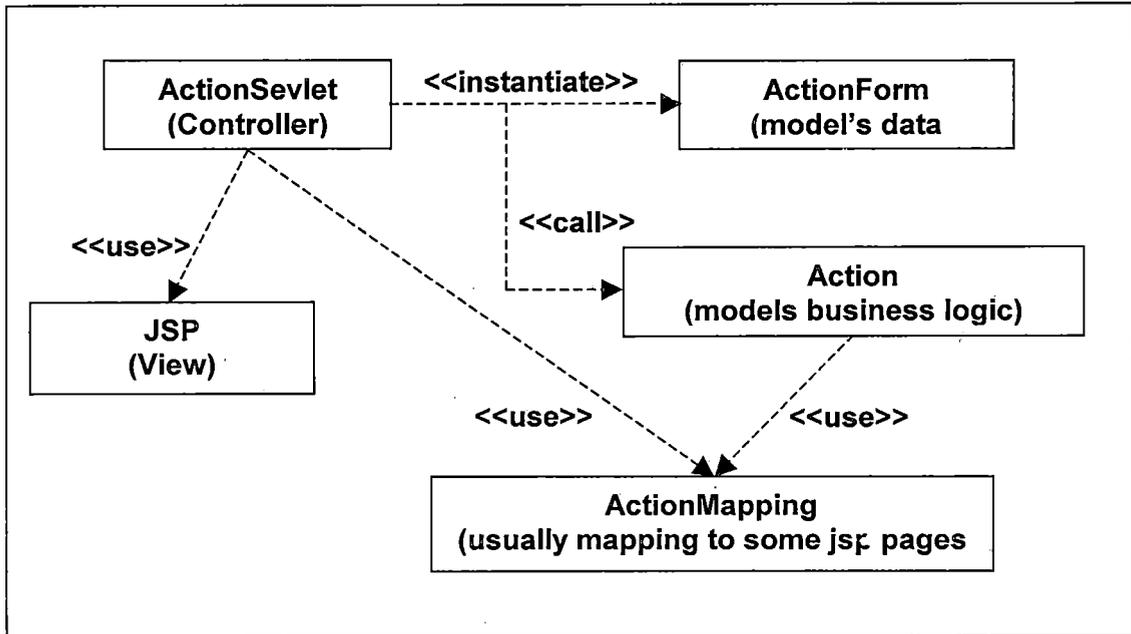


Figure 2.2. Struts Architecture

particular model implementation. Next, I will briefly describe DynaActionForm and Validation.

2.4.2.1 Overview of DynaActionForm. It is a useful class, which generates a dynamic form bean object. In the normal case, a developer needs to create different form bean objects to deal with each form. However, in Struts, a developer just specifies a form format in an XML file. Then, the form can be used in the program without writing any other code.

2.4.2.2 Overview of Validation. Struts offers both server side and client side validations. Furthermore, a developer just needs to specify the check condition in an

XML file. Then Struts automatically handles the validation for you.

2.4.2.3 Conclusion. The Struts framework offers a well-defined and powerful API such as DyanActionForm, Validation and so on. Also, it can interact with a lot of popular frameworks. The framework reduces the development time and code size, and helps programmers to build a robust and flexible web application.

## 2.5 Database Connection Pooling

The mechanism of database connection pooling is to build a connection pool in the middle layer between the web server and the database server. The connection pool has permanent connections with the database and accepts requests from web server to access the database. For every new request comes in, the pool just reuses the connection object. Thus, the connection pool reduces the resources used by the web server and reduces the overhead when used in place of the acquiring database connections through the DriverManager class. Although the database interactions are usually quite short, usually in a web server, the number of database requests is large, so the performance loss becomes bigger and bigger. Figure 2.3 shows the differences between normal JDBC programs and database

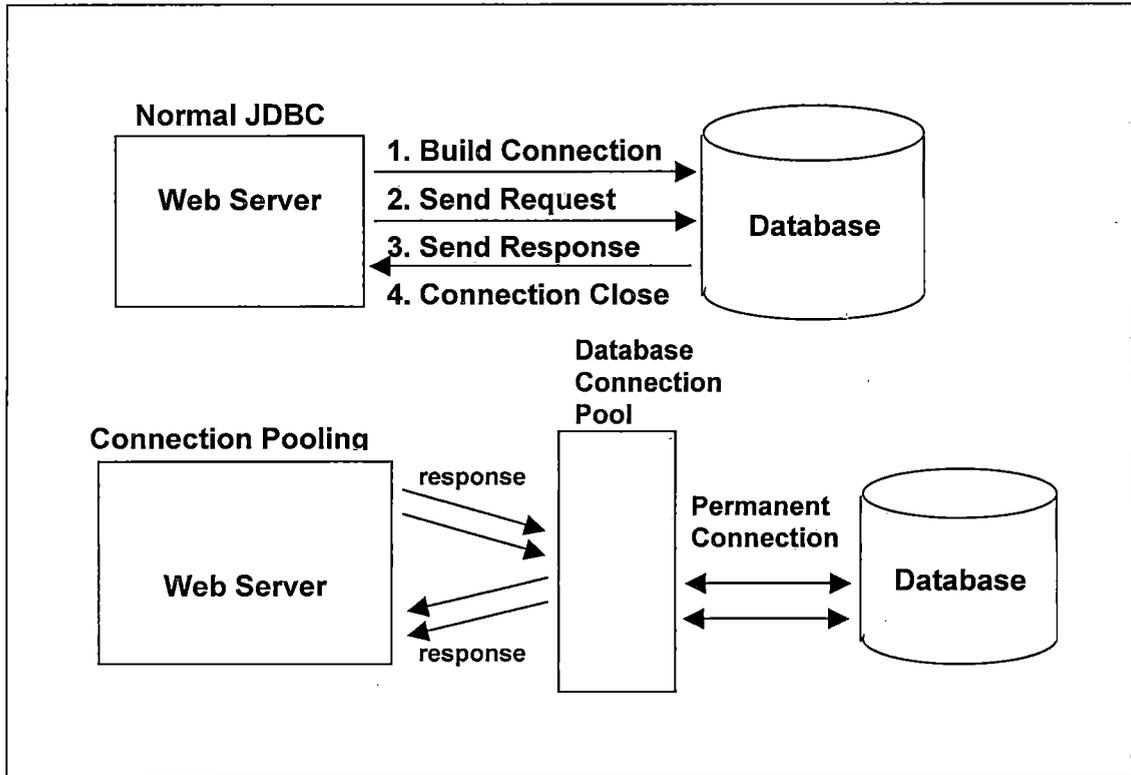


Figure 2.3. JDBC and Database Connection Pooling

connection pooling technology. Because the performance issue is very important for a web application, the project uses this technology to improve the database access performance and save some more resources for the server.

## 2.6 iBatis Database Layer Framework

iBatis Database Layer [2] was created by Clinton Begin (<http://www.ibatis.com>). The framework helps one design and implements better persistent database layers for Java applications. SQL Maps and Data Access Objects are two main components in this framework. In this

project, I use SQL Maps and same philosophy as Data Access Objects.

### 2.6.1 SQL Maps

The SQL Map API lets developers map JavaBean objects directly to PreparedStatement parameters and ResultSets. The creator of iBATIS said that the philosophy behind SQL Maps is to "provide a simple framework to provide 80% of the functionality of JDBC in 20% of the code." SQL Maps uses XML descriptors to map JavaBeans onto a JDBC PreparedStatement. Also, the configuration of XML descriptor files is pretty simple. A developer just creates an XML file (sql-map-config.xml) with database settings and other XML files which contain the function names that map to the method we call in the Java class and the corresponding SQL scripts. Figure 2.4 shows how SQL Maps work.

### 2.6.2 Conclusion

iBATIS Database Layer framework includes many convenient utilities and is easy to use. Also, the framework works well with Struts and Servlets. It helped me to better organize the class design and reduce the code size in the project.

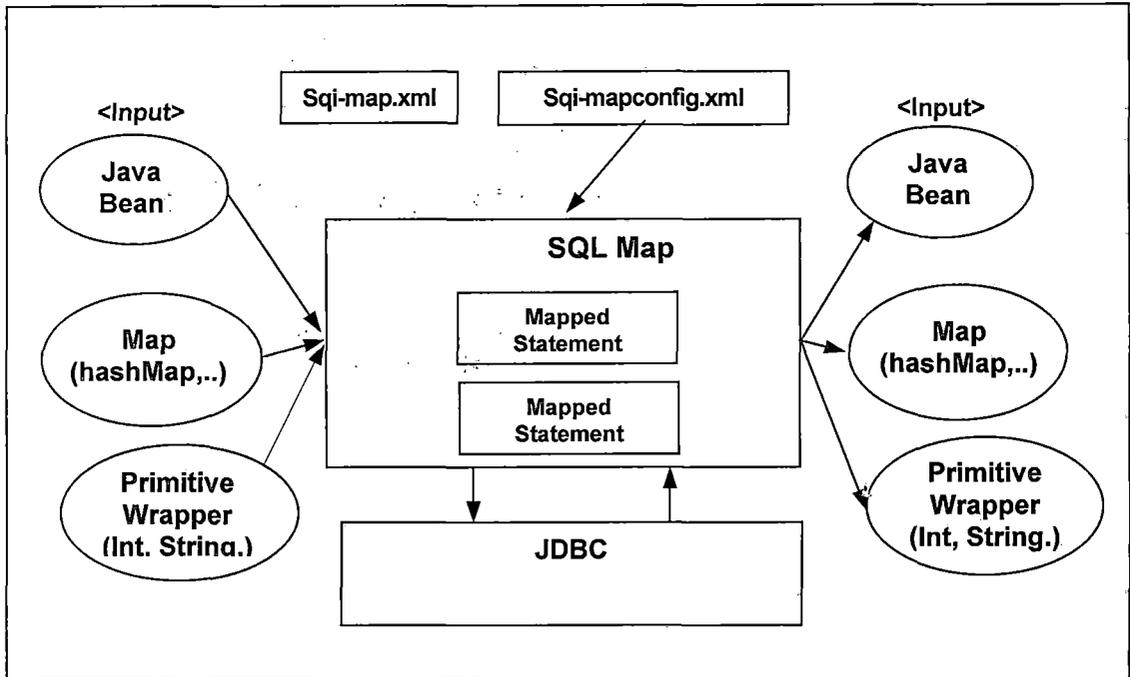


Figure 2.4. Flow of SQL Maps

## 2.7 PostgreSQL

Both PostgreSQL and MySQL are very popular open source database management systems. Although MySQL is faster, PostgreSQL is more standard, strict, and powerful. For instance, MySQL doesn't support the foreign key mechanism and transactions aren't well supported. Thus, PostgreSQL is the better solution to build a reliable and integrated database system.

## CHAPTER THREE

### SECURITY

Protecting transmitted data on the internet is a critical issues. Lightweight currency for each entity in the P2P resource market is like money in the real world. Protecting it is one of the main purposes of this project. Otherwise, a hacker would probably spend your currencies without your agreement and knowledge. In this chapter, I will describe the security mechanisms I used in this project to protect all transfer of data through the network and also the data stored in the database.

#### 3.1 Public/Private Key Mechanism

Figure 3.1 shows the flow of the public/private key mechanism. There are several tools to help you build the public and private key pairs by using different algorithms such as RSA, DSA, etc. These algorithms generate the asymmetric key pairs. If the key size is large enough like 1024 bit or more, the decrypted text by the mechanism is almost impossible to be decrypted. That means even if a hacker catches the packets in the Internet, he will not be able to access the data, because he doesn't have your private key. However, this mechanism is not prefect. The most significant disadvantage of this mechanism is the

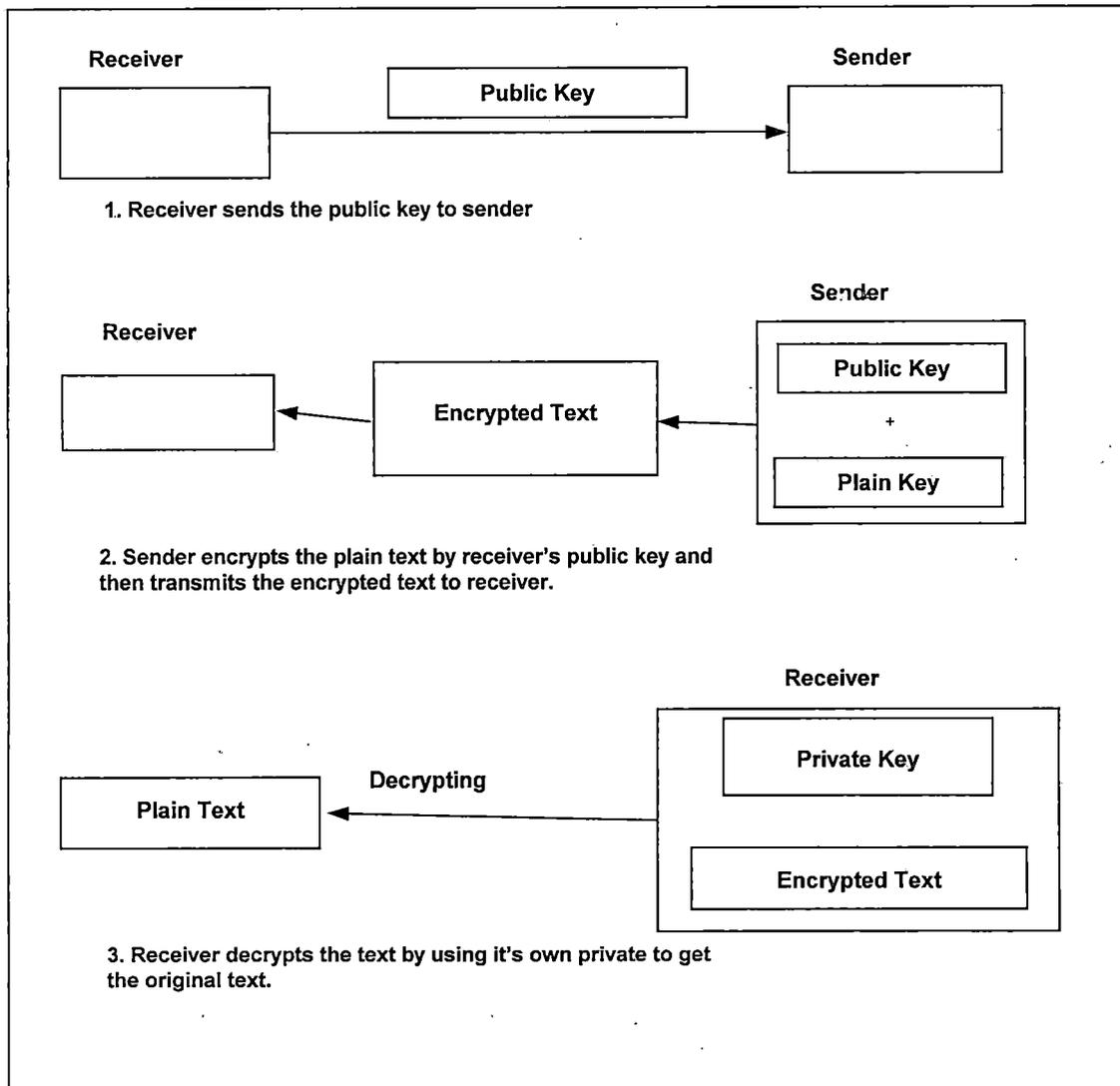


Figure 3.1. Flow of Public/Private Key Mechanism

performance problem because the encryption and decryption processes consume too much time by using asymmetric key pairs. If every message transferred between the starting point and the destination adopts the mechanism, the performance loss is obvious. Therefore, the SSL is the alternative solution that I will describe in section 3.2.

In the project, I used Java keytool to generate the certificate, which contains the public and private key pairs and some information related to the entity. Later, the certificate will be used in the SSL handshake process.

### 3.2 Secure Socket Layer Protocol

The secure Socket Layer (SSL) protocol provides one of the most commonly available security mechanisms on the Internet. Both client and server communicate in a private and secure channel sharing secret symmetric keys until the end of the connection. All transmitted data in the connection will be encrypted and decrypted by these keys, which are only held on the client and server. Compared to the asymmetric key cryptography algorithm, the symmetric key is faster. However, how can client and server receive and share the key pairs? Actually, the key pairs were generated in the SSL handshake process before establishing the secure channel. There are two types of SSL handshake processes, and I used both of them on different parts of this project. I will now describe these two types of SSL handshake processes. I also explain why I need two types of SSL handshakes in the project.

### 3.2.1 SSL Handshake Type One

The flow of SSL handshake type 1 is shown Figure 3.2. In the beginning, the client sends a client hello message to which the server has to respond with a server hello message. Both hello messages are used to establish security enhancement capabilities between client and server. After that, the server sends its certificate to the client and then the client checks if the certificate exists in the truststore, which is a file containing the trust certificate lists. If the client trusts the server's certificate, they will exchange the CipherSpec. Finally,

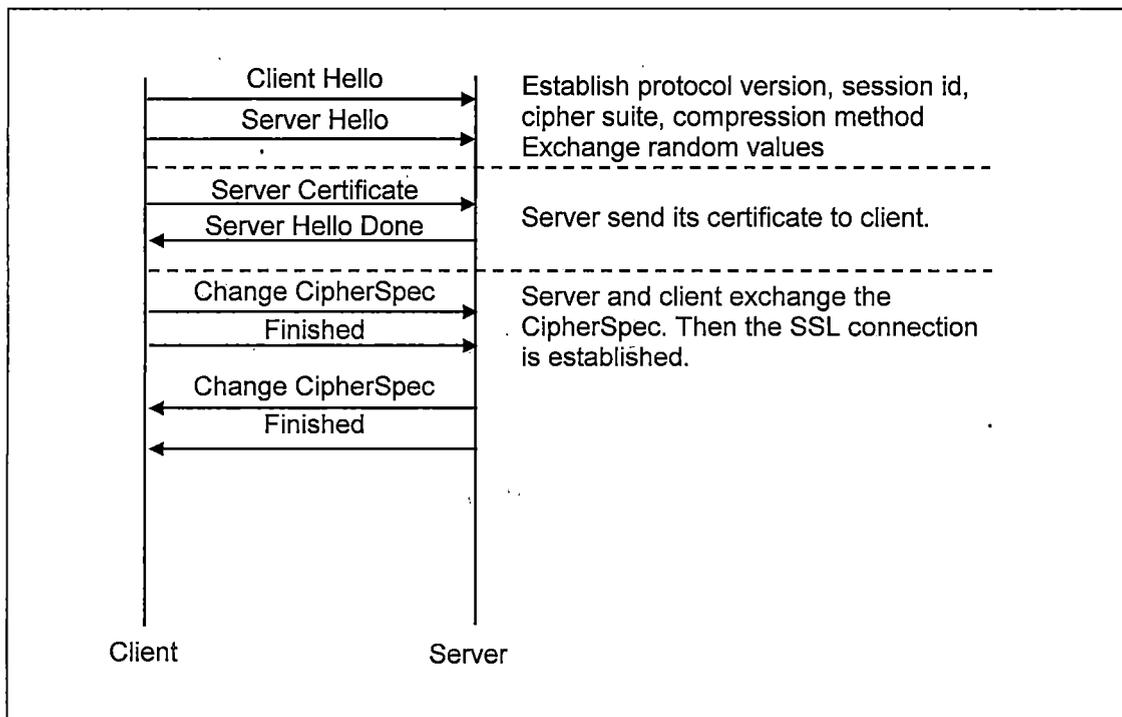


Figure 3.2. Flow of SSL Handshake Type One

they design the key generation algorithm and key pairs used in the secure connection. In the project, a user who uses the bank system is unable to provide a certificate. Therefore, the handshake type 1 is using between the user and the bank web server.

### 3.2.2 SSL Handshake Type Two

The flow of type 2 is shown in Figure 3.3. The only difference between type 1 and type 2 is that in type 2 the server requests the client certificate to authenticate. In this type, if the server can't trust the client certificate or the client can't send his/her certificate, it causes a fatal error, and the connection will be terminated. In the project, the currency issuer uses the public key to identify the client and execute the client request such as transfer or verify deposit. The only way to get the public key of the client is through the SSL handshake type 2. Because type 2 provides the client's certificate, the currency issuer can extract from it the client's public key. Otherwise, the currency issuer has no way to locate the client account.

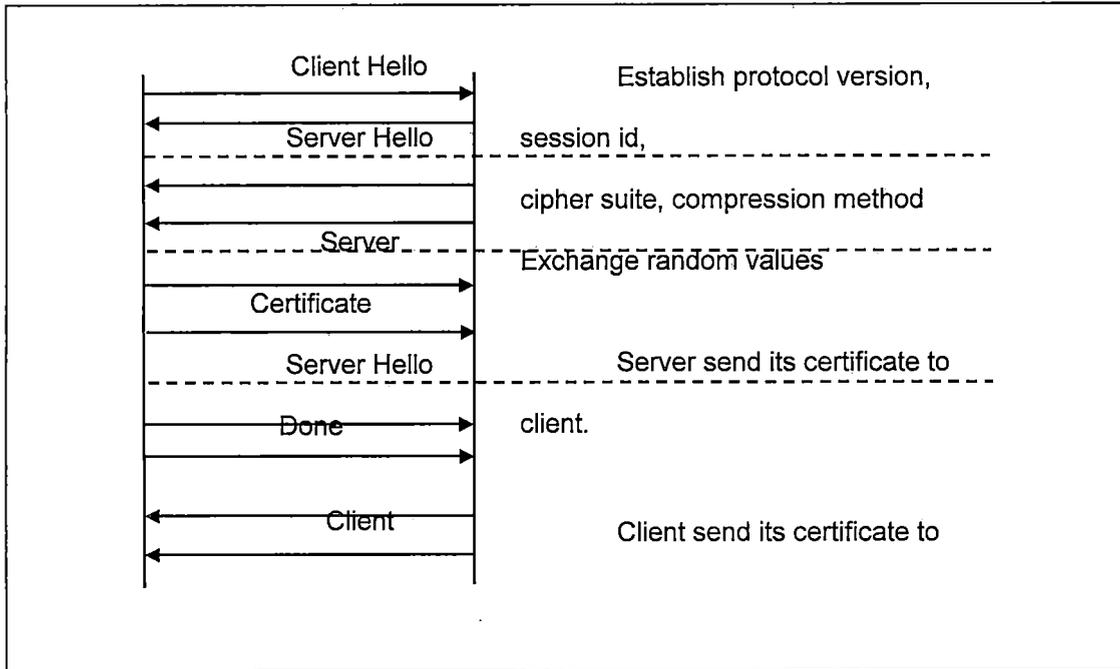


Figure 3.3. Flow of SSL Handshake Type Two

### 3.3 HTTPS Protocol

HTTP is the most popular protocol on the Internet. Basically, a client sends an HTTP request message to the web server. Then, the web server sends an HTTP response message back to the client. Unfortunately, both request and response messages are readable. That means that someone can catch and read these messages without your permission. HTTPS is an encrypted version of HTTP. The secure connection is built by running standard HTTP connection on the top of an SSL connection. Therefore, these messages are no longer in plain text, and only

client and server owns the key to decrypt these messages. Except for this, HTTPS is basically same as HTTP.

### 3.4 Login Authentication

The login authentication is the most basic security mechanism for web servers so far. A user has to use his/her user name and password to login to the bank system to management his/her currencies. According to the protection of HTTPS, a hacker won't get the user name and password on the Internet, unless he finds the solution to decrypt the messages that he caught.

### 3.5 Applet Security

In this project, I use an applet to interact with the server and other users. However, it is possible that another person "in the middle" between client and server loads the applet that server supposes to send to client. Then the person can pretend to be the client to steal some currency or information. To solve this problem, I use a random number. Once a user logs in the bank system, he/she will get a random number. It is difficult for a hacker to get or guess the number. Even if he loaded the applet, the hacker can't get the applet to run properly on his computer.

### 3.6 Database Security

In this project, I just take the simplest mechanism to protect the database. The database is only allowed requests from the localhost. No one can access the database through a remote computer.

CHAPTER FOUR  
PROJECT DESIGN

4.1 Overall Description

4.1.1 Project Perspective

4.1.1.1 System Interface. In this project, there are two system interfaces. The first interface is located between the bank user and the bank system, and it is accomplished through HTML documents over the HTTPS protocol. The second interface is the one between the bank system and the currency issuer, and it is accomplished through XML documents over LCP protocol over SSL protocol. The database management system is developed by using PostgreSQL on both bank system and currency issuer. HTTPS, XML, Java, Java Servlets, Java Applets, Java Server Page (JSP), Socket paradigm, Struts framework, iBATIS Database Layer framework and ANT were all used to implement the system interface of the bank system on the Linux operating system. In addition, all source codes are reusable for further related projects.

4.1.1.2 Hardware Interface. Because the Linux operating system already handles the hardware interfaces for this project. Thus, no implementation relates to hardware interfaces in this project.

4.1.1.3 Software Interface. Both bank system and currency issuers' software interfaces are provided by using Linux operating system, Tomcat 4.1.29 for Java Server Page (JSP), Java Servlets, Struts framework, and iBATIS framework, J2SE 1.4.2 and Apache Ant 1.5.3 for compiling Java source codes, and PostgreSQL for developing the database management system. Besides, a bank system user needs to access the bank system by using web browser such as Microsoft IE and Netscape. The Table 4.1 shows the summarization of software interfaces.

Table 4.1. Software Interfaces

Software	System
Operating System	Red Hat 9.0
Web Browser	Microsoft IE Netscape
Web Server Container	Tomcat 4.1.29
Java Compiler	J2SE 1.4.2
Java-based build tool	Apache ant 1.5.3
Database Management System	PostgreSQL 7.3

4.1.1.4 Assumptions and Dependencies. In this project, there are following assumptions:

It is assumed that there are two bank systems named bank1 and bank2 running on different machines with different IP addresses.

In each bank system, there are several registered users and most of them possess some lightweight currencies in their accounts. Also, a bank system allows new users to create new accounts.

It is supposed that there are two currency issuers named issuer1 and issuer2 running on different computers with different IP addresses.

In each currency issuer, each bank system has an account. Also, a currency issuer allows a bank system to transfer funds to any entity with different public key identifiers, even if the entity doesn't possess an account on the issuer database.

## 4.2 Project Architecture Design

### 4.2.1 The Architecture

The whole project is separated into two web services: bank system and currency issuer. The architecture of the project is shown in Figure 4.1. Three main components are presented: the bank system user, the bank system, and the currency issuer.

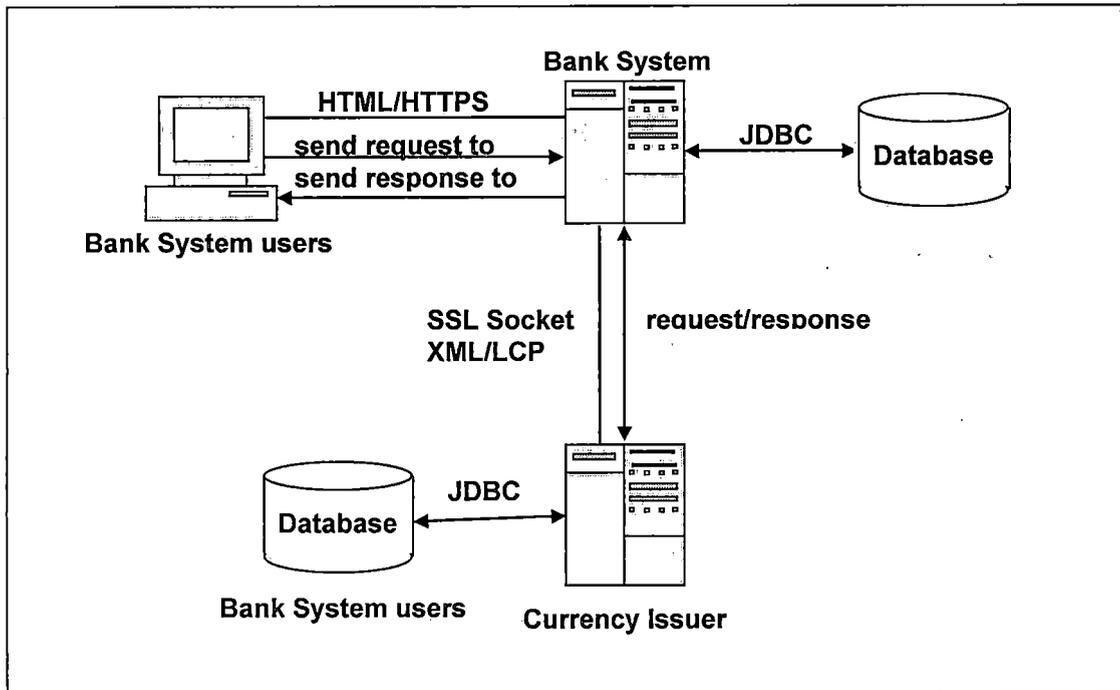


Figure 4.1. Architecture of the Project

#### 4.2.2 Description

The components in the Figure 4.1 are described as follows:

- A bank system user is a person who uses a Web browser to manage his/her personal profile and currencies.
- A bank system is a web service provider that executes its clients' requests. If the request is just an internal request that the bank system can handle itself, the system will send back the response to the user immediately. Otherwise, the

bank system needs to contact the currency issuer before it can report a result to the user.

- A currency issuer is a Lightweight Currency service that processes outside requests and stores entities' currencies. In this project, the entities are the bank systems.

A bank system's life cycle is shown as follows:

- A bank system user goes to a browser, fills out an HTML form with request information, and clicks the "submit" button. The request message will be sent to the bank system as an HTML document over the HTTPS protocol.
- The bank system takes and analyzes the request message. If it is an internal request message, the bank system will process the request, modify the database, and then send a response back to the user. If it is an external request message, the bank system will generate an XML document following the LCP format, open an SSL socket connection with a currency issuer, and then send the XML document to the issuer.
- Once the bank system gets the response from the currency issuer, the system will modify the

database, and then send the result back to the user.

A currency issuer life cycle is described as follows:

- The currency issuer takes the entity's request message, which is an XML document over LCP.
- In order to extract the information in the XML document, the currency issuer parses the document, analyzes the request, modifies the database, converts the response message to an XML document, then sends it back through the SSL socket connection.

### 4.3 Bank System Design

#### 4.3.1 Use Case Diagram

Figure 4.2 shows the Use Case Diagram for the bank system. The system offers different interfaces for different operations to users.

#### 4.3.2 Graphical User Interface and Description

4.3.2.1 Login Page. When a user types the bank URL to connect the bank web server, the first page is the login page. It allows a user to type a userid and password to authenticate the user. If the input data is incorrect,

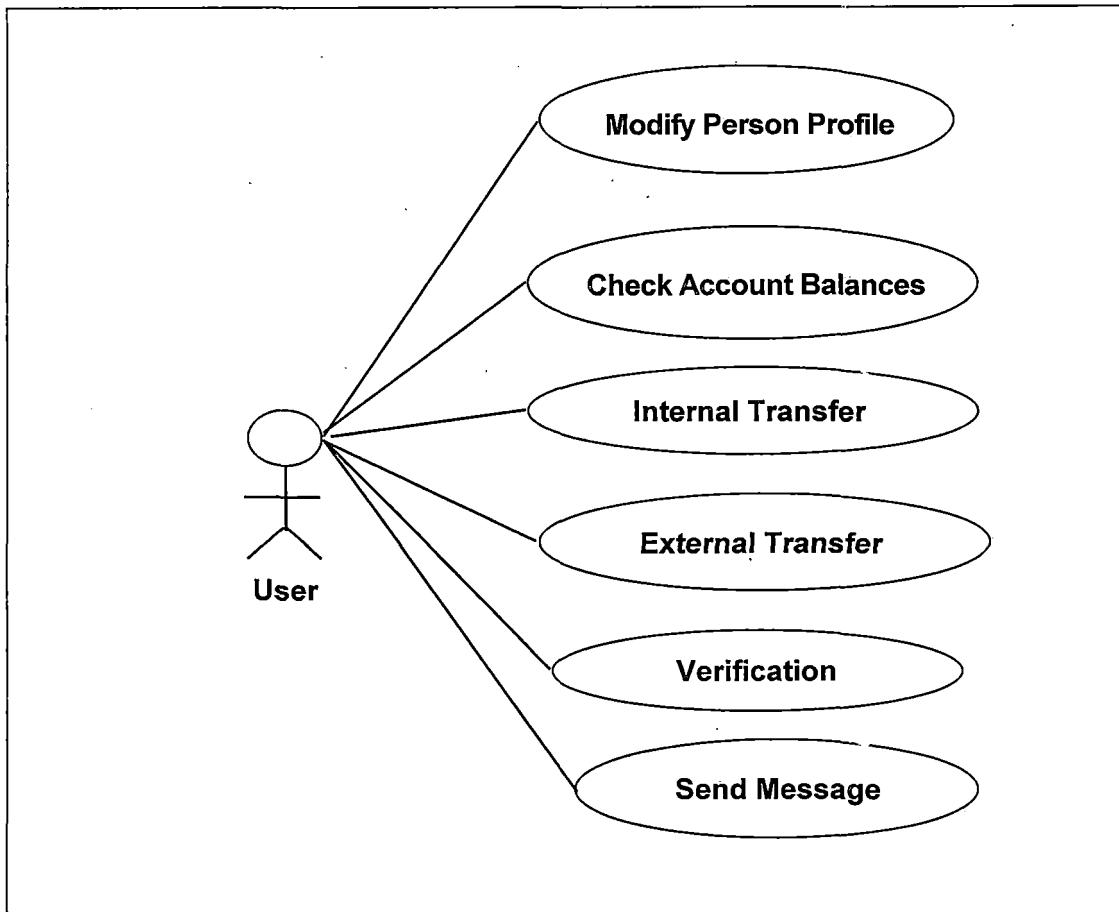


Figure 4.2. Bank System Use Case Diagram

Struts framework's validation will generation an error display message in the same page. If the data matches the database record, the page will forward to the main page. Also, the page contains two links. One is a registration page link and the other is an information page. The graphic interface of login page is shown in Figure 4.3.

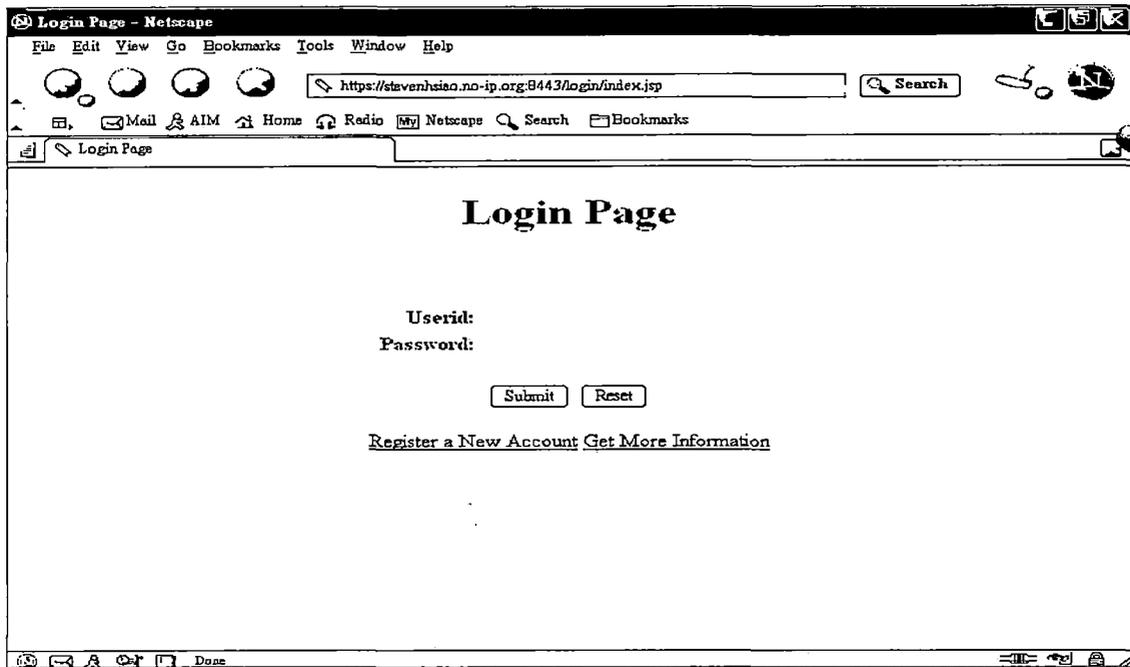


Figure 4.3. Login Page

4.3.2.2 Registration Page. The graphic interface for the registration page is shown in Figure 4.4. A new user needs to fill up the form to register a new account. If the input data is incorrect, for example, the userid is too short or password is not matched, then the browser will display an appropriate error message. Otherwise, the bank system will generate an account for this user and forward to the main page.

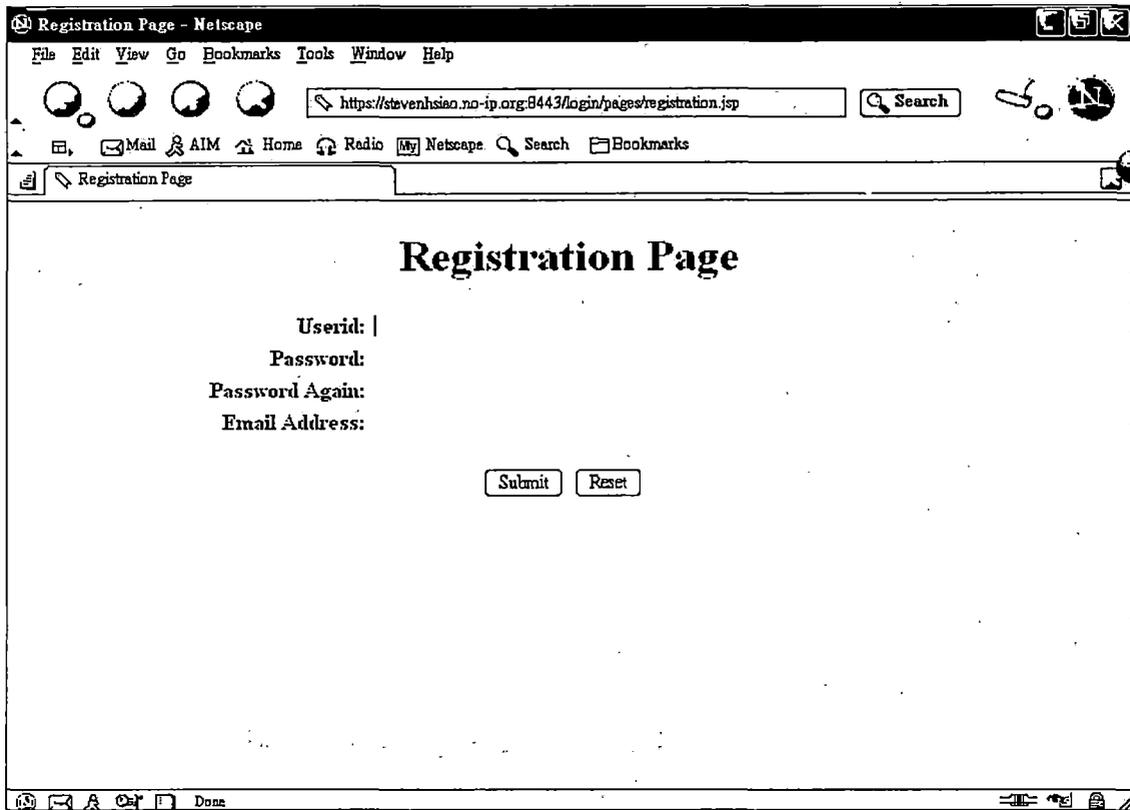


Figure 4.4. Registration Page

4.3.2.3 Main Page. See Figure 4.5. The main page is separated into three frames. The left frame shows the list of available functions. The right frame shows the particular function's graphical interface. The default right frame is the Transfer Fund page. The bottom frame is an applet, which is a chat interface and also a message box that displays instant messages from the bank server.

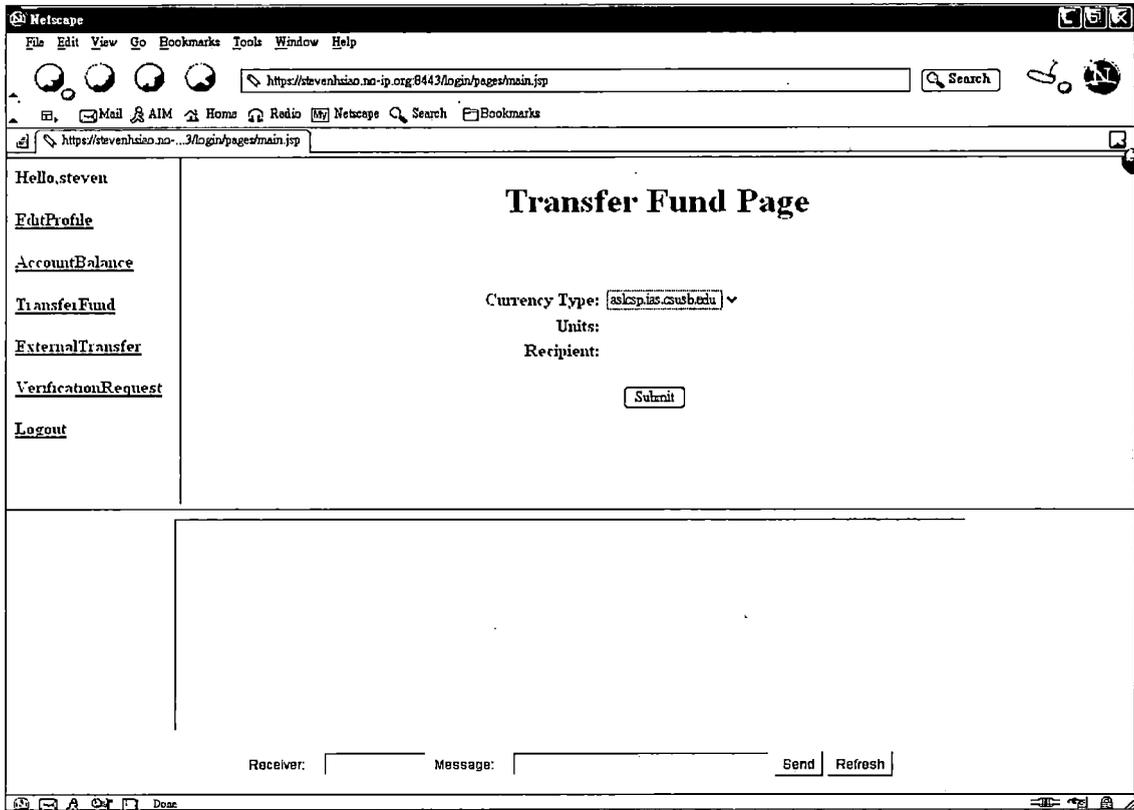


Figure 4.5. Default Main Page

4.3.2.4 Transfer Fund Page. The graphical interface of the internal transfer function is shown in Figure 4.5. A user specifies the currency type, amount, and a recipient userid, then clicks the "submit" button. The system will process the user's request and display a proper response. Also, a response message sent from the system will show on the recipient's applet window.

4.3.2.5 Profile Page. The Profile page is pretty simple. The system allows a user to modify his/her password and e-mail address. Then, the system modifies the

information in the database. Once the system completes the process, it forwards the user to the default page. The graphic interface is shown in Figure 4.6.

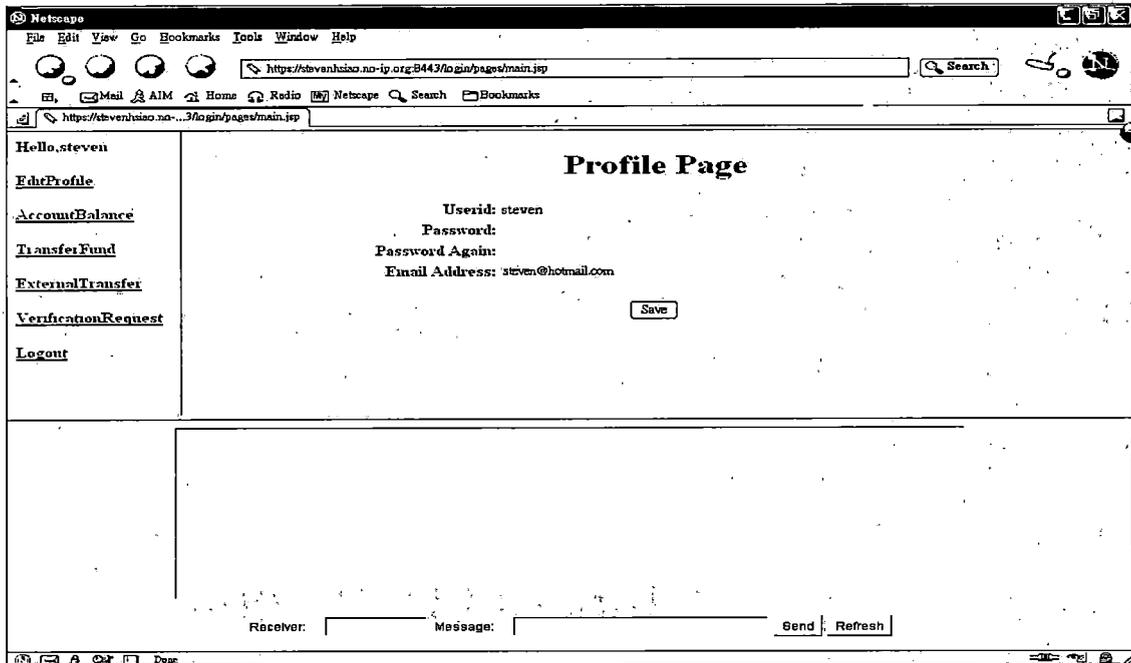


Figure 4.6. Profile Page

4.3.2.6 Account Balances Page. The display of this page is shown in Figure 4.7. It simply lists the currencies that the user owns.

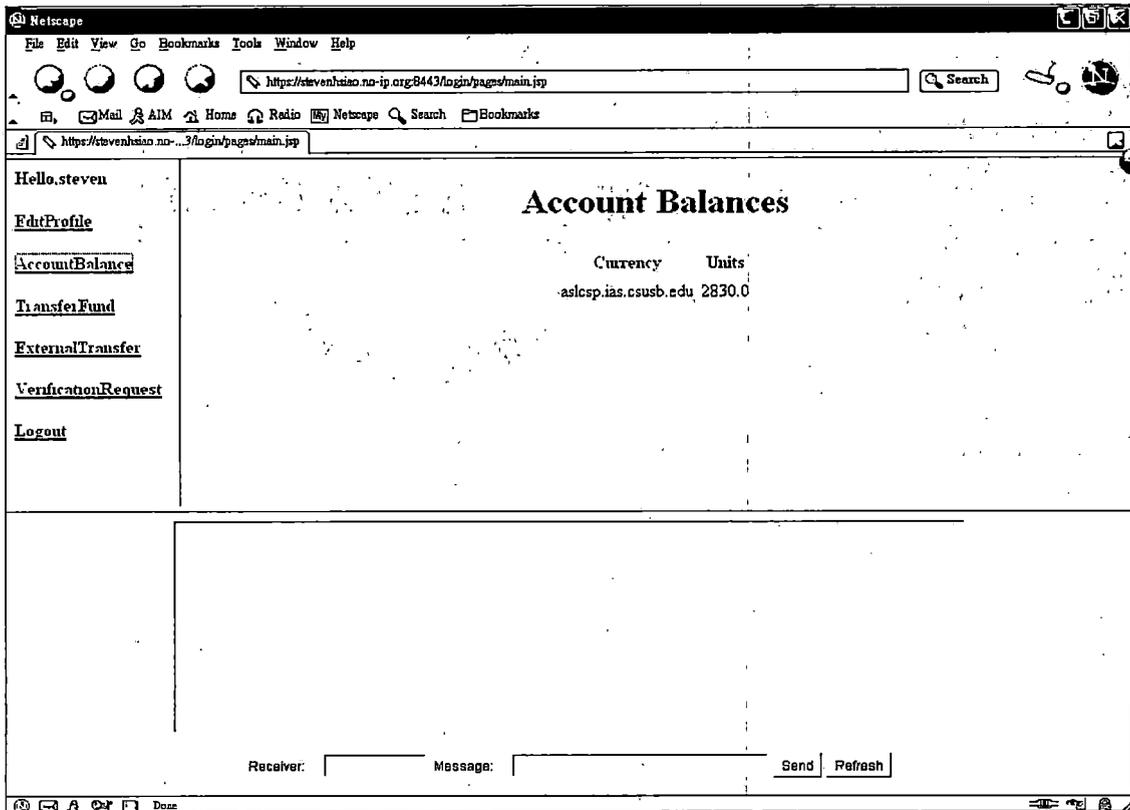


Figure 4.7. Account Balances Page

4.3.2.7 External Transfer Fund Page. The bank system offers an external transfer request function that allows a user to send funds to another user on different systems. For instance, a user in bank system 1 can transfer funds to a user in bank system 2. A user needs to fill out a form with necessary information to construct an external transfer request message. The system will convert the form to an XML document, send the XML document over SSL socket connection to a currency issuer, and then get an XML document response from the issuer. Afterward, the system

displays a result page to the user. The graphical interface is shown in Figure 4.8.

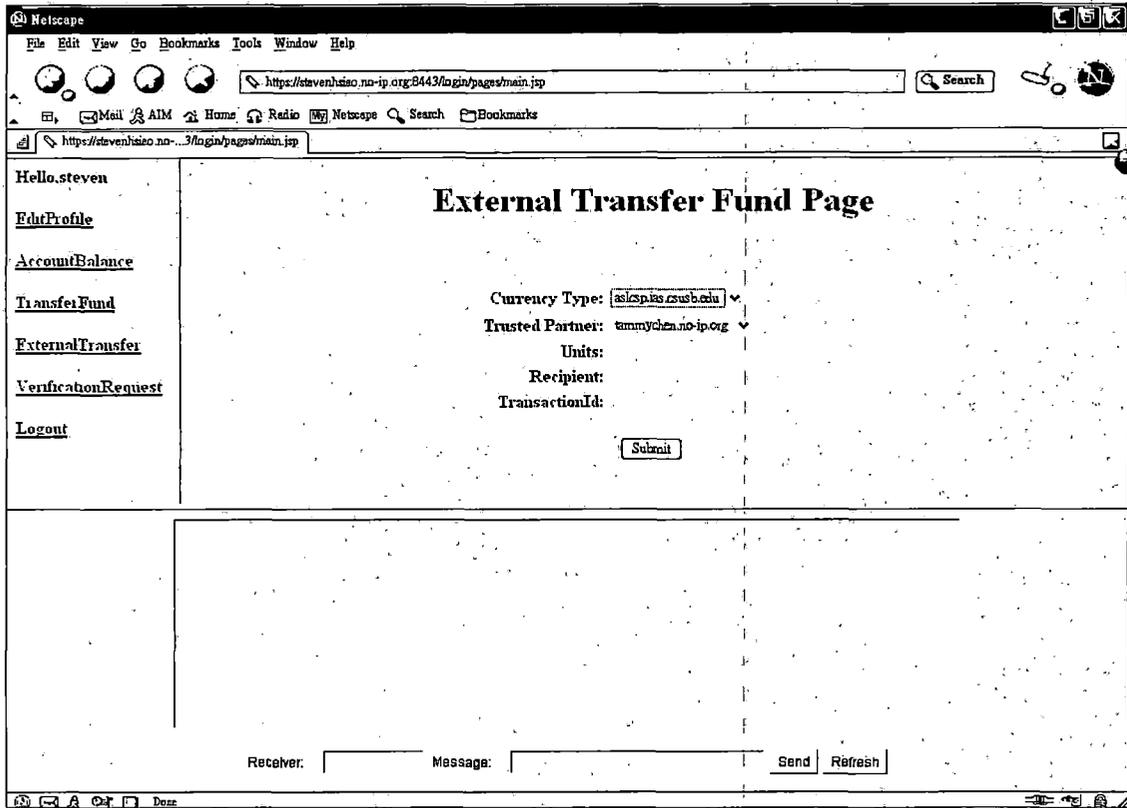


Figure 4.8. External Transfer Fund Page

4.3.2.8 Verification Request Page. A verification Request function is used to check if any deposit exists from a user outside the bank system. A user needs to specify currency type, and transactionId corresponding to a particular deposit in the currency issuer's database. The operation is similar to the external request function. In the end, the system displays an appropriate result page

to the user. The graphical interface is shown in Figure 4.9.

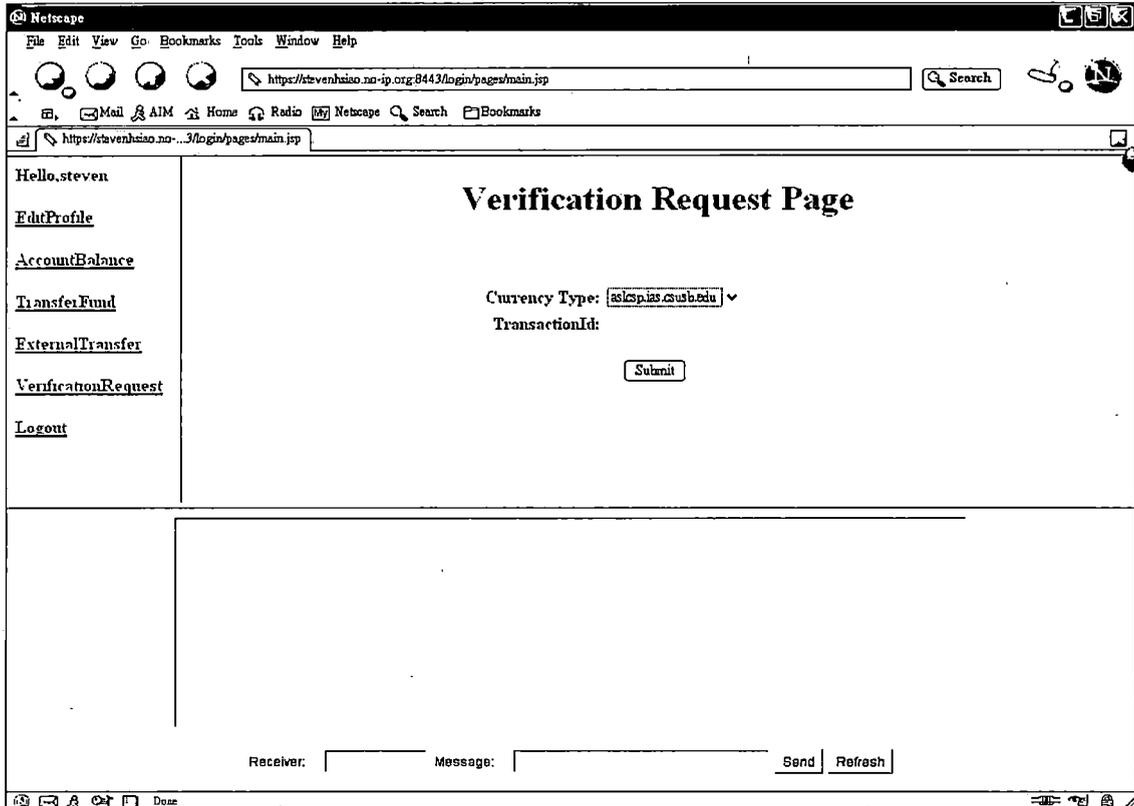


Figure 4.9. Verification Request Page

### 4.3.3 Database Design

We assume that a bank system must have at least one user and a trusted bank system. A user on the system either has no account or several accounts related to different currency types. Also, a user has an email address, and userid and password to get the privilege to login the system. Furthermore, the system detects the IP address of the user's computer and assigns a random number

to the user for the applet security issue. Figure 4.10 is the ER diagram showing the relations and the attributes of each entity.

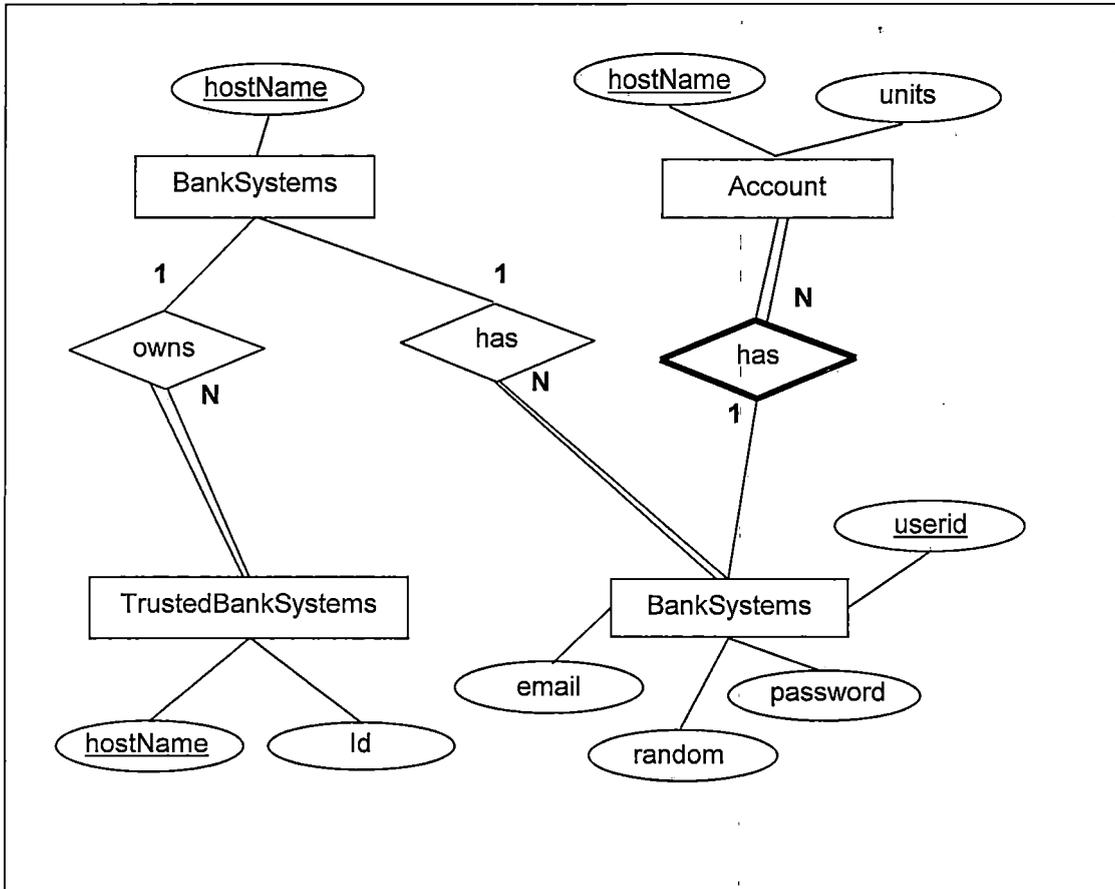


Figure 4.10. Bank System ER Diagram

#### 4.4 Currency Issuer Design

##### 4.4.1 Request and Response Message Format Design

Chapter one section 1.1 summarized the LCP protocol. This section shows the format of each message. LCP protocol uses XML documents to encapsulate messages. The reason is because XML is a cross platform, software and

hardware independent standard for transmitting information. Even if a program is written in different language like C++, C#, etc., the program can still communicate with the currency issuer because of the flexibility and compatibility of XML.

4.4.1.1 Transfer Request Message Format. An XML document with a root element of "transferRequest" is used to specify a request message. The root element contains four child elements:

- recipient: The content of this element is the recipient public key.
- amount: The content of this element is the amount of this transaction.
- transactionId: The content of this element is the unique transactionId related to this transaction.
- messageCounter: The content of this element is either 1 or 0. The purpose of messageCounter is described in Chapter one.

An example of a transferRequest message is shown in Figure 4.11.

```
<?xml version="1.0" encoding="utf-8"?>
<request>
  <transferRequest>
    <recipient>public key</recipient>
    <amount>10000</amount>
    <transactionId>alice;1234</transactionId>
    <messageCounter>0</messageCounter>
  </transferRequest>
</request>
```

Figure 4.11. Transfer Request Message Format

4.4.1.2 Verification Request Message Format. The root element is "verificationRequest." The element contains two child elements:

- transactionId: The content of this element is a unique transactionId to locate the corresponding deposit.
- messageCounter: Same as transferRequest message, the content is either 1 or 0.

Figure 4.12 shows an example of verification request message.

```
<?xml version="1.0" encoding="utf-8"?>
<request>
  <verificationRequest>
    <transactionId>alice;1234</transactionId>
    <messageCounter>0</messageCounter>
  </verificationRequest>
</request>
```

Figure 4.12. Verification Request Message Format

4.4.1.3 Transfer Response Message Format. After the execution of transfer fund request, a currency issuer generates a transfer response message. The response message root element is named "transferResponse". Inside "transferResponse", there are two child elements:

- result: The content shows if the transaction success or not.
- fee: The currency charges fees for each request.

An example of a transfer response message is shown in Figure 4.13.

```
<?xml version="1.0" encoding="utf-8"?>
<request>
  <transferRequest>
    <result>success</result>
    <fee>1.0</fee>
  </transferRequest>
</request>
```

Figure 4.13. Transfer Response Message Format

4.4.1.4 Verification Response Message Format. The root element of the transfer response message is called "verificationResponse." For a success case, the element "verificationResponse" contains two child elements:

- amount: The content is the amount of the lightweight currency received from another entity.

- fee: The currency issuer charges fees for requests.

Figure 4.14 shows a success example.

```
<?xml version="1.0" encoding="utf-8"?>
<request>
  <verificationRequest>
    <amount>1000</amount>
    <fee>2.0</fee>
  </verificationRequest>
</request>
```

Figure 4.14. Success Verification Response Message

Technically, not every verification message can locate a deposit record. For a failure case, the response message must describe the reason. We specify the reason in an attribute of the element "verificationResponse" named "reason." Also, a child element called fee is used to indicate the processing fee charged by the currency issuer. A failure example is shown in Figure 4.15.

```
<?xml version="1.0" encoding="utf-8"?>
<response>
  <verificationResponse
    reson="inexistent transactionId">
    <fee>2.0</fee>
  </verificationResponse>
</response>
```

Figure 4.15. Failure Verification Response Message

#### 4.4.2 Database Design

We suppose that a currency issuer must have at least one account belonging to a user. Also, the currency issuer holds deposit records for users and backups the response messages for users. The ER diagram illustrating the relations and attributes of each entity is shown in Figure 4.16.

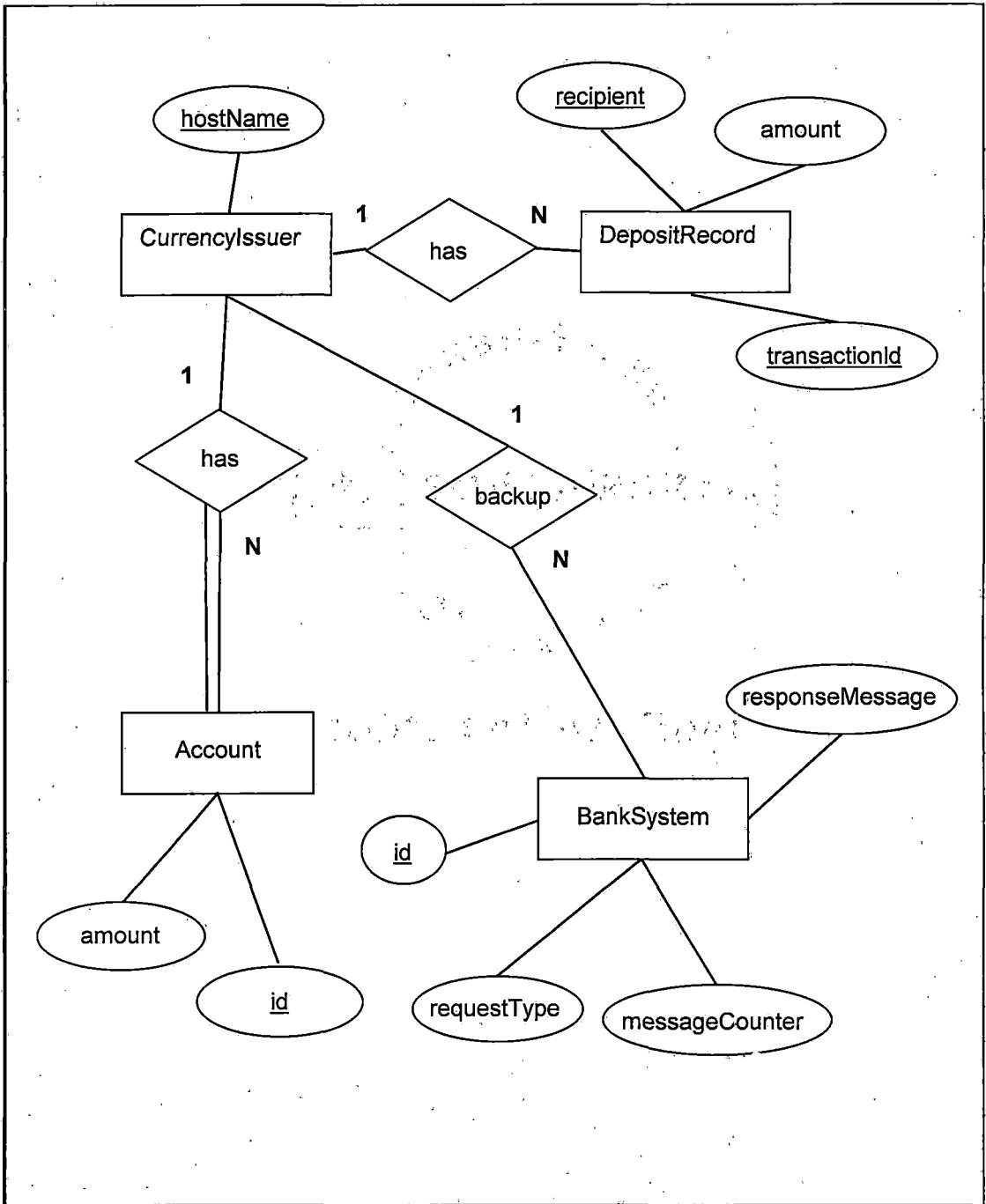


Figure 4.16. Currency Issuer ER Diagram

## CHAPTER FIVE

### PROJECT IMPLEMENTATION

#### 5.1 Project Class Design

Since the last chapter described the design of this project, the implementation is to be the next step. The Unified Modeling Language (UML) class diagram is a well known and commonly used diagram to analysis and design the relationships and functionalities of each class. As I mentioned before, the whole project can be separated into two main systems. One is the currency issuer and the other is the bank system. For the bank system, a client module that handles the communication with a currency issuer and can be integrated into other related applications will be described independently.

##### 5.1.1 Bank System Class Design

The summary of classes of the bank system and all classes are listed alphabetically:

- **Account:** the class is a bean object which is a JavaBean class used to interact with jsp.
- **AccountDAO:** An account database access object class used to access the database related to the client's account table.

- BaseDAO: A super class for all database access object classes like AccountDAO. The class offers the basic functions such as query, update, and insert operations.
- ChatServer: A Java Servlet that handles clients' SSL socket connections. Furthermore, the classes control the transmission of messages between user and user, and server and user.
- CheckAccountBalanceAction: A Struts Action class handling the action that once a user uses this function on the main page. After the action is completed, the class will forward the user to another page.
- DAOException: The database access object classes exception.
- EditProfileAction: It is a Struts Action class handling an action that a user modifies his profile.
- ExternalTransferAction: The action class handles the action that a user desires to perform an external transferRequest over LCP protocol.
- LoginAction: This action class authenticates a user. If success, the user will be forwarded to

the main page. Otherwise, the user will be returned to the login page.

- **LogoutAction:** This action class operates the preparation processes when a user tries to log out of the system.
- **LogoutHandler:** A Java Servlet between LogoutAction and ChatServer class. The class notifies the ChatServer to turn off the socket connection with the logout user and remove this user in the chat system.
- **RegistrationAction:** A Struts action class that deals with the registration operation. If the operation success, the class forwards the user to the main page.
- **RequestHandleServlet:** This handles all clients' externalTransfer and verificationRequest functions.
- **RequestType:** A JavaBean class that stores the external operation type. The type is either transferRequest or verificationRequest.
- **TransferFundAction:** The action class deals with the internal transfer fund operation.

- TransferFundHandler: A Java Servlet between TransferFundAction and ChatServer. It generates the dynamic response message for the user and also contacts the ChatServer to send the result to the recipient who received the funds.
- TransferRequestBean: A JavaBean object used for external transferRequest operation.
- TransferResultBean: The bean object stores the result of external transferRequest operation.
- TrustPartner: The class is a bean object saves the information of another bank system's information.
- TrustPartnerDAO: A database access object is used to access the database.
- User: A JavaBean object to store the current user's information.
- UserDAO: A database access object is used to access the database related to user table.
- UserService: A database access object manager handles all database access operations.
- VerificationRequestAction: This action class deals with the verificationRequest operation.

- VerificationRequestBean: A bean object stores the necessary parameters for verificationRequest operation.
- VerificationResultBean: A bean object stores the result of verificationResponse received from the currency issuer.

### 5.1.2 Lightweight Currency Client Module Class Design

The summary of classes of client module and all classes are listed alphabetically:

- ParseResult: The class is used to store the element name and content from a received XML document.
- RequestHandler: This class handles external requests to the currency issuer.
- ServerConnectionHandler: This class hides the preparation for SSL socket connection with a currency issuer. It also offers some setting methods for different applications with different setting.
- TransferResult: This class stores the results of transferRequest operation.
- VerificationResult: This class stores the result of verificationRequest operation.

- XMLHandler: Parse the coming XML document and pull out the element name and content.

The class diagram is shown in Figure 5.1.

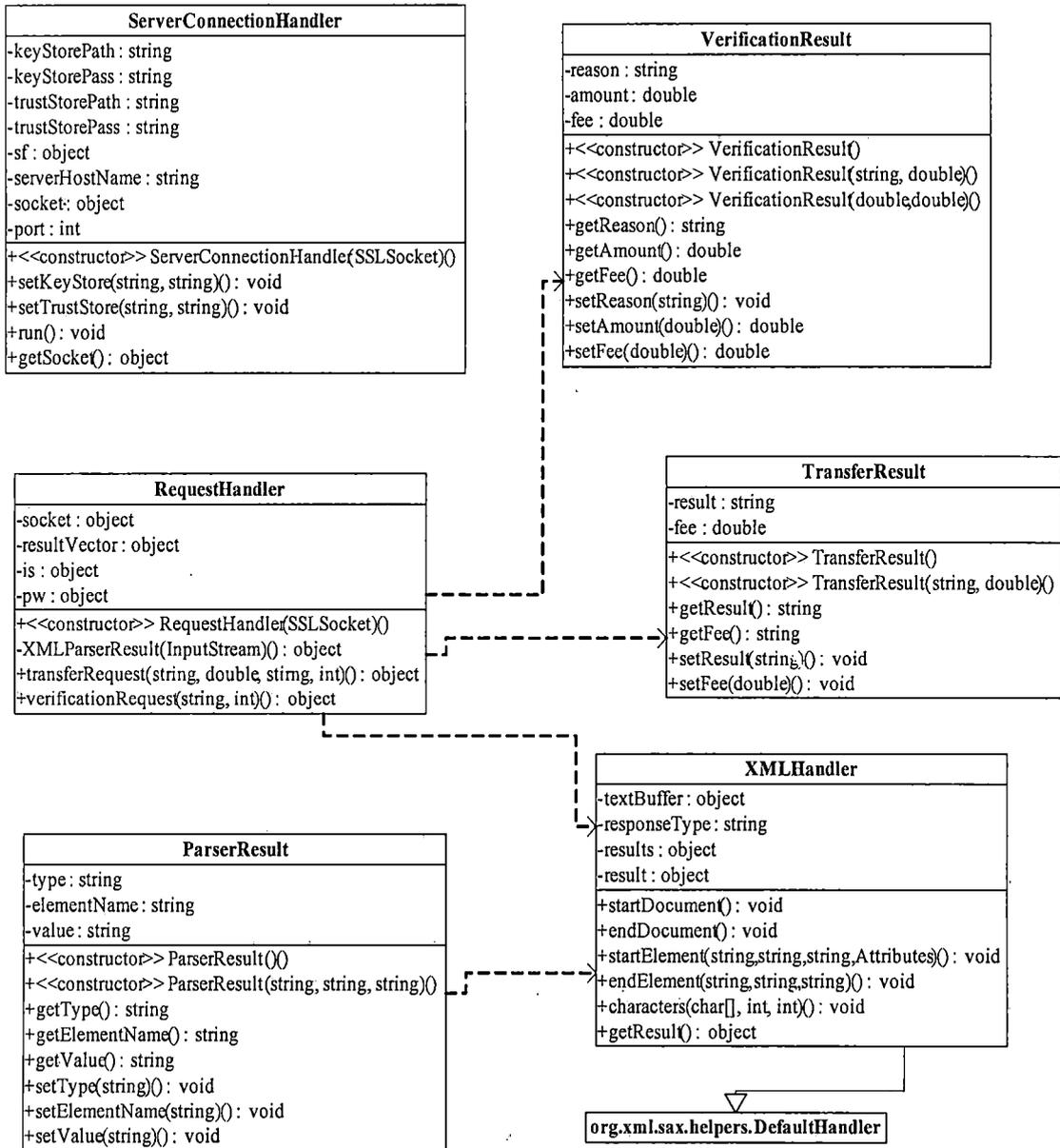


Figure 5.1. Client Module Class Diagram

### 5.1.3 The Currency Issuer Classes Design

The summary of classes of the currency issuer and all classes are listed alphabetically:

- Client: This class handles the request from the client.
- ClientConnectionHandler: This class prepares the SSL setting and handles the coming client connection. It also provides some setter methods for the issuer with different settings.
- ParserResult: The class stores the XML document parsing result for further analysis and operation.
- SSLSimpleServer: A main class runs a currency issuer server for waiting and operating on outside requests.
- XMLHandler: The class is used to parse the XML document and put the result in a vector for further usage.

The class diagram is shown in Figure 5.2.

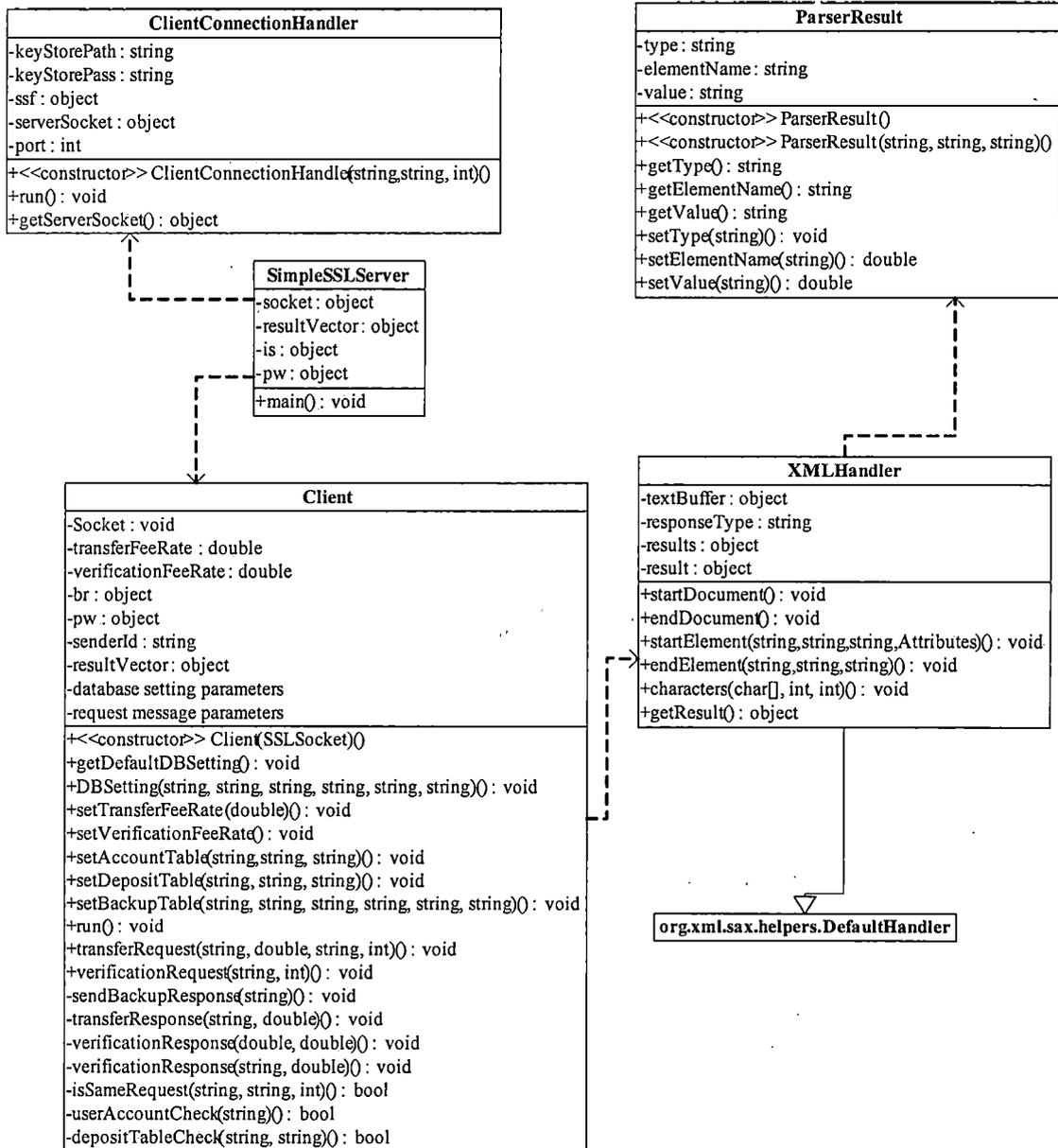


Figure 5.2. Currency Issuer Class Diagram

## 5.2 Project Classes Implementation

This section describes the logical algorithms used in this project and explanation for each method. To make the

classes easier to read, all classes will be written in plain English language.

### 5.2.1 Bank System

#### 5.2.1.1 The Account Class.

Class name: Account Class
Type: JavaBean Class
Constructor: Default Account
Methods:
getUserid: return value type String
getCurrency: return value type String
getUnits: return value type double
setUserid: set the class attribute userid
setCurrency: set up the class attribute currency
setUnits: set up the class attribute units

Figure 5.3. The Account Class

#### 5.2.1.2 The AccountDAO Class.

<p>Class name: AccountDAO Class</p> <p>Type: Database Access Object Class</p> <p>Constructor: Default AccountDAO</p> <p>Super Class: BaseDAO</p> <p>Exception: DAOException</p> <p>Methods:</p> <p>getAccountBalance: get the particular user account balances. Return a List of Account objects.</p> <p>readAccount: get a account record corresponded to a specified currency type. Return an Account object.</p> <p>createAccount: create a account in the database to the specified. Return an Account object.</p> <p>updateAccount: update the account record in the database to the specified currency type. Return value type integer.</p>
---

Figure 5.4. The AccountDAO Class

#### 5.2.1.3 The BaseDAO Class.

<p>Class name: BaseDAO Class</p> <p>Type: Database Access Object Super Class</p> <p>Constructor: Default BaseDAO</p> <p>Methods:</p> <p>getList: get a result list for a specified method name same as the sqlmap XML file. Return a List of objects.</p> <p>getObject: get a result for a specified method name same as the sqlmap XML file. Return an object.</p> <p>update: Perform update, create, and delete database operations.</p>
--

Figure 5.5. The BaseDAO Class

#### 5.2.1.4 The ChatServer Class.

Class name: ChatServer Class  
Type: Java Servlet Class  
Constructor: Default ChatServer  
Seper Class: HttpServlet  
Methods:  
init: Initialize the startup setting for the Servlet  
destroy: terminate the Server Ssocket.  
runServer: start running the server. Open a SSL server socket to listen clients' connection.  
doGet: Operate the request asked from other Servlets.  
Begin:  
    If the request comes from LogoutHandler Servlet, the Chat Server closes the socket with the particular client.  
    If the request comes from TransferFundHandler Servlet, the Chat Server send a notify message to the specific user.  
End doGet method  
Child Classes:  
ChatHandler  
MessageHandler  
Class name: ChatHandler Class  
Type: Java Thred Class  
Conctructor: Default ChatHandler  
Method of ChatHandler  
run: a run method is used to define threads. The purpose of the run method is create a MessageHandler class to handle the transmit message for each client by passing in the socket setting.  
Class name: MessageHandler Class  
Type: Java Thread Class  
Constructor: MessageHandler  
Parameter in SSLSocket  
Method of MessageHandler:  
run: a run method is used to define threads. The purpose of the run method is listening to the coming message to process the appropriate operations.  
Begin:  
    If request message comes from the client by passing a specific recipient id, the handler will send the message to the recipient.  
    If request message comes from the client without a recipient id, the handler will broadcast the message to all users on the system.  
End run method  
broadcast: method for sending the message to all users on the system now  
close: close the socket connection for this user

Figure 5.6. The ChatServer Class

#### 5.2.1.5 The CheckAccountBalanceAction Class.

Class name: CheckAccountBalanceAction Class
Type: Struts Action Class
Constructor: Default CheckAccountBalanceAction
Methods: execute: check the database and forward the user to the destination page.

Figure 5.7. The CheckAccountBalanceAction Class

#### 5.2.1.6 The DAOException Class.

Class name: DAOException Class
Type: Exception Class
Super Class: Exception
Constructor: Default CheckAccountBalanceAction
Constructor: CheckAccountBalanceAction Parameter in: cause
Constructor: CheckAccountBalanceAction Parameter in: message

Figure 5.8. The DAOException Class

#### 5.2.1.7 The EditProfileAction Class.

Class name: EditProfileAction Class
Type: Struts Action Class
Constructor: Default EditProfileAction Class
Methods: Execute: update the user profile record in the database and forward the user to the destination page,

Figure 5.9. The EditProfileAction Class

#### 5.2.1.8 The ExternalTransferAction Class.

<p>Class name: ExternalTransferAction Class</p> <p>Type: Struts Action Class</p> <p>Constructor: Default ExternalTransferAction</p> <p>Methods:</p> <p>Execute: Call the LCP client module to process the transferRequest operation and update the user account record in the database and forward the user to the destination page.</p>
--

Figure 5.10. The ExternalTransferAction Class

#### 5.2.1.9 The LoginAction Class.

<p>Class name: LoginAction Class Class</p> <p>Type: Struts Action Class</p> <p>Constructor: Default LoginAction Class</p> <p>Methods:</p> <p>Execute: Authenticate the user status, save some information in Http Session, and forward the user to the destination page.</p>
--

Figure 5.11. The LoginAction Class

#### 5.2.1.10 The LogoutAction Class.

<p>Class name: LogoutAction Class</p> <p>Type: Struts Action Class</p> <p>Constructor: Default LogoutAction</p> <p>Methods:</p> <p>Execute: Prepare the logout process and forward the request to LogoutHandler class.</p>
--

Figure 5.12. The LogoutAction Class

#### 5.2.1.11 The LogoutHandler Class.

<p>Class name: LogoutHandler Class</p> <p>Type: Java Servlet Class</p> <p>Constructor: Default LogoutHandler</p> <p>Methods:</p> <p>DoGet: Prepare the logout operation such as clear the session and ask the Chat Server to close the socket connection with the user.</p>
---

Figure 5.13. The LogoutHandler Class

#### 5.2.1.12 The RegistrationAction Class.

<p>Class name: RegistrationAction Class</p> <p>Type: Struts Action Class</p> <p>Constructor: Default RegistrationAction</p> <p>Methods:</p> <p>Execute: check if the user exists in the database. If not, create an account for the user. Forward the user to the destination page.</p>
---

Figure 5.14. The RegistrationAction Class

### 5.2.1.13 The RequestHandleServlet Class.

Class name: RequestHandleServlet Class  
Type: Java Servlet Class  
Constructor: Default RequestHandleServlet  
Super Class: HttpServlet  
Methods:  
init: Initialize the startup setting for the Servlet.  
DoGet: Operate the request asked from other Servlets.  
Begin:  
    Create a Request object for the user.  
    If the request type is transferRequest, pass the parameters of transferRequest message to the Request object.  
    If the request type is verificationRequest, pass the parameters of verificationRequest message object.  
End doGet method  
GetTransferResults: return a TransferResult object.  
GetVerificationResult: return a VerificationResult object.  
Child Class: Request Class  
Constructor: Request  
Method:  
Run: Process the request and save the result.  
Begin:  
    If the request type transferRequest, call the LCP client module to ask the currency issuer to process the request, get the result, and save the result in a TransferResult Object.  
    If the request type is verificationRequest, call the LCP client module to ask the currency issuer to process the request, get the result, and save the result in a VerificationResult Object.  
End run method

Figure 5.15. The RequestHandleServlet Class

#### 5.2.1.14 The RequestType Class.

<p>Class name: RequestType Class Type: JavaBean Class Constructor: Default RequestType Methods: getType: return value type String setType: set up the type.</p>
---

Figure 5.16. The RequestType Class

#### 5.2.1.15 The TransferFundAction Class.

<p>Class name: TransferFundAction Class Type: Strud Action Class Constructor: Default TransferFundAction Methods: execute: execute the internal transfer fund operation and send the notification message. Begin:     If the transfer is success, update both the user and recipient's records, and forward to the TransferFundHandler Servlet for generating output interface.     If the transfer is fail, forward to the TransferFund Handler Servlet for generating output interface. End execute method</p>
--

Figure 5.17. The TransferFundAction Class

#### 5.2.1.16 The TransferFundHandler Class.

<p>Class name: TransferFundHandler Class</p> <p>Type: Java Servlet</p> <p>Constructor: Default TransferFundHandler</p> <p>Methods:</p> <p>doGet: generate the dynamic output interface and ask ChatServer to send a notification message to the recipient.</p> <p>Begin:</p> <ul style="list-style-type: none"><li>    If the transfer is success, generate the success page and call ChatServer to send a success message to the recipient who got the money.</li><li>    If the transfer is fail, generate the failure page with appropriate messages and call ChatServer to send a failure messages with reason to the recipient.</li></ul> <p>End doGet method</p>
--

Figure 5.18. The TransferFundHandler Class

### 5.2.1.17 The TransferRequestBean Class.

<p>Class name: TransferRequestBean Class</p> <p>Type: JaveBean Class</p> <p>Constructor: TransferRequestBean</p> <p>Parameter in:</p> <p>recipient: value type String</p> <p>issuer: value type String</p> <p>amount: value type double</p> <p>transactionId: value type String</p> <p>Methods:</p> <p>getRecipient: return value type String</p> <p>getIssuer: return value type String</p> <p>getAmount: return value type String</p> <p>getTransactionId: return value type String</p> <p>setRecipient: set the recipient name</p> <p>setIssuer: set the currency issuer</p> <p>setAmount: set the amount</p> <p>setTransactionId: set up thetransactionId</p>
---

Figure 5.19. The TransferRequestBean Class

#### 5.2.1.18 The TransferResultBean Class.

Class name: TransferResultBean Class  
Type: JaveBean Class  
Constructor: Default TransferResultBean  
Constructor: TrustPartner  
Parameter in:  
Result: value type String  
Fee: value type double  
Methods:  
getResult: return value type String  
getFee: return value type double  
setResult: set the result.  
setFee: set the fee charged form the currency issuer.

Figure 5.20. The TransferResultBean Class

#### 5.2.1.19 The TrustPartner Class.

Class name: TrustPartner Class  
Type: JaveBean Class  
Constructor: Default TrustPartner  
Methods:  
getId: return value type String  
getName: return value type double  
setId: set the public key identifier of the specific currency.  
setName: set the currency issuer name which is equals to the hostname of the issuer.

Figure 5.21. The TrustPartner Class

#### 5.2.1.20 The TrustPartnerDAO Class.

<p>Class name: TrustPartnerDAO Class</p> <p>Type: Database Access Object Class</p> <p>Constructor: Default TrustPartnerDAO</p> <p>Super Class: BaseDAO</p> <p>Exception: DAOException</p> <p>Methods:</p> <p>getTrustPartner: get a list of trusted bank system data from database.</p> <p>setTrustPartnerByName: set a particular trusted bank system data by using the key name.</p>
--

Figure 5.22. The TrustPartnerDAO Class

#### 5.2.1.21 The User Class.

<p>Class name: User Class</p> <p>Type: JaveBean Class</p> <p>Constructor: Default User</p> <p>Constructor: User</p> <p>Parameter in:</p> <p>userid: value type String</p> <p>Methods: return value type String</p> <p>getUserId: return value type String</p> <p>getPassword: return value type String</p> <p>gePassword2: return value type String</p> <p>getEmail: return value type String</p> <p>getIp: return value type String</p> <p>getRandom: return value type String</p> <p>setUserId: set the userid of this user</p> <p>setPassword:set the password</p> <p>setPassword2: set the confirmation password</p> <p>setEmail: set the e-mail address</p> <p>setRandom: set a random to locate the user</p>
--

Figure 5.23. The User Class

#### 5.2.1.22 The UserDao Class.

<p>Class name: UserDao Class</p> <p>Type: Database Access Object Class</p> <p>Constructor: Default UserDao</p> <p>Super Class: BaseDAO</p> <p>Exception: DAOException</p> <p>Methods:</p> <p>checkUser: get user data by pass a User object.</p> <p>getUserByIp: get a particular user by using a key parameter ip.</p> <p>GetUserByRandom: get a particular user by using a key parameter random.</p> <p>insertUser: insert the user data into the database.</p> <p>updateUser: update the specific user data in table user.</p> <p>updateUserProfile: update the specific userProfile data in table profile.</p>
--

Figure 5.24. The UserDao Class

#### 5.2.1.23 The UserService Class.

<p>Class name: UserService Class</p> <p>Type: Database Access Object Super Class</p> <p>Constructor: Default UserService</p> <p>Methods:</p> <p>checkUser: get user data by pass a User object.</p> <p>getUserByIp: get a particular user by using a key parameter ip.</p> <p>getUserByRandom: get a particular user by using a key parameter random.</p> <p>insertUser: insert the user data into the database.</p> <p>updateUser: update the specific user data in table user.</p> <p>updateaUserProfile: update the specific userProfile data in table profile.</p> <p>getAccountBalance: get a list of the user's account information.</p> <p>readAccount: get a particular account data by specifying a currency type.</p> <p>createAccount: create a new account for the user.</p> <p>updateAccount: update the account record in the database.</p> <p>getTrustPartner: get a list of trusted bank system information.</p> <p>getTrustPartnerByName: get a particular trusted bank system by using a key name.</p>
--

Figure 5.25. The UserService Class

#### 5.2.1.24 The VerificationRequestAction Class.

<p>Class name: VerificationRequestAction Class</p> <p>Type: Struts Action Class</p> <p>Constructor: Default VerificationAction</p> <p>Methods:</p> <p>execute: Call the RequestHandleServlet to process the verificationRequest operation and forward to the destination page.</p>
--

Figure 5.26. The VerificationRequestAction Class

#### 5.2.1.25 The VerificationRequestBean Class.

<p>Class name: VerificationRequestBean Class</p> <p>Type: JaveBean Class</p> <p>Constructor: Default VerificationRequestBean</p> <p>Constructor: VerificationRequestBean</p> <p>Parameter in:</p> <p>issuer: value type String</p> <p>transactionId: value type String</p> <p>Methods:</p> <p>getIssuer: return value type String</p> <p>getTransactionId: return value type String</p> <p>setIssuer: set the issue</p> <p>setTransactionId: set up the transactionId</p>
---

Figure 5.27. The VerificationRequestBean Class

#### 5.2.1.26 The VerificationResultBean Class.

<p>Class name: VerificationResultBean Class</p> <p>Type: JaveBean Class</p> <p>Constructor: Default VerificationResultBean</p> <p>Constructor: VerificationResultBean</p> <p>Parameter in:</p> <p>reason: value type String</p> <p>amount: value type double</p> <p>fee: value type double</p> <p>Method:</p> <p>getReason: return value type String</p> <p>getAmount: return value type String</p> <p>getFee: return value type double</p> <p>setReason: set the failure reason</p> <p>setAmount: set the amount the user got from a user in another bank system.</p> <p>setFee: set the fee charged from the currency.</p>
--

Figure 5.28. The VerificationResultBean Class

## 5.2.2 Currency Client Module Classes Implementation

### 5.2.2.1 The ParserResult Class.

<p>Class name: ParserResult Class</p> <p>Constructor: Default ParserResult</p> <p>Constructor: ParserResult</p> <p>Parameter in:</p> <p>type: value type String</p> <p>elementName: value type String</p> <p>value: value type String</p> <p>Method:</p> <p>getType: value type String</p> <p>getElementName: value type String</p> <p>getValue: value type String</p> <p>setType: set the response type.</p> <p>setElementName: set the element name.</p> <p>setValue: set the value.</p>
--

Figure 5.29. The ParserResult Class

### 5.2.2.2 The RequestHandler Class.

<p>Class name: RequestHandler Class</p> <p>Constructor: Default RequestHandler</p> <p>Parameter in:</p> <p>Socket: SSLSocket</p> <p>Methods:</p> <p>XMLParserResult: return a vector stored the results.</p> <p>TransferRequest: operate a transferRequest process and return a TransferResult object.</p> <p>VerificationRequest: operate a verificationRequest process and return a VerificationResult object.</p>
--

Figure 5.30. The RequestHandler Class

### 5.2.2.3 The ServerConnectionHandler Class.

<p>Class name: ServerConnectionHandler Class</p> <p>Constructor: ServerConnectionHandler</p> <p>Parameter in:</p> <p>severHostName: value type String</p> <p>port: value type integer</p> <p>Methods:</p> <p>setKeyStore: pass the keyStore path and keyStore password to set the keyStore.</p> <p>getKeyStore: pass the trustStore path and trustStore password to set the trustStore.</p> <p>run: process the SSL socket connection.</p> <p>getSocket: return a socket reference.</p>
---

Figure 5.31. The ServerConnectionHandler Class

### 5.2.2.4 The TrasferResult Class.

<p>Class name: TransferResult Class</p> <p>Constructor: Default TransferResult</p> <p>Constructor: TransferResult</p> <p>Parameter in:</p> <p>result: value type String</p> <p>fee: value type String</p> <p>Methods:</p> <p>getResult: return value type String</p> <p>getFee: return value type double</p> <p>setResult: set the transferResponse result.</p> <p>setFee: set the fees charged from the currency issuer.</p>
---

Figure 5.32. The TransferResult Class

#### 5.2.2.5 The VerificationResult Class.

**Class name:** VerificationResult Class

**Constructor:** Default VerificationResult

**Constructor:** VerificationResult

**Parameter in:**

reason: value type String

fee: value type double

**Constructor:** VerificationResult

**Parameter in:**

amount: value type double

fee: value type double

**Methods:**

getReason: return value type String

getAmount: return value type double

getFee: return value type double

setReason: set the failure reason

setAmount: set the amount get from a user on another bank system.

setFee: set the fees charged from the currency issuer.

Figure 5.33. The VerificationResult Class

#### 5.2.2.6 The XMLHandler Class.

**Class name:** XMLHandler Class

**Super class:** org.xml.sax.helpers.DefaultHandler

**Constructor:** Default XMLHandler

**Methods:**

startDocument: detect the start of the XML document.

endDocument: detect the end of the XML document.

startElement: detect the start of the element. Extract the element name, and attribute name and value of each attribute.

endElement: extract the content of the element. Detect the end of the element.

characters: the value of content.

getResult: get a vector stores the parsing result.

Figure 5.34. The XMLHandler Class

### 5.2.3 The Currency Issuer Classes Implementation

#### 5.2.3.1 The Client Class.

Class name: Client Class  
Constructor: Client  
Parameter in:  
socket: Socket  
Methods:  
setDefaultDBSetting: set up the default database setting.  
DBSetting: pass in the necessary parameter to adjust the database setting to the system.  
setTransferFeeRate: set the transferRequest fee rate.  
setVerificationFeeRate: set the verificationRequest fee rate.  
setAccountTable: set the table name attributes' names for the table account.  
setDepositTable: set the table name attributes' names for the table deposit.  
setBackupTable: set the table name attributes' name for the table backup.  
userAccountCheck: check if the client owns any account in the database.  
run: connect to the database, parse the XML document and process the request after parsing.  
Begin:  
    If userAccountCheck is not true, close the SSL socket connection with the client because the client doesn't own any account in the currency issuer.  
    If the request type is transferRequest, call the method transferRequest to execute the request.  
    If the request type is verificationRequest, call the method verificationRequest to execute the request.  
End ren method  
isSameRequest: check if the coming request message same as the last request sent from the client.  
transferRequest: process the transferRequest  
Begin:  
    If isSameRequest is true, call the sendBackupResponse to send the backup response which stored in the database back to the client.  
    If depositTableCheck is true, call the method transferResponse to perform and send a failure response message.  
    If not, update the database, call the method transferResponse to perform and send the response message.  
End transferRequest method  
verificationRequest: process the verificationRequest  
Begin:  
    If isSameRequest is true, call the sendBackupResponse to send the backup response which stored in the database back to the client.  
    If depositTableCheck is true, call the method verificationResponse to perform and send a success response message.  
    If not, call the method verificationResponse to perform and send a failure response message.  
End verificationRequest method  
transferResponse: perform a XML document with the response message and send the message back to the client.  
verificationResponse: perform a XML document with the response message and send the message baxk to the client.

Figure 5.35. The Client Class

### 5.2.3.2 The ClientConnectionHandler Class.

<p>Class name: ClientConnectionHandler Class</p> <p>Constructor: ClientConnectionHandler</p> <p>Parameter in:</p> <p>keystorePath: value type String</p> <p>keystorePass: value type double</p> <p>port: value type integer</p> <p>Methods:</p> <p>run: prepare the connection setting and create a SSLServerSocket.</p> <p>getServerSocket: return a ServerSocket.</p>
---

Figure 5.36. The ClientConnectionHandler Class

### 5.2.3.3 The ParserResult Class.

<p>Class name: ParserResult Class</p> <p>Constructor: Default ParserResult</p> <p>Constructor: ParserResult</p> <p>Parameter in:</p> <p>type: value type String</p> <p>elementName: value type String</p> <p>value: value type String</p> <p>Methods:</p> <p>getType: return value type String</p> <p>getElementName: return value type String</p> <p>getValue: return value type String</p> <p>setType: set the response type</p> <p>setElementName: set the element name.</p> <p>setValue: set the value.</p>
---

Figure 5.37. The ParserResult Class

#### 5.2.3.4 The SSLSimpleServer Class.

<p>Class name: SSLSimpleServer Class</p> <p>Constructor: Default SSLSimpleServer</p> <p>Methods:</p> <p>main: set up the database and SSL setting, run the server, and wait for client connecting.</p>
--

Figure 5.38. The SSLSimpleServer Class

#### 5.2.2.5 The XMLHandler Class.

<p>Class name: XMLHandler Class</p> <p>Super class: org.xml.sax.helpers.DefaultHandler</p> <p>Constructor: Default XMLHandler</p> <p>Methods:</p> <p>startDocument: detect the start of the XML document.</p> <p>endDocument: detect the end of the XML document.</p> <p>startElement: detect the start of the element. Extract the element name, and attribute name value of each attribute.</p> <p>endElement: extract the content of the element. Detect the end of the element.</p> <p>characters: the value of content.</p> <p>getResult: get a vector stores the parsing result.</p>
--

Figure 5.39. The XMLHandler Class

## CHAPTER SIX

### FURTHER ENHANCEMENT AND CONCLUSION

#### 6.1 Related Projects

This project is just the beginning of LCP research. Nowadays, there are many projects working on other problems that integrate the LCP protocol with some application. These projects are listed as follows:

- A P2P content distribution system that relies on LCP protocol is being built by two graduate students in Polytechnic University, NY.
- A project title named "Economics with LCP" is being worked on by Yingzhuo Wang in CSU San Bernardino.
- A project title named "Second Generation Email" is being worked on by Ni Dang in CSU San Bernardino.
- A project title named "Currency Exchange Market" is being worked on by Yiyao Huo in CSU San Bernardion.

Because of the high potential of LCP, there can be more projects relying on the protocol.

## 6.2 Further Enhancement

The protocol is not yet perfect. The creators are trying to improve the protocol's ability to handle any kind of situation, therefore, LCP protocol probably evolves some extensions. The beginning version I used is not working very well with the bank system. For example, a user without understanding the protocol will be confused by the external functions. Possible enhancements are described as follows:

- According to the possible changes of LCP protocol, the later version may use XML document over SOAP message. There are two solutions to protect the SOAP messages. One solution is transferring the SOAP message over HTTPS and the other solution is encrypting the SOAP message first and transmitting the message over HTTP. The later version of bank system should handle the changes of LCP.
- Build more friendly graphical interfaces and improve the performance of the system is another goal for a further project.

### 6.3 Conclusion

Bandwidth, storage, and computation are most important resources on the P2P resource market. However, the market does not have any good solution to trade these surplus resources until the birth of LCP. The LCP protocol provides a strict standard to build a robust trading mechanism on the P2P resource market. This protocol provides at least three main advantages: high security, high flexibility, and ease of use. Besides, the protocol is not limited in the P2P resource market. It is possible for it to be used in commercial areas. I believe the LCP protocol will become one of famous standards in the future.

APPENDIX A

XML FILE

The web.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Login Practice</display-name>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>application</param-name>
      <param-value>properties/ApplicationResources</param-value>
    </init-param>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>3</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>3</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>chat</servlet-name>
    <servlet-class>myapp.ChatServer</servlet-class>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>test</servlet-name>
    <servlet-class>myapp.MessageCenter</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>logoutHandler</servlet-name>
    <servlet-class>myapp.LogoutHandler</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>request</servlet-name>
    <servlet-class>myapp.RequestHandleServlet</servlet-class>
    <load-on-startup>3</load-on-startup>
  </servlet>
  <!-- Action Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>/do/*</url-pattern>
  </servlet-mapping>
```

```

<servlet-mapping>
  <servlet-name>chat</servlet-name>
  <url-pattern>/chat</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>request</servlet-name>
  <url-pattern>/request</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>test</servlet-name>
  <url-pattern>/test</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>logoutHandler</servlet-name>
  <url-pattern>/logoutHandler</url-pattern>
</servlet-mapping>
<!-- The Welcome File List -->
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<!-- tag libs -->
<taglib>
  <taglib-uri>struts/bean-el</taglib-uri>
  <taglib-location>/WEB-INF/lib/struts-bean-el.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>struts/bean</taglib-uri>
  <taglib-location>/WEB-INF/lib/struts-bean.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>struts/html-el</taglib-uri>
  <taglib-location>/WEB-INF/lib/struts-html-el.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>struts/html</taglib-uri>
  <taglib-location>/WEB-INF/lib/struts-html.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>struts/logic-el</taglib-uri>
  <taglib-location>/WEB-INF/lib/struts-logic-el.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>struts/logic</taglib-uri>
  <taglib-location>/WEB-INF/lib/struts-logic.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>jstl/c</taglib-uri>
  <taglib-location>/WEB-INF/lib/c.tld</taglib-location>
</taglib>

<!-- resource def -->
<resource-ref>

```

```

    <res-ref-name>database</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</web-app>

```

## The struts-config.xml:

```

<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
  "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd" >
<struts-config>
  <!-- Form Bean Definitions -->
  <form-beans>
    <!-- DyanActionForm userForm used for Login -->
    <form-bean name="userForm"
      type="org.apache.struts.validator.DynaValidatorForm">
      <form-property name="userid"
        type="java.lang.String"/>
      <form-property name="password"
        type="java.lang.String"/>
    </form-bean>
    <!-- RegistrationForm used for registration and edit profile -->
    <form-bean name="registrationForm"
      type="org.apache.struts.validator.DynaValidatorForm">
      <form-property name="userid"
        type="java.lang.String"/>
      <form-property name="password"
        type="java.lang.String"/>
      <form-property name="password2"
        type="java.lang.String"/>
      <form-property name="email"
        type="java.lang.String"/>
    </form-bean>
    <form-bean name="transactionForm"
      type="org.apache.struts.validator.DynaValidatorForm">
      <form-property name="currency"
        type="java.lang.String"/>
      <form-property name="units"
        type="java.lang.String"/>
      <form-property name="userid"
        type="java.lang.String"/>
    </form-bean>
    <form-bean name="exteriorTransactionForm"
      type="org.apache.struts.validator.DynaValidatorForm">
      <form-property name="currency"
        type="java.lang.String"/>
      <form-property name="name"
        type="java.lang.String"/>
      <form-property name="units"
        type="java.lang.String"/>
      <form-property name="userid"
        type="java.lang.String"/>

```

```

    <form-property name="transactionId"
        type="java.lang.String"/>
</form-bean>
<form-bean name="verificationRequestForm"
    type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="currency"
        type="java.lang.String"/>
    <form-property name="transactionId"
        type="java.lang.String"/>
</form-bean>
</form-beans>
<!-- Global forwards -->
<global-forwards>
    <forward name="error"
        path="/pages/error.jsp"/>
    <forward name="login"
        path="/pages/login.jsp"/>
    <forward name="information"
        path="/pages/information.jsp"/>
    <forward name="editProfile"
        path="/pages/profile.jsp"/>
    <forward name="welcome"
        path="/pages/login.jsp"/>
    <forward name="main"
        path="/pages/main.jsp"/>
    <forward name="registration"
        path="/pages/registration.jsp"/>
    <forward name="accountBalance"
        path="/do/checkAccountAction"/>
    <forward name="transferFund"
        path="/pages/transferfund.jsp"/>
    <forward name="exteriorTransfer"
        path="/pages/exteriorTransfer.jsp"/>
    <forward name="verificationRequest"
        path="/pages/verificationRequest.jsp"/>
    <forward name="logout"
        path="/do/logoutAction"/>
    <forward name="success"
        path="/pages/login.jsp"/>
</global-forwards>
<!-- Action Mapping Definitions -->
<action-mappings>
    <!-- LoginAction definition
        Success: forward to profile page.
        Error: forward to error page. -->
    <action path="/loginAction"
        type="myapp.LoginAction"
        name="userForm"
        scope="request"
        validate="true"
        input="/pages/login.jsp"
    >
    <exception

```

```

        key="exception.database.error"
        type="myapp.DAOException"
        path="/pages/error.jsp"/>
    <forward
        name="success"
        path="/pages/main.jsp"
        redirect="true"/>
</action>

<!-- LogoutAction definition -->
<action path="/logoutAction"
        type="myapp.LogoutAction">
    <forward name="success"
        path="/logoutHandler"
        redirect="true"/>
</action>

<!-- RegistrationAction definition -->
<action path="/registrationAction"
        type="myapp.RegistrationAction"
        name="registrationForm"
        scope="request"
        validate="true"
        input="/pages/registration.jsp">
    <exception key="exception.database.error"
        type="myapp.DAOException"
        path="/pages/error.jsp"/>
    <forward name="success"
        path="/pages/main.jsp"
        redirect="true"/>
</action>

<!-- Manage Profile Action (Update User Data) definition -->
<action path="/editProfileAction"
        type="myapp.EditProfileAction"
        name="registrationForm"
        scope="request"
        validate="true"
        input="/pages/profile.jsp">
    <exception key="exception.database.error"
        type="myapp.DAOException"
        path="/pages/error.jsp"/>
    <forward name="success"
        path="/pages/transferfund.jsp"
        redirect="true"/>
</action>

<action path="/checkAccountAction"
        type="myapp.CheckAccountBalanceAction"
        scope="request"
        validate="false"
        input="/pages/accountBalance.jsp" >
    <exception key="exception.database.error"
        type="myapp.DAOException"
        path="/pages/error.jsp"/>
    <forward name="success"

```

```

        path="/pages/accountBalance.jsp"/>
    </action>
    <action path="/transferFundAction"
        type="myapp.TransferFundAction"
        name="transactionForm"
        scope="request"
        validate="false"
        input="/pages/transferfund.jsp" >
        <exception key="exception.database.error"
            type="myapp.DAOException"
            path="/pages/transferfund.jsp"/>
        <forward name="success"
            path="/test"
            redirect="true"/>
    </action>
    <action path="/exteriorTransferAction"
        type="myapp.ExteriorTransferAction"
        name="exteriorTransactionForm"
        scope="request"
        validate="false"
        input="/pages/exteriorTransfer.jsp" >
        <exception key="exception.database.error"
            type="myapp.DAOException"
            path="/pages/exteriorTransfer.jsp"/>
        <forward name="success"
            path="/request"
            redirect="true"/>
    </action>
    <action path="/verificationRequestAction"
        type="myapp.VerificationRequestAction"
        name="verificationRequestForm"
        scope="request"
        validate="false"
        input="/pages/verificationRequest.jsp" >
        <exception key="exception.database.error"
            type="myapp.DAOException"
            path="/pages/verificationRequest.jsp"/>
        <forward name="success"
            path="/request"
            redirect="true"/>
    </action>
</action-mappings>
<!-- message resources -->
<message-resources
    parameter="properties/ApplicationResources"
    null="false" />

<!-- plugins -->
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,
        /WEB-INF/validation.xml"/>
</plug-in>
</struts-config>

```

## The validation.xml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.0//EN"
    "http://jakarta.apache.org/commons/dtds/validator_1_0.dtd">
<form-validation>
  <!-- ===== Default Language Form Definitions ===== -->
  <formset>
    <form name="userForm">
      <field property="userid"
        depends="required,minlength,maxlength">
        <arg0 key="prompt.userid" />
        <arg1 key="{var:minlength}"
          name="minlength"
          resource="false" />
        <arg2 key="{var:maxlength}"
          name="maxlength"
          resource="false" />
        <var>
          <var-name>maxlength</var-name>
          <var-value>16</var-value>
        </var>
        <var>
          <var-name>minlength</var-name>
          <var-value>3</var-value>
        </var>
      </field>
      <field property="password"
        depends="required,minlength,maxlength"
        bundle="alternate">
        <arg0 key="prompt.password"/>
        <arg1 key="{var:minlength}"
          name="minlength"
          resource="false"/>
        <arg2 key="{var:maxlength}"
          name="maxlength"
          resource="false"/>
        <var>
          <var-name>maxlength</var-name>
          <var-value>16</var-value>
        </var>
        <var>
          <var-name>minlength</var-name>
          <var-value>3</var-value>
        </var>
      </field>
    </form>
    <form name="registrationForm">
      <field property="userid"
        depends="required,minlength,maxlength">
        <arg0 key="prompt.userid" />
```

```

    <arg1 key="{var:minlength}"
      name="minlength"
      resource="false" />
    <arg2 key="{var:maxlength}"
      name="maxlength"
      resource="false" />
    <var>
      <var-name>maxlength</var-name>
      <var-value>16</var-value>
    </var>
    <var>
      <var-name>minlength</var-name>
      <var-value>3</var-value>
    </var>
  </field>
  <field property="password"
    depends="required,minlength,maxlength"
    bundle="alternate">
    <arg0 key="prompt.password"/>
    <arg1 key="{var:minlength}"
      name="minlength"
      resource="false"/>
    <arg2 key="{var:maxlength}"
      name="maxlength"
      resource="false"/>
    <var>
      <var-name>maxlength</var-name>
      <var-value>16</var-value>
    </var>
    <var>
      <var-name>minlength</var-name>
      <var-value>3</var-value>
    </var>
  </field>
  <field property="email"
    depends="required,email">
    <arg0 key="prompt.email"/>
  </field>
</form>
</formset>
</form-validation>

```

The sql-map-config.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sql-map-config
  PUBLIC "-//IBATIS.com//DTD SQL Map Config 1.0//EN"
  "http://www.ibatis.com/dtd/sql-map-config.dtd">
<sql-map-config>
  <properties resource="properties/database.properties" />
  <datasource name="database"
    factory-class="com.ibatis.db.sqlmap.datasource.JndiDataSourceFactory"
    default="true" >

```

```

        <property name="DBInitialContext" value="java:comp/env" />
        <property name="DBLookup" value="{DatabaseJNDIPatn}"/>
    </datasource>
    <sql-map resource="sqlMap/SQL.xml" />
</sql-map-config>

```

The SQL.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sql-map PUBLIC "-//IBATIS.com//DTD SQL Map 1.0//EN"
"http://www.ibatis.com/dtd/sql-map.dtd">
<sql-map name="UserSQL">
    <cache-model name="user_cache"
        reference-type="WEAK">
        <flush-interval hours="24"/>
        <flush-on-execute statement="insertUser" />
        <flush-on-execute statement="updateUser" />
        <flush-on-execute statement="updateUserProfile" />
        <flush-on-execute statement="updateAccount" />
        <flush-on-execute statement="createAccount" />
    </cache-model>
    <!-- User Result Map -->
    <result-map name="user_result"
        class="myapp.User">
        <property name="userid"
            column="userid"/>
        <property name="password"
            column="password"/>
        <property name="email"
            column="email"/>
        <property name="ip"
            column="ip"/>
        <property name="random"
            column="random"/>
    </result-map>
    <!-- Result Map for AccountResult method -->
    <result-map name="account_result"
        class="myapp.Account">
        <property name="currency"
            column="currency"/>
        <property name="units"
            column="units"/>
    </result-map>
    <result-map name="trustPartner_result"
        class="myapp.TrustPartner">
        <property name="id"
            column="id"/>
        <property name="name"
            column="name"/>
    </result-map>
    <!-- Insert User functionality used for registration-->
    <mapped-statement name="insertUser">
        INSERT INTO users(userid, password,

```

```

        email, ip, random)
VALUES (#userid#, #password#,
        #email#, #ip#, #random#)
</mapped-statement>
<!-- Update functionality used for LoginAction -->
<mapped-statement name="updateUser">
    UPDATE users SET password = #password#,
        email = #email#,
        ip = #ip#,
        random = #random#,
    WHERE userid = #userid#
</mapped-statement>
<mapped-statement name="updateUserProfile">
    UPDATE users SET password = #password#,
        email = #email#
    WHERE userid = #userid#
</mapped-statement>
<!-- Extract User data from database. It is used to check if the user exist -->
<mapped-statement name="checkUser"
    result-map="user_result"
    cache-model="user_cache">
    SELECT * FROM users
    WHERE users.userid = #userid#
</mapped-statement>
<mapped-statement name="getUserByIp"
    result-map="user_result"
    cache-model="user_cache">
    SELECT * FROM users
    WHERE users.ip = #ip#
</mapped-statement>
<mapped-statement name="getUserByRandom"
    result-map="user_result"
    cache-model="user_cache">
    SELECT * FROM users
    WHERE users.random = #random#
</mapped-statement>
<!-- Extract specific currency account information. -->
<mapped-statement name="readAccount"
    result-map="account_result"
    cache-model="user_cache">
    SELECT * FROM account
    WHERE account.currency = #currency# AND
        account.userid = #userid#
</mapped-statement>
<!-- Insert new currency account inside database -->
<mapped-statement name="createAccount">
    INSERT INTO account (userid, currency, units)
    VALUES (#userid#, #currency#, #units#)
</mapped-statement>
<!-- Update the specific currency account data -->
<mapped-statement name="updateAccount">
    UPDATE account SET units = #units#
    WHERE account.userid = #userid# AND

```

```

        account.currency = #currency#
    </mapped-statement>
<!-- Extract User's account information from database
    Note: (1)We have to check if users.userid == account.userid
        (2)Use Dynamic mapped statement we can easy to handle
            query from different tables. Also, it makes our
            javacode clear and easy to read -->
    <dynamic-mapped-statement name="getAccountBalance"
        result-map="account_result"
        cache-model="user_cache">
        SELECT account.currency AS currency,
            account.units As units
        FROM account, users
        WHERE account.userid = #userid#
    <dynamic>
        <isNotEmpty prepend="AND" property="userid">
            users.userid = #userid#
        </isNotEmpty>
    </dynamic>
    </dynamic-mapped-statement>
    <mapped-statement name="getTrustPartners"
        result-map="trustPartner_result">
        SELECT * FROM trustpartner
    </mapped-statement>
    <mapped-statement name="getTrustPartnerByName"
        result-map="trustPartner_result">
        SELECT * FROM trustpartner
        WHERE name = #name#
    </mapped-statement>
</sql-map>

```

APPENDIX B

JSP FILE

### The chat.jsp:

```
<HTML><HEAD><TITLE>Java ChatRoom Demo</TITLE></HEAD>
<BODY BGCOLOR=Black TEXT=White Link=White VLink=White>
<CENTER>
<APPLET code="ChatRoom" WIDTH=650 HEIGHT=230>
    <param name="port" value="9999">
</APPLET>
</CENTER>
</BODY>
</HEAD>
```

### The accountBalance.jsp:

```
<%@ taglib uri="struts/html-el" prefix="html" %>
<%@ taglib uri="struts/bean" prefix="bean" %>
<%@ taglib uri="jstl/c" prefix="c" %>

<html>
<head>
    <center><h1>Account Balances</h1></center>
</head>
<body>
<center><table>
<tr>
    <td>
        <table border="1" cellpadding="2" cellspacing="1">
            <tr>
                <th>Currency</th>
                <th>Units</th>
            </tr>
            <tr>
                <td><c:forEach var="account" items="{accounts}">
                    <tr>
                        <td><c:out value="{account.currency}"/></td>
                        <td><c:out value="{account.units}"/></td>
                    </tr>
                </c:forEach>
            </td>
        </table>
    </td>
</tr>
</table></center>
</body>
</html>
```

### The error.jsp:

```
<%@ taglib uri="struts/bean-el" prefix="bean" %>
<%@ taglib uri="struts/html-el" prefix="html" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<html>
<head>
<link href="<html:rewrite page="/rr.css" />" rel="stylesheet" type="text/css">
```

```

<title><bean:message key="title.error"/></title>
</head>
<body>
<h1><bean:message key="title.error"/></h1>
<!-- this should be an include or, better yet, use tiles and make this a footer !-->
<html:link page="/index.jsp">Main Menu</html:link>
<br><br>
<html:messages id="error">
  <span id="error"><c:out value="{error}" escapeXml="false"/></span><br>
</html:messages>
<br><br>
</body>
</html>

```

### The externalTransfer.jsp:

```

<%@ taglib uri="struts/bean-el" prefix="bean" %>
<%@ taglib uri="struts/html" prefix="html" %>
<%@ taglib uri="struts/logic-el" prefix="logic" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<html>
<head>
  <center><h1>
    External Transfer Fund Page
  </h1></center>
</head>
<body>
<html:errors/>
<br><br>
<html:form action="/externalTransferAction" focus="currency">
  <table width="550" align="center">
    <tr>
      <th align="right">Currency Type:</th>
      <td align="left"><html:select property="currency">
        <html:options collection="accounts"
          property="currency"/>
      </html:select>
    </tr>
    <tr>
      <th align="right">Trusted Partner:</th>
      <td align="left"><html:select property="name">
        <html:options collection="trustPartners"
          property="name"/>
      </html:select>
    </tr>
    <tr>
      <th align="right">Units:</th>
      <td align="left"><html:text property="units" size="10"/>
    </tr>
    <tr>
      <th align="right">Recipient:</th>
      <td align="left"><html:text property="userid" size="20"/>
    </tr>
  </table>
</html:form>

```

```

</tr>
<tr>
  <th align="right">TransactionId:</th>
  <td align="left"><html:text property="transactionId" size="20"/>
</tr>
</table>
<br>
<center><html:submit>
  <bean:message key="button.submit"/>
</html:submit>
</html:form>
<br><br>
</body>
</html>

```

### The exTransferResult.jsp:

```

<%@ taglib uri="struts/html-el" prefix="html" %>
<%@ taglib uri="struts/bean" prefix="bean" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<html>
<head>
  <center><h1>Transfer Request Result Page</h1></center>
</head>
<body>
  <table border="1" align="center">
    <tr>
      <th align="right">Result:</th>
      <td align="left">
        <bean:write name="requestResult"
          property="result"
          scope="session"/>
      </td>
    </tr>
    <tr>
      <th align="right">Fee:</th>
      <td align="left">
        <bean:write name="requestResult"
          property="fee"
          scope="session"/>
      </td>
    </tr>
  </table>
</body>
</html>

```

### The login.jsp:

```

<%@ taglib uri="struts/bean" prefix="bean" %>
<%@ taglib uri="struts/html-el" prefix="html" %>
<%@ taglib uri="struts/logic-el" prefix="logic" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<html>

```

```

<head>
  <title><bean:message key="login.title"/></title>
  <center><h1><bean:message key="login.title"/></h1></center>
</head>
<body>
<html:errors/>
<br><br>
<html:form action="/loginAction" focus="userid"
  onsubmit="return validationUserForm(this);">
  <center><table>
    <tr><td>
      <table width="300">
        <tr>
          <th align="right">Userid:</th>
          <td align="left"><html:text property="userid" size="24"/></td>
        </tr>
        <tr>
          <th align="right">Password:</th>
          <td align="left"><html:password property="password" size="24"/></td>
        </tr>
      </table>
    </td></tr>
  </table></center>
  <br>
  <center><html:submit>
    <bean:message key="button.submit"/>
  </html:submit>&nbsp;  
  <html:reset>
    <bean:message key="button.reset"/>
  </html:reset></center><br>
  <center>
    <html:link forward="registration">Register a New Account</html:link>
    <html:link forward="information">Get More Information</html:link>
  </center>
</html:form>
<br><br>
</body>
</html>

```

The main.jsp:

```

<html>
<frameset rows="55%,45%">
  <frameset cols="15%,85%">
    <frame src="menu.jsp" name="left">
    <frame src="transferfund.jsp" name="right">
  </frameset>
  <frame src="messageBox.jsp" name="down">
</frameset>
</html>

```

### The menu.jsp:

```
<%@ taglib uri="struts/bean" prefix="bean" %>
<%@ taglib uri="struts/html-el" prefix="html" %>
<%@ taglib uri="struts/logic-el" prefix="logic" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<%@page import = "java.util.*, myapp.User" %>
<html>
<body>
  <h4>Hello, <bean:write name="user" property="userid" scope="session"/>
  <br><br>
  <html:link forward="editProfile"
    target="right">EditProfile</html:link><br><br>
  <html:link forward="accountBalance"
    target="right">AccountBalance</html:link><br><br>
  <html:link forward="transferFund"
    target="right">TransferFund</html:link><br><br>
  <html:link forward="exteriorTransfer"
    target="right">ExternalTransfer</html:link><br><br>
  <html:link forward="verificationRequest"
    target="right">VerificationRequest</html:link><br><br>
  <html:link forward="logout"
    target="_top">Logout</html:link><br><br>
</h4>
</body>
</html>
```

### The messageBox.jsp:

```
<%@ taglib uri="struts/bean" prefix="bean" %>
<%@ taglib uri="struts/html-el" prefix="html" %>
<%@ taglib uri="struts/logic-el" prefix="logic" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<%@page import = "java.util.*, myapp.User" %>

<html>
<body>
  <center>
    <% User user = (User) session.getAttribute("user");
    String random = user.getRandom(); %>
    <applet code="ChatRoom" WIDTH=720 HEIGHT=250>
      <param name="port" value="9999">
      <param name="rand" value="<%= random %>">
    </applet>
  </center>
</body>
</html>
```

### The profile.jsp:

```
<%@ taglib uri="struts/html-el" prefix="html" %>
<%@ taglib uri="struts/bean" prefix="bean" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<html>
```

```

<html:errors/>
<head>
  <center><h1>Profile Page</h1></center>
</head>
<body>
<html:form action="editProfileAction" focus="password">
  <table width="550" align="center">
    <tr>
      <th align="right">Userid:</th>
      <td align="left">
        <bean:write name="user" property="userid" scope="session"/>
        <html:hidden name="user" property="userid"/>
      </td>
    </tr>
    <tr>
      <th align="right">Password:</th>
      <td align="left"><html:password property="password" size="24"/></td>
    </tr>
    <tr>
      <th align="right">Password Again:</th>
      <td align="left"><html:password property="password2" size="24"/></td>
    </tr>
    <tr>
      <th align="right">Email Address:</th>
      <td align="left"><html:text name="user" property="email" size="50"/></td>
    </tr>
  </table>
  <br>
  <center><html:submit>
    <bean:message key="button.save"/>
  </html:submit>&nbsp;  
</html:form>
</body>
</html>

```

### The registration.jsp:

```

<%@ taglib uri="struts/bean-el" prefix="bean" %>
<%@ taglib uri="struts/html" prefix="html" %>
<%@ taglib uri="struts/logic-el" prefix="logic" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<html:html>
<head>
  <title><bean:message key="app.registration.title"/></title>
  <center><h1>
    <bean:message key="app.registration.title"/>
  </h1></center>
</head>
<body>
<html:errors/>
<html:form action="/registrationAction" focus="userid">
  <table width="550" align="center">
    <tr>

```

```

    <th align="right">Userid:</th>
    <td align="left"><html:text property="userid" size="24"/>
</tr>
<tr>
    <th align="right">Password:</th>
    <td align="left"><html:password property="password" size="24"/>
</tr>
<tr>
    <th align="right">Password Again:</th>
    <td align="left"><html:password property="password2" size="24"/>
</tr>
<tr>
    <th align="right">Email Address:</th>
    <td align="left"><html:text property="email" size="50"/>
</tr>
</table>
<br>
<center><html:submit>
    <bean:message key="button.submit"/>
</html:submit>&nbsp;
<html:reset>
    <bean:message key="button.reset"/>
</html:reset>&nbsp;
</html:form>
<br><br>
</body>
</html:html>

```

### The transferFund.jsp:

```

<%@ taglib uri="struts/bean-el" prefix="bean" %>
<%@ taglib uri="struts/html" prefix="html" %>
<%@ taglib uri="struts/logic-el" prefix="logic" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<html>
<head>
    <center><h1>
        Transfer Fund Page
    </h1></center>
</head>
<body>
<html:errors/>
<br><br>
<html:form action="/transferFundAction" focus="currency">
    <table width="550" align="center">
        <tr>
            <th align="right">Currency Type:</th>
            <td align="left"><html:select property="currency">
                <html:options collection="accounts"
                    property="currency"/>
                </html:select>
            </td>
        </tr>
    </table>
</html:form>

```

```

    <th align="right">Units:</th>
    <td align="left"><html:text property="units" size="10"/>
</tr>
<tr>
    <th align="right">Recipient:</th>
    <td align="left"><html:text property="userid" size="20"/>
</tr>
</table>
<br>
<center><html:submit>
    <bean:message key="button.submit"/>
</html:submit>
</html:form>
<br><br>
</body>
</html>

```

### The verificationRequest.jsp:

```

<%@ taglib uri="struts/bean-el" prefix="bean" %>
<%@ taglib uri="struts/html" prefix="html" %>
<%@ taglib uri="struts/logic-el" prefix="logic" %>
<%@ taglib uri="jstl/c" prefix="c" %>
<html>
<head>
    <center><h1>
        Verification Request Page
    </h1></center>
</head>
<body>
<html:errors/>
<br><br>
<html:form action="/verificationRequestAction" focus="currency">
    <table width="550" align="center">
        <tr>
            <th align="right">Currency Type:</th>
            <td align="left"><html:select property="currency">
                <html:options collection="accounts"
                    property="currency"/>
            </html:select>
        </tr>
        <tr>
            <th align="right">TransactionId:</th>
            <td align="left"><html:text property="transactionId" size="20"/>
        </tr>
    </table>
    <br>
    <center><html:submit>
        <bean:message key="button.submit"/>
    </html:submit>
</html:form>
<br><br>
</body>

```

```
</html>
```

### The verificationResult.jsp:

```
<%@ taglib uri="struts/html-el" prefix="html" %>
<%@ taglib uri="struts/bean" prefix="bean" %>
<%@ taglib uri="jstl/c" prefix="c" %>

<html>
<head>
  <center><h1>Verification Request Result Page</h1></center>
</head>
<body>
  <table border="1" align="center">
    <tr>
      <th align="right">
        <bean:write name="verification"
          property="name"
          scope="session"/>:
      </th>
      <td align="left">
        <bean:write name="verification"
          property="result"
          scope="session"/>
      </td>
    </tr>
    <tr>
      <th align="right">Fee:</th>
      <td align="left">
        <bean:write name="verification"
          property="fee"
          scope="session"/>
      </td>
    </tr>
  </table>
</body>
</html>
```

APPENDIX C  
JAVA SOURCE CODE

### The Account.java:

```
package myapp;

import java.io.Serializable;

public class Account implements Serializable {
    private String userid;
    private String currency;
    private double units;

    public Account() {
        userid = null;
        units = 0;
        currency = null;
    }

    //getting method.
    public String getUserid() { return userid; }
    public String getCurrency() { return currency; }
    public double getUnits() { return units; }

    //setting method.
    public void setUserid(String userid) { this.userid = userid; }
    public void setCurrency(String currency) {
        this.currency = currency;
    }
    public void setUnits(double units) { this.units = units; }
}
```

### The AccountDAO.java:

```
package myapp;

import java.util.List;

public class AccountDAO extends BaseDAO {

    public List getAccountBalance(Object parameterObject) throws DAOException {
        return super.getList("getAccountBalance", parameterObject);
    }

    public Object readAccount(Object parameterObject) throws DAOException {
        return super.getObject("readAccount", parameterObject);
    }

    public int createAccount(Object parameterObject) throws DAOException {
        return super.update("createAccount", parameterObject);
    }

    public int updateAccount(Object parameterObject) throws DAOException {
        return super.update("updateAccount", parameterObject);
    }
}
```

## The BaseDAO.java:

```
package myapp;
```

```
import java.io.Reader;
import java.sql.SQLException;
import java.util.List;
```

```
import com.ibatis.common.resources.Resources;
import com.ibatis.db.sqlmap.SqlMap;
import com.ibatis.db.sqlmap.XmlSqlMapBuilder;
```

```
public class BaseDAO {
    private static SqlMap sqlMap = null;
```

```
    static {
        try {
            String resource = "sqlMap/sql-map-config.xml";
            Reader reader = Resources.getResourceAsReader(resource);
            sqlMap = XmlSqlMapBuilder.buildSqlMap(reader);
            reader.close();
        } catch (Exception ex) {
            throw new RuntimeException("Error Initializing BaseDAO : " + ex);
        }
    }
}
```

```
public List getList(String statementName, Object parameterObject)
throws DAOException {
    List list = null;
```

```
    try {
        sqlMap.startTransaction();
        list = sqlMap.executeQueryForList(statementName, parameterObject*);
        sqlMap.commitTransaction();
    } catch (SQLException e) {
        try {
            sqlMap.rollbackTransaction();
        } catch (SQLException ex) {
            throw new DAOException(ex.fillInStackTrace());
        }
    }
}
```

```
    throw new DAOException(e.fillInStackTrace());
}
```

```
return list;
}
```

```
public Object getObject(String statementName, Object parameterObject)
throws DAOException {
    Object result = null;
```

```

try {
    sqlMap.startTransaction();
    result = sqlMap.executeQueryForObject(statementName, parameterObject);
    sqlMap.commitTransaction();
} catch (SQLException e) {
    try {
        sqlMap.rollbackTransaction();
    } catch (SQLException ex) {
        throw new DAOException(ex.fillInStackTrace());
    }
    throw new DAOException(e.fillInStackTrace());
}
return result;
}

public int update(String statementName, Object parameterObject)
throws DAOException {
    int result = 0;

    try {
        sqlMap.startTransaction();
        result = sqlMap.executeUpdate(statementName, parameterObject);
        sqlMap.commitTransaction();
    } catch (SQLException e) {
        try {
            sqlMap.rollbackTransaction();
        } catch (SQLException ex) {
            throw new DAOException(ex.fillInStackTrace());
        }
        throw new DAOException(e.fillInStackTrace());
    }

    return result;
}
}

```

The ChatServer.java:

```

package myapp;

import java.io.*;
import java.net.*;
import java.util.*;
import javax.net.ssl.*;
import javax.net.*;
import java.security.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class ChatServer extends HttpServlet {
    /** The Server Socket */

```

```

protected ServerSocket servSock;
/** The list of my current clients */
protected Hashtable clientHashTable;
protected ChatHandler c1;
/** Main just constructs a ChatServer, which should never return */

public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
throws ServletException, IOException {
    HttpSession session = request.getSession();

    String[] notifyMessage = (String[])
        session.getAttribute("notifyMessage");
    String[] logoutMessage = (String[])
        session.getAttribute("logoutMessage");

    /** This if condition deals with Logout Action process because
    when a user logout the system, the ChatServer have to close
    the socket connection with the user and remove the data from
    clientHashTable. */
    if(logoutMessage != null && logoutMessage[0].equals("logout")) {
        MessageHandler client =
            (MessageHandler) clientHashTable.get(logoutMessage[1]);
        client.close();
        clientHashTable.remove(logoutMessage[1]);
        session.removeAttribute("account");
        session.removeAttribute("accounts");
        session.removeAttribute("trustPartners");
        session.removeAttribute("logoutMessage");
        session.removeAttribute("notifyMessage");
        session.removeAttribute("user");
        String link = "/login/pages/login.jsp";
        response.sendRedirect(link);
    }
    /** This section deals with the notification problem. Once the user
    transfer fund to another user, the recipient should see some
    information showing on his(her) applet screen. */
    else {
        MessageHandler sender =
            (MessageHandler) clientHashTable.get(notifyMessage[1]);
        MessageHandler receiver =
            (MessageHandler) clientHashTable.get(notifyMessage[2]);

        if (receiver != null) {
            if(notifyMessage[0].equals("fail")) {
                receiver.send("Transaction Fail!");
                receiver.send("From: " + notifyMessage[1]);
                receiver.send("Reason: " + notifyMessage[3]);
            } else {
                receiver.send("Transaction Success!");
                receiver.send("You got " + notifyMessage[5] + " " +
                    notifyMessage[4] + " dollars from " +

```

```

        notifyMessage[1] + ".");
    }
}
}
}

public void init() {
    clientHashTable = new Hashtable();
    try {
        KeyManagerFactory kmf =
            KeyManagerFactory.getInstance("SunX509");
        KeyStore ks = KeyStore.getInstance("JKS");
        char[] passphrase = "changeit".toCharArray();
        ks.load(new FileInputStream("/root/.keystore"),
            passphrase);
        kmf.init(ks, passphrase);
        SSLContext ctx = SSLContext.getInstance("TLS");
        ctx.init(kmf.getKeyManagers(), null,
            new java.security.SecureRandom());
        SSLServerSocketFactory ssf =
            ctx.getServerSocketFactory();
        servSock = (SSLServerSocket) ssf.createServerSocket(9999);
        ((SSLServerSocket)servSock).setNeedClientAuth(false);
    } catch(Exception e) { return; }

    runServer(); // should never return.
}

public void destroy() {
    super.destroy();
    try {
        if(servSock == null) return;
        else servSock.close();
    } catch (Exception e) { return; }
}

public void runServer() {
    try {
        c1 = new ChatHandler();
        c1.start();
    } catch(Exception e) {
    } finally {
        Iterator iterator =
            clientHashTable.keySet().iterator();
        while(iterator.hasNext()) {
            clientHashTable.remove(iterator.next());
        }
        c1.interrupt();
        c1 = null;
    }
}

protected class MessageHandler extends Thread {

```

```

protected Socket clientSock;
protected BufferedReader is;
protected PrintWriter pw;
private ChatHandler next;
private String line;
private String userId;
public MessageHandler(Socket clientSock)
{
    this.clientSock = clientSock;
}

public void run() {
    try {
        is = new BufferedReader(
            new InputStreamReader(clientSock.getInputStream()));
        pw = new PrintWriter(clientSock.getOutputStream(), true);
    } catch (Exception e) {
        close();
        return;
    }

    try {
        line = is.readLine();
        User tempUser = new User();
        tempUser.setRandom(line);
        UserService service1 = new UserService();
        User user1 = (User) service1.getUserByRandom(tempUser);
        if (user1 == null) {
            close();
            return;
        }
        userId = user1.getUserid();
        send(userId);
        clientHashTable.put(userId, this);
        while ((line = is.readLine()) != null) {
            StringBuffer temp = new StringBuffer();
            temp.append(userId);
            String message;
            temp.append(": ");
            StringTokenizer tokenizer = new StringTokenizer(line, "|");
            String recipientId = tokenizer.nextToken();
            line = tokenizer.nextToken();
            temp.append(line);
            message = temp.toString();
            if (recipientId.equals("all")) {
                broadcast(message);
            } else {
                if (clientHashTable.get(recipientId) == null)
                    pw.println("User is not exist or online.");

                else if (recipientId.equals(userId))

```

```

        pw.println("You sent message to yourself.");

        else
            send(recipientId, message);
    }
} catch (Exception e) {
} finally {
    this.interrupt();
    clientHashTable.remove(userId);
    close();
}
}

protected void close() {
    if (clientSock == null) {
        return;
    }
    try {
        clientSock.close();
        clientSock = null;
    } catch (IOException e) {
    }
}

public void send(String id, String mesg) {
    pw.println(mesg);
    MessageHandler recipient =
        (MessageHandler) clientHashTable.get(id);
    recipient.send(mesg);
}

public void send(String mesg) {
    pw.println(mesg);
}

/** Send one message to all users */
public void broadcast(String mesg) {
    Set keySets = clientHashTable.keySet();
    Iterator iterator = keySets.iterator();
    while(iterator.hasNext()) {
        String key = (String) iterator.next();
        MessageHandler sib = (MessageHandler)
            clientHashTable.get(key);
        sib.send(mesg);
    }
}

/** Inner class to handle one conversation */
protected class ChatHandler extends Thread {

    protected Socket clientSock;

```

```

private MessageHandler messageHandler;
public ChatHandler() {}

public void run() {
    try {
        while(true) {
            messageHandler = new MessageHandler(servSock.accept());
            messageHandler.start();
        }
    } catch (Exception e) { return; }
}
}
}

```

### The CheckAccountBalanceAction.java:

```

package myapp;

import org.apache.commons.beanutils.BeanUtils;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.DynaActionForm;

import java.util.List;

import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CheckAccountBalanceAction extends Action {

    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
        throws Exception {
        ActionErrors errors = new ActionErrors();
        UserService service = new UserService();
        HttpSession session = request.getSession();
        User user = (User) session.getAttribute("user");
        Account account = new Account();
        account.setUserid(user.getUserid());
        List accounts = service.getAccountBalance(account);
        session.setAttribute("accounts", accounts);
        return mapping.findForward("success");
    }
}

```

### The DAOException.java:

```
package myapp;

public class DAOException extends Exception {

    public DAOException() { super(); }

    public DAOException(String message) { super(message); }

    public DAOException(Throwable cause) { super(cause); }

    public DAOException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

### The EditProfileAction.java:

```
package myapp;

import org.apache.commons.beanutils.BeanUtils;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.DynaActionForm;

import java.util.List;

import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class EditProfileAction extends Action {

    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
        throws Exception {
        ActionErrors errors = new ActionErrors();
        UserService service = new UserService();
        DynaActionForm registrationForm = (DynaActionForm) form;
        User user = new User();
        BeanUtils.copyProperties(user, registrationForm);

        if(!user.getPassword().equals(user.getPassword2())) {
            errors.add(ActionErrors.GLOBAL_ERROR,
                new ActionError("error.password.match"));
            saveErrors(request, errors);
        }
    }
}
```

```

        return mapping.getInputForward();
    }

    service.updateUserProfile(user);
    return mapping.findForward("success");
}
}

```

### The ExternalTransferAction.java:

```

package myapp;

import lightweightcurrency.client.*;
import org.apache.commons.beanutils.BeanUtils;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.DynaActionForm;
import org.apache.commons.beanutils.PropertyUtils;

import java.util.List;

import java.security.cert.Certificate;
import java.security.*;
import java.io.*;
import javax.net.*;
import javax.net.ssl.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ExternalTransferAction extends Action {
    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
        throws Exception {
        String sender = null;
        String issuer = null;
        String partnerPublicKey = null;
        String partnerName = null;
        String recipient = null;
        String currency = null;
        String units = null;
        String transactionId = null;
        UserService userService = new UserService();
        double amount;
        HttpSession session = request.getSession();

        User user = (User) session.getAttribute("user");
        sender = user.getUserid();
    }
}

```

```

    issuer =
        (String) PropertyUtils.getSimpleProperty(form, "currency");
    partnerName =
        (String) PropertyUtils.getSimpleProperty(form, "name");
    recipient =
        (String) PropertyUtils.getSimpleProperty(form, "userid");
    currency =
        (String) PropertyUtils.getSimpleProperty(form, "currency");
    units =
        (String) PropertyUtils.getSimpleProperty(form, "units");
    amount = Double.parseDouble(units);
    transactionId = recipient +
        (String) PropertyUtils.getSimpleProperty(form, "transactionId");
    TrustPartner partner = new TrustPartner();
    partner.setName(partnerName);
    TrustPartner receiverPartner =
        (TrustPartner) userService.getTrustPartnerByName(partner);
    partnerPublicKey = receiverPartner.getId();
    TransferRequestBean requestBean =
        new TransferRequestBean(partnerPublicKey, issuer,
            amount, transactionId);
    RequestType type = new RequestType("transferRequest");
    session.setAttribute("requestType", type);
    session.setAttribute("request", requestBean);

    return mapping.findForward("success");
}
}

```

### The LoginAction.java:

```

package myapp;

import org.apache.commons.beanutils.BeanUtils;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.DynaActionForm;

import java.util.List;
import java.util.Random;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LoginAction extends Action {
    protected List accounts = null;
    protected List trustPartners = null;
    protected List issuers = null;
    public ActionForward execute(ActionMapping mapping,

```

```

        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
throws Exception {
    ActionErrors errors = new ActionErrors();
    UserService service = new UserService();
    DynaActionForm userForm = (DynaActionForm) form;
    User loginUser = new User();
    User user = new User();
    BeanUtils.copyProperties(loginUser, userForm);
    user = (User) service.checkUser(loginUser);

    if (user == null) {
        errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.userid.login.failed"));
        saveErrors(request, errors);
        return mapping.getInputForward();
    }
    if (!loginUser.getPassword().equals(user.getPassword())) {
        errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.password.login.failed"));
        saveErrors(request, errors);
        return mapping.getInputForward();
    }
    user.setIp(request.getRemoteAddr());
    user.setRandom(Long.toString(new Random().nextLong()));
    user.setStatus("online");
    service.updateUser(user);
    Account account = new Account();
    account.setUserid(user.getUserid());
    accounts = service.getAccountBalance(account);
    TrustPartner trustPartner = new TrustPartner();
    trustPartners = service.getTrustPartners(trustPartner);
    HttpSession session = request.getSession();
    session.setAttribute("user", user);
    session.setAttribute("accounts", accounts);
    session.setAttribute("trustPartners", trustPartners);
    return mapping.findForward("success");
}
}

```

### The LogoutAction.java:

```

package myapp;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

```

```

import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * User logout.
 */

public class LogoutAction extends Action
{
// private Log log = LogFactory.getLog("myapp.LogoutAction");

/**
 * Process user logout request.
 * @return null if response object is completed, else return ActionForward
 */

public ActionForward execute( ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
throws Exception {

    HttpSession session = request.getSession();
    User user = (User) session.getAttribute("user");
    return mapping.findForward("success");
}

}

```

### The LogoutHandler.java:

```

package myapp;

import java.io.*;
import java.net.*;
import java.util.*;
import javax.net.ssl.*;
import javax.net.*;
import java.security.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class LogoutHandler extends HttpServlet {
    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
throws ServletException, IOException {
        HttpSession session = request.getSession();
        User user = (User) session.getAttribute("user");
        String[] logoutMessage = new String[2];

```

```

logoutMessage[0] = "logout";
logoutMessage[1] = user.getUserid();
response.setContentType("text/html");
PrintWriter out = response.getWriter();
session.setAttribute("logoutMessage", logoutMessage);
String link = "/chat";
RequestDispatcher dispatcher =
request.getRequestDispatcher(link);
dispatcher.forward(request, response);
}
}

```

### The TransferFundHandler.java:

```

package myapp;

import java.io.*;
import java.net.*;
import java.util.*;
import javax.net.ssl.*;
import javax.net.*;
import java.security.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class TransferFundHandler extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String link = "/chat";
        HttpSession session = request.getSession();
        String[] notifyMessage =
            (String[]) session.getAttribute("notifyMessage");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        RequestDispatcher dispatcher =
            request.getRequestDispatcher(link);
        dispatcher.include(request, response);
        out.println("<html>");
        out.println("<head><center>");
        out.println("<h1>Trasaction Result Page</h1>");
        out.println("</center></head>");
        out.println("<body><center><table border='1'>");
        if(notifyMessage[0].equals("fail")) {
            out.println("<tr><th align='right'>Transaction Result</th>");
            out.println("<td align='left'>" + notifyMessage[0] +
                "</td></tr>");
        }
        out.println("<tr><th align='right'>Sender ID</th>");
        out.println("<td align='left'>" + notifyMessage[1] +

```

```

        "</td></tr>");
    out.println("<tr><th align='right'>Recipient ID</th>");
    out.println("<td align='left'>" + notifyMessage[2] +
        "</td></tr>");
    out.println("<tr><th align='right'>Reason</th>");
    out.println("<td align='left'>" + notifyMessage[3] +
        "</td></tr>");
} else {
    out.println("<tr><th align='right'>Transaction Result</th>");
    out.println("<td align='left'>" + notifyMessage[0] +
        "</td></tr>");
    out.println("<tr><th align='right'>Sender ID</th>");
    out.println("<td align='left'>" + notifyMessage[1] +
        "</td></tr>");
    out.println("<tr><th align='right'>Recipient ID</th>");
    out.println("<td align='left'>" + notifyMessage[2] +
        "</td></tr>");
    out.println("<tr><th align='right'>Currency Type</th>");
    out.println("<td align='left'>" + notifyMessage[4] +
        "</td></tr>");
    out.println("<tr><th align='right'>Units</th>");
    out.println("<td align='left'>" + notifyMessage[5] +
        "</td></tr>");
}
}
out.println("</table></center></body></html>");
out.println("<body></body></html>");
}
}
}

```

### The RegistrationAction.java:

```

package myapp;

import org.apache.commons.beanutils.BeanUtils;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.DynaActionForm;

import java.util.List;
import java.util.Random;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class RegistrationAction extends Action {
    List accounts = null;
    List trustPartners = null;
    public ActionForward execute(ActionMapping mapping,
        ActionForm form,

```

```

        HttpServletRequest request,
        HttpServletResponse response)
throws Exception {
    ActionErrors errors = new ActionErrors();
    UserService service = new UserService();
    DynaActionForm registrationForm = (DynaActionForm) form;
    User user = new User();
    User applicant = new User();
    BeanUtils.copyProperties(applicant, registrationForm);
    user = (User) service.checkUser(applicant);

    if(user != null) {
        errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.userid.unique"));
        saveErrors(request,errors);
        return mapping.getInputForward();
    }

    if(!applicant.getPassword().equals(applicant.getPassword2())) {
        errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.password.match"));
        saveErrors(request, errors);
        return mapping.getInputForward();
    }

    applicant.setIp(request.getRemoteAddr());
    applicant.setRandom(Long.toString(new Random().nextLong()));
    applicant.setStatus("online");
    service.insertUser(applicant);
    HttpSession session = request.getSession();
    Account account = new Account();
    account.setUserid(applicant.getUserid());
    accounts = service.getAccountBalance(account);
    TrustPartner trustPartner = new TrustPartner();
    trustPartners = service.getTrustPartners(trustPartner);
    session.setAttribute("user", applicant);
    session.setAttribute("accounts", accounts);
    session.setAttribute("trustPartners", trustPartners);
    return mapping.findForward("success");
}
}

```

### The RequestHandleServlet.java:

```

package myapp;

import lightweightcurrency.client.*;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.net.ssl.*;
import javax.net.*;
import java.security.*;

```

```

import javax.servlet.*;
import javax.servlet.http.*;

public class RequestHandleServlet extends HttpServlet {
    private int messageCounter;
    private Vector requestQueue;
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession();
        User user = (User) session.getAttribute("user");
        RequestType r = (RequestType) session.getAttribute("requestType");
        String requestType = r.getType();
        String userName = user.getUserid();
        requestQueue.addElement(userName);
        Account clientAccount = new Account();
        UserService service = new UserService();
        double accountBalance;
        if (requestType.equals("transferRequest"))
        {
            TransferRequestBean requestBean =
                (TransferRequestBean) session.getAttribute("request");
            clientAccount = new Account();
            clientAccount.setUserid(userName);
            clientAccount.setCurrency(requestBean.getIssuer());
            try {
                clientAccount = (Account) service.readAccount(clientAccount);
            } catch (DAOException e) {
            }

            if (clientAccount == null)
            {
                clientAccount = new Account();
                clientAccount.setUserid(userName);
                clientAccount.setCurrency(requestBean.getIssuer());
                clientAccount.setUnits(0);
                try {
                    service.updateAccount(clientAccount);
                } catch (DAOException e) {
                }
                accountBalance = 0;
            }
            else
            {
                accountBalance = clientAccount.getUnits();
            }

            if (accountBalance < requestBean.getAmount())

```

```

{
    out.println("<html>");
    out.println("<head><center>");
    out.println("<h1>TransferRequest Result Page</h1>");
    out.println("</center></head>");
    out.println("<body><center><table border='1'>");
    out.println("<tr><th align='right'>Result:</th>");
    out.println("<td align='left'>Request Reject!</td></tr>");
    out.println("<tr><th align='right'>Reason:</th>");
    out.println("<td align='left'>Not enough funds</td></tr>");
    out.println("</table></center></body></html>");
    out.println("</html>");
}
else
{
    Request userRequest = new Request(userName, requestType,
        accountBalance, requestBean);
    try {
        userRequest.run();
    } catch (Exception e) {
        out.print("Error (TransferRequest) !");
    }
    TransferResult result = userRequest.getTransferResult();

    out.println("<html>");
    out.println("<head><center>");
    out.println("<h1>TransferRequest Result Page</h1>");
    out.println("</center></head>");
    out.println("<body><center><table border='1'>");
    out.println("<tr><th align='right'>Result:</th>");
    out.println("<td align='left'>" + result.getResult() +
        "</td></tr>");
    out.println("<tr><th align='right'>Fee:</th>");
    out.println("<td align='left'>" + result.getFee() +
        "</td></tr>");
    out.println("</table></center></body></html>");
    out.println("</html>");
    session.removeAttribute("requestType");
    session.removeAttribute("request");
}
}
else
{
    VerificationRequestBean requestBean =
        (VerificationRequestBean) session.getAttribute("request");
    clientAccount = new Account();
    clientAccount.setUserid(userName);
    clientAccount.setCurrency(requestBean.getIssuer());
    try {
        clientAccount = (Account) service.readAccount(clientAccount);
    } catch (DAOException e) {
    }
}
}

```

```

if (clientAccount == null)
{
    clientAccount = new Account();
    clientAccount.setUserid(userName);
    clientAccount.setCurrency(requestBean.getIssuer());
    clientAccount.setUnits(0);
    try {
        service.updateAccount(clientAccount);
    } catch (DAOException e) {
    }
    accountBalance = 0;
}
else
{
    accountBalance = clientAccount.getUnits();
}
Request userRequest = new Request(userName, requestType,
    accountBalance, requestBean);
try {
    userRequest.run();
} catch (Exception e) {
    out.print("Error (VerificationRequest) !");
}
VerificationResult result = userRequest.getVerificationResult();
out.println("<html>");
out.println("<head><center>");
out.println("<h1>VerificationRequest Result Page</h1>");
out.println("</center></head>");
out.println("<body><center><table border='1'>");
if (!result.getReason().equals(""))
{
    out.println("<tr><th align='right'>Reason:</th>");
    out.println("<td align='left'>" + result.getReason() +
        "</td></tr>");
}
else
{
    out.println("<tr><th align='right'>Reason:</th>");
    out.println("<td align='left'>" + result.getAmount() +
        "</td></tr>");
}
out.println("<tr><th align='right'>Fee:</th>");
out.println("<td align='left'>" + result.getFee() +
    "</td></tr>");
out.println("</table></center></body></html>");
out.println("</html>");
session.removeAttribute("requestType");
session.removeAttribute("request");
}
}

public void init() {
    messageCounter = 0;
}

```

```

    requestQueue = new Vector();
}

protected class Request {
    private String name;
    private String requestType;
    private double accountBalance;
    private TransferRequestBean transfer;
    private VerificationRequestBean verification;
    private TransferResult transferResult;
    private VerificationResult verificationResult;

    public Request(String name, String requestType,
        double accountBalance, TransferRequestBean transfer)
    {
        this.name = name;
        this.requestType = requestType;
        this.accountBalance = accountBalance;
        this.transfer = transfer;
    }
    public Request(String name,
        String requestType,
        double accountBalance,
        VerificationRequestBean verification)
    {
        this.name = name;
        this.requestType = requestType;
        this.accountBalance = accountBalance;
        this.verification = verification;
    }

    public void run() throws Exception
    {
        String userName;
        String issuer;
        while(!(userName = (String) requestQueue.get(0)).equals(name))
        {
            Thread.sleep(200);
        }
        if (transfer != null)
        {
            issuer = transfer.getIssuer();
        }
        else
        {
            issuer = verification.getIssuer();
        }
        ConnectionHandler c = new ConnectionHandler(issuer, 8888);
        c.setKeyStore("/root/.keystore", "changeit");
        c.setTrustStore("/root/.truststore", "changeit");
        c.run();
        SSLSocket s = (SSLSocket) c.getSocket();
        RequestHandler requestHandler = new RequestHandler(s);
    }
}

```

```

if (requestType.equals("transferRequest"))
{
    String recipient = transfer.getRecipient();
    double amount = transfer.getAmount();
    String transactionId = transfer.getTransactionId();
    transferResult = requestHandler.transferRequest(recipient,
        amount, transactionId, messageCounter);
    UserService service = new UserService();
    Account account = new Account();
    account.setUserid(userName);
    account.setCurrency(issuer);
    if (transferResult.getResult().equals("success"))
    {
        account.setUnits(accountBalance - amount -
            transferResult.getFee());
        service.updateAccount(account);
    }
    else
    {
        account.setUnits(accountBalance - transferResult.getFee());
        service.updateAccount(account);
    }
}
else
{
    String transactionId = verification.getTransactionId();
    verificationResult =
        requestHandler.verificationRequest(transactionId,
            messageCounter);
    UserService service = new UserService();
    Account account = new Account();
    account.setUserid(userName);
    account.setCurrency(issuer);
    if (!verificationResult.getReason().equals(""))
    {
        account.setUnits(accountBalance -
            verificationResult.getFee());
        service.updateAccount(account);
    }
    else
    {
        account.setUnits(accountBalance +
            verificationResult.getAmount() -
            verificationResult.getFee());
        service.updateAccount(account);
    }
}

if (messageCounter == 0) { messageCounter = 1; }
else { messageCounter = 0; }
requestQueue.removeElementAt(0);
}

```

```

public TransferResult getTransferResult()
{
    return transferResult;
}

public VerificationResult getVerificationResult()
{
    return verificationResult;
}
}
}

```

The RequestType.java:

```

package myapp;

public class RequestType
{
    private String type;

    public RequestType(String type)
    {
        this.type = type;
    }

    //getter method
    public String getType() { return type; }

    //setter method
    public void setType() { this.type = type; }
}

```

The TransferFundAction.java:

```

package myapp;

import org.apache.commons.beanutils.BeanUtils;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.DynaActionForm;
import org.apache.commons.beanutils.PropertyUtils;

import java.util.List;

import javax.servlet.*;
import javax.servlet.http.*;

public class TransferFundAction extends Action {

```

```

public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
throws Exception {

    String[] notifyMessage = new String[6];
    HttpSession session = request.getSession();
    ActionErrors errors = new ActionErrors();
    UserService service = new UserService();
    User recipient = new User();

    //Get the sender information from session
    User sender = (User) session.getAttribute("user");

    Account senderAccount = new Account();
    Account recipientAccount = new Account();

    // get sender userid
    String senderUserId = sender.getUserId();
    //get recipient userid
    String recipientUserId =
        (String) PropertyUtils.getSimpleProperty(form, "userid");
    //get currency type the sender wanna transfer.
    String currency =
        (String) PropertyUtils.getSimpleProperty(form, "currency");
    //get how many units the sender wanna transfer to recipient.
    String unitsString =
        (String) PropertyUtils.getSimpleProperty(form, "units");
    double units;
    if(currency.equals("")) {
        notifyMessage[0] = "fail";
        notifyMessage[1] = senderUserId;
        notifyMessage[2] = recipientUserId;
        notifyMessage[3] = "Don't have any currency type.";
        session.setAttribute("notifyMessage", notifyMessage);
        return mapping.findForward("success");
    }
    try {
        units = Double.parseDouble(unitsString);
    } catch (NumberFormatException ne) {
        notifyMessage[0] = "fail";
        notifyMessage[1] = senderUserId;
        notifyMessage[2] = recipientUserId;
        notifyMessage[3] = "Units field input error";
        session.setAttribute("notifyMessage", notifyMessage);
        return mapping.findForward("success");
    }

    recipient.setUserId(recipientUserId);
    senderAccount.setUserId(sender.getUserId());
    senderAccount.setCurrency(currency);
    senderAccount = (Account) service.readAccount(senderAccount);
}

```

```

recipient = (User) service.checkUser(recipient);
recipientAccount.setUserid(recipientUserid);
recipientAccount.setCurrency(currency);
recipientAccount = (Account) service.readAccount(recipientAccount);
//Check if sender owns any kind of currency
if(senderAccount == null) {
    senderAccount = new Account();
    notifyMessage[0] = "fail";
    notifyMessage[1] = senderUserid;
    notifyMessage[2] = recipientUserid;
    notifyMessage[3] = "this currency type inexistent.";
    session.setAttribute("notifyMessage", notifyMessage);
    return mapping.findForward("success");
}

//Check if recipient exists inside the database.
if(recipient == null) {
    notifyMessage[0] = "fail";
    notifyMessage[1] = senderUserid;
    notifyMessage[2] = recipientUserid;
    notifyMessage[3] = "The recipient is inexistent.";
    session.setAttribute("notifyMessage", notifyMessage);
    return mapping.findForward("success");
}

//Check if the sender has enough fund for this transaction.
if(senderAccount.getUnits() < units) {
    notifyMessage[0] = "fail";
    notifyMessage[1] = senderUserid;
    notifyMessage[2] = recipientUserid;
    notifyMessage[3] = "not have enough funds.";
    session.setAttribute("notifyMessage", notifyMessage);
    return mapping.findForward("success");
}

//Check if the recipient owns this kind of currency.

if (recipientAccount == null) {
    recipientAccount = new Account();
    recipientAccount.setUserid(recipient.getUserid());
    recipientAccount.setCurrency(currency);
    recipientAccount.setUnits(0);
    service.createAccount(recipientAccount);
}
else {
    recipientAccount.setUserid(recipient.getUserid());
}

//Renew both sender and recipient data.
recipientAccount.setUnits((recipientAccount.getUnits() + units));
senderAccount.setUnits((senderAccount.getUnits() - units));
service.updateAccount(recipientAccount);
senderAccount.setUserid(sender.getUserid());
service.updateAccount(senderAccount);

```

```

        notifyMessage[0] = "success";
        notifyMessage[1] = senderUserid;
        notifyMessage[2] = recipientUserid;
        notifyMessage[3] = "";
        notifyMessage[4] = currency;
        notifyMessage[5] = Double.toString(units);
        String test = "test";
        session.setAttribute("notifyMessage", notifyMessage);
        return mapping.findForward("success");
    }
}

```

### The TransferRequestBean.java:

```

package myapp;

public class TransferRequestBean
{
    private String recipient;
    private String issuer;
    private double amount;
    private String transactionId;

    public TransferRequestBean(String recipient, String issuer,
                               double amount, String transactionId)
    {
        this.recipient = recipient;
        this.issuer = issuer;
        this.amount = amount;
        this.transactionId = transactionId;
    }

    //getter method
    public String getRecipient() { return recipient; }
    public String getIssuer() { return issuer; }
    public double getAmount() { return amount; }
    public String getTransactionId() { return transactionId; }

    //setter method
    public void setRecipient() { this.recipient = recipient; }
    public void setIssuer() { this.issuer = issuer; }
    public void setAmount() { this.amount = amount; }
    public void setTransactionId() { this.transactionId = transactionId; }
}

```

### The TransferResultBean.java:

```

package myapp;

public class TransferResultBean
{
    private String result;
    private double fee;
}

```

```

public TransferResultBean(String result, double fee)
{
    this.result = result;
    this.fee = fee;
}

public TransferResultBean()
{
    result = "";
    fee = 0.0;
}

//getter method
public String getResult() { return result; }
public double getFee() { return fee; }

//setter method
public void setResult(String result) { this.result = result; }
public void setFee(double fee) { this.fee = fee; }
}

```

#### The TrustPartner.java:

```

package myapp;

import java.io.Serializable;

public class TrustPartner implements Serializable {
    private String id;
    private String name;

    public TrustPartner() {
        id = "";
        name = "";
    }

    //getting method.
    public String getId() { return id; }
    public String getName() { return name; }

    //setting method.
    public void setId(String id) { this.id = id; }
    public void setName(String name) { this.name = name; }
}

```

#### The TrustPartnerDAO.java:

```

package myapp;

import java.util.List;

public class TrustPartnerDAO extends BaseDAO {

```

```

public List getTrustPartners(Object parameterObject)
throws DAOException {
    return super.getList("getTrustPartners", parameterObject);
}
public Object getTrustPartnerByName(Object parameterObject)
throws DAOException {
    return super.getObject("getTrustPartnerByName", parameterObject);
}
}

```

### The User.java:

```

package myapp;

import java.io.Serializable;

public class User implements Serializable {
    private String userid;
    private String password;
    private String password2;
    private String email;
    private String ip;
    private String random;
    private String status;

    public User() {
        userid = null;
        password = null;
        password2 = null;
        email = null;
        ip = null;
        random = null;
        status = null;
    }

    public User(String userid) {
        this.userid = userid;
    }

    //getting method.
    public String getUserid() { return userid; }
    public String getPassword() { return password; }
    public String getPassword2() { return password2; }
    public String getEmail() { return email; }
    public String getIp() { return ip; }
    public String getRandom() { return random; }
    public String getStatus() { return status; }
    //setting method.
    public void setUserid(String userid) { this.userid = userid; }
    public void setPassword(String password) { this.password = password; }
    public void setPassword2(String password2) {
        this.password2 = password2;
    }
}

```

```

    }
    public void setEmail(String email) { this.email = email; }
    public void setIp(String ip) { this.ip = ip; }
    public void setRandom(String random) {
        this.random = random;
    }
    public void setStatus(String status) {
        this.status = status;
    }
}

```

The UserDao.java:

```

package myapp;

import java.util.List;

public class UserDao extends BaseDAO {

    public Object checkUser(Object parameterObject) throws DAOException {
        return super.getObject("checkUser", parameterObject);
    }

    public Object getUserByIp(Object parameterObject)
        throws DAOException {
        return super.getObject("getUserByIp", parameterObject);
    }

    public Object getUserByRandom(Object parameterObject)
        throws DAOException {
        return super.getObject("getUserByRandom",
            parameterObject);
    }

    public int insertUser(Object parameterObject) throws DAOException {
        return super.update("insertUser", parameterObject);
    }

    public int updateUser(Object parameterObject) throws DAOException {
        return super.update("updateUser", parameterObject);
    }

    public int updateUserProfile(Object parameterObject)
        throws DAOException {
        return super.update("updateUserProfile", parameterObject);
    }
}

```

## The UserService.java:

```
package myapp;

import java.util.List;

public class UserService {
    private static UserDAO userDAO = new UserDAO();
    private static AccountDAO accountDAO = new AccountDAO();
    private static TrustPartnerDAO trustPartnerDAO =
        new TrustPartnerDAO();

    //users Table
    public int insertUser(Object parameterObject) throws DAOException {
        return userDAO.insertUser(parameterObject);
    }

    public int updateUser(Object parameterObject) throws DAOException {
        return userDAO.updateUser(parameterObject);
    }

    public int updateUserProfile(Object parameterObject)
    throws DAOException {
        return userDAO.updateUserProfile(parameterObject);
    }

    public Object checkUser(Object parameterObject) throws DAOException {
        return userDAO.checkUser(parameterObject);
    }

    public Object getUserByIp(Object parameterObject)
    throws DAOException {
        return userDAO.getUserByIp(parameterObject);
    }

    public Object getUserByRandom(Object parameterObject)
    throws DAOException {
        return userDAO.getUserByRandom(parameterObject);
    }

    //account Table
    public List getAccountBalance(Object parameterObject) throws DAOException {
        return accountDAO.getAccountBalance(parameterObject);
    }

    public Object readAccount(Object parameterObject) throws DAOException {
        return accountDAO.readAccount(parameterObject);
    }

    public int createAccount(Object parameterObject) throws DAOException {
        return accountDAO.createAccount(parameterObject);
    }

    public int updateAccount(Object parameterObject) throws DAOException {
```

```

    return accountDAO.updateAccount(parameterObject);
}

//trustpartner Table
public List getTrustPartners(Object parameterObject)
throws DAOException {
    return trustPartnerDAO.getTrustPartners(parameterObject);
}
public Object getTrustPartnerByName(Object parameterObject)
throws DAOException {
    return trustPartnerDAO.getTrustPartnerByName(parameterObject);
}
}
}

```

The VerificationRequestAction.java:

```

package myapp;

import lightweightcurrency.client.*;
import org.apache.commons.beanutils.BeanUtils;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.DynaActionForm;
import org.apache.commons.beanutils.PropertyUtils;

import java.util.List;

import java.security.cert.Certificate;
import java.security.*;
import java.io.*;
import javax.net.*;
import javax.net.ssl.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class VerificationRequestAction extends Action {
    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
throws Exception {
    String issuer = null;
    String transactionId = null;
    String userName = null;
    HttpSession session = request.getSession();
    User user = (User) session.getAttribute("user");
    userName = user.getUserid();
    issuer =
        (String) PropertyUtils.getSimpleProperty(form, "currency");
}
}

```

```

transactionId = userName +
    (String) PropertyUtils.getSimpleProperty(form, "transactionId");
VerificationRequestBean requestBean =
    new VerificationRequestBean(issuer, transactionId);
RequestType type = new RequestType("verificationRequest");
session.setAttribute("requestType", type);
session.setAttribute("request", requestBean);
return mapping.findForward("success");
}
}

```

### The VerificatonRequestBean.java:

```

package myapp;

public class VerificationRequestBean
{
    private String issuer;
    private String transactionId;

    public VerificationRequestBean(String issuer, String transactionId)
    {
        this.issuer = issuer;
        this.transactionId = transactionId;
    }

    //getter method
    public String getIssuer() { return issuer; }
    public String getTransactionId() { return transactionId; }

    //setter method
    public void setIssuer() { this.issuer = issuer; }
    public void setTransactionId() { this.transactionId = transactionId; }
}

```

### The VerificationResultBean.java:

```

package myapp;

public class VerificationResultBean
{
    private String name;
    private String result;
    private double fee;

    public VerificationResultBean()
    {
        name = "";
        result = "";
        fee = 0.0;
    }
    public VerificationResultBean(String name, String result, double fee)
    {

```

```

    this.name = name;
    this.result = result;
    this.fee = fee;
}

//getter method
public String getName() { return name; }
public String getResult() { return result; }
public double getFee() { return fee; }

//setter method
public void setName(String name) { this.name = name; }
public void setResult(String result) { this.result = result; }
public void setFee(double fee) { this.fee = fee; }
}

```

### The ChatRoom.java:

```

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

import javax.net.*;
import java.security.*;
import javax.net.ssl.*;

public class ChatRoom extends Applet {
    /** Random Number used to verify the user */
    protected String rand;
    /** The actual port number */
    protected int port;
    /** The network socket */
    protected Socket sock;
    /** BufferedReader for reading from socket */
    protected BufferedReader is;
    /** PrintWriter for sending lines on socket */
    protected PrintWriter pw;
    /** TextField for receiver id */
    protected TextField receiverId;
    /** TextField for input */
    protected TextField tf;
    /** TextArea to display conversations */
    protected TextArea ta;
    /** The Send Message button */
    protected Button sendMessage;
    /** The Refresh button */
    protected Button refresh;
    /** The message that we paint */
    protected String paintMessage;
    /** Input Message */
    protected String line;
}

```

```

/** The Userid */
protected String userid;

/** Init, inherited from Applet */
public void init() {

    setLayout(new BorderLayout());
    String portNum = getParameter("port");
    rand = getParameter("rand");
    port = Integer.parseInt(portNum);

    ta = new TextArea(14, 80);
    ta.setEditable(false);           // readonly
    ta.setFont(new Font("Monospaced", Font.PLAIN, 11));
    add(BorderLayout.NORTH, ta);

    Panel p = new Panel();

    p.add(new Label("Receiver:"));
    receiverId = new TextField(10);
    p.add(receiverId);

    p.add(new Label("Message:"));
    tf = new TextField(30);
    p.add(tf);

    p.add(sendMessage = new Button("Send"));
    sendMessage.setEnabled(true);
    sendMessage.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String message;
            if(receiverId.getText().equals(""))
                message = "|all|" + tf.getText();
            else
                message = "|" + receiverId.getText() + "|" + tf.getText();
            pw.println(message);
            tf.setText("");
        }
    });

    p.add(refresh = new Button("Refresh"));
    refresh.setEnabled(true);
    refresh.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            ta.setText("");
        }
    });

    add(BorderLayout.SOUTH, p);
    startClient();
}

```

```

protected String serverHost = "localhost";

public void startClient() {
    serverHost = getCodeBase().getHost();

    TrustManager[] trustAllCerts = new TrustManager[] {
        new X509TrustManager() {
            public java.security.cert.X509Certificate[]
                getAcceptedIssuers() { return null; }

            public void checkClientTrusted(
                java.security.cert.X509Certificate[] certs,
                String authType) {}

            public void checkServerTrusted(
                java.security.cert.X509Certificate[] certs,
                String authType) {}
        }
    };

    try {
        SSLSocketFactory sf = null;
        SSLContext ctx = SSLContext.getInstance("TLS");
        ctx.init(null, trustAllCerts, new java.security.SecureRandom());
        sf = ctx.getSocketFactory();
        sock = sf.createSocket("stevenhsiao.no-ip.org",9999);
        is = new BufferedReader(
            new InputStreamReader(sock.getInputStream()));
        pw = new PrintWriter(sock.getOutputStream(), true);
        pw.println(rand);
    } catch (Exception e) {
        return;
    }

    new Thread(new Runnable() {
        public void run() {
            try {
                userId = is.readLine();
                while (((line = is.readLine()) != null))
                    ta.append(line + "\n");
            } catch (Exception e) {
                close();
                return;
            }
        }
    }).start();
}

public void close() {
    try {
        if (sock != null)

```

```

        sock.close();
    } catch (IOException ioe) { }
}
}

```

The ServerConnectionHandler.java:

```
package lightweightcurrency.client;
```

```
import java.io.*;
import java.security.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
```

```
public class ConnectionHandler {
    String keyStorePath;
    String keyStorePass;
    String trustStorePath;
    String trustStorePass;
    SSL.SocketFactory sf;
    String serverHostName;
    Socket socket;
    int port;

    public ServerConnectionHandler(String serverHostName, int port)
    {
        this.serverHostName = serverHostName;
        this.port = port;
    }

    public void setKeyStore(String keyStorePath, String keyStorePass)
    {
        this.keyStorePath = keyStorePath;
        this.keyStorePass = keyStorePass;
    }

    public void setTrustStore(String trustStorePath, String trustStorePass)
    {
        this.trustStorePath = trustStorePath;
        this.trustStorePass = trustStorePass;
    }

    public void run() throws Exception
    {
        KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
        KeyStore keyStore = KeyStore.getInstance("JKS");
        char[] keyStorePassPhrase = keyStorePass.toCharArray();
        keyStore.load(new FileInputStream(keyStorePath), keyStorePassPhrase);
        kmf.init(keyStore, keyStorePassPhrase);
        TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
        KeyStore trustStore = KeyStore.getInstance("JKS");
        char[] trustStorePassPhrase = trustStorePass.toCharArray();
    }
}

```

```

trustStore.load(new FileInputStream(trustStorePath), trustStorePassPhrase);
tmf.init(trustStore);

SSLContext ctx = SSLContext.getInstance("TLS");
ctx.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
sf = ctx.getSocketFactory();
socket = sf.createSocket(serverHostName, port);
}

public Socket getSocket()
{
    return socket;
}
}

```

### The ParserResult.java:

```

package lightweightcurrency.client;

import java.util.*;

/* This class is used to store the XML parse result */
public class ParserResult
{
    private String type;
    private String elementName;
    private String value;

    ParserResult(String type, String elementName, String value)
    {
        this.type = type;
        this.elementName = elementName;
        this.value = value;
    }

    ParserResult()
    {
        type = "";
        elementName = "";
        value = "";
    }

    /* getter methods */
    public String getType() { return type; }
    public String getElementName() { return elementName; }
    public String getValue() { return value; }

    /* setter methods */
    public void setType() { this.type = type; }
    public void setElementName() { this.elementName = elementName; }
    public void setValue() { this.value = value; }
}

```

## The RequestHandler.java:

```
package lightweightcurrency.client;

import java.io.*;
import java.net.*;
import java.util.Vector;
import java.security.cert.Certificate;
import javax.net.*;
import javax.net.ssl.*;
import java.security.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

public class RequestHandler {
    SSLSocket socket;
    Vector resultVector;
    InputStream is;
    PrintWriter pw;
    public RequestHandler(SSLSocket socket)
    {
        this.socket = socket;
    }

    Vector XMLParserResult(InputStream is)
    {
        Vector results = new Vector();
        try {
            DefaultHandler handler = new XMLHandler();
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            System.out.println("Before parse");
            saxParser.parse(is, handler);
            System.out.println("After parse");

            results = ((XMLHandler)handler).getResult();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return results;
    }

    public TransferResult transferRequest(String recipient, double amount,
                                         String transactionId, int messageCounter)
    throws IOException
    {
        String result = "";
        double fee = 0.0;
        TransferResult transferResult;
```

```

is = socket.getInputStream();
pw = new PrintWriter(socket.getOutputStream());
pw.println("<?xml version='1.0' encoding='utf-8'?>");
pw.println("<request>");
pw.println("<transferRequest>");
pw.println("<recipient>" + recipient + "</recipient>");
pw.println("<amount>" + Double.toString(amount) + "</amount>");
pw.println("<transactionId>" + transactionId + "</transactionId>");
pw.println("<messageCounter>" + Integer.toString(messageCounter) +
    "</messageCounter>");
pw.println("</transferRequest>");
pw.println("</request>");
pw.flush();
resultVector = XMLParserResult(is);
for (int i = 0; i < resultVector.size(); i++) {
    ParserResult parserResult = (ParserResult) resultVector.get(i);
    if (parserResult.getElementName().equals("result"))
    {
        result = parserResult.getValue();
    }
    else if (parserResult.getElementName().equals("fee"))
    {
        fee = Double.parseDouble(parserResult.getValue());
    }
}
resultVector.clear();
transferResult = new TransferResult(result, fee);
return transferResult;
}

```

```

public VerificationResult verificationRequest(String transactionId,
    int messageCounter)
throws IOException
{
    String reason = "";
    double amount = 0.0;
    double fee = 0.0;
    VerificationResult verificationResult;
    is = socket.getInputStream();
    pw = new PrintWriter(socket.getOutputStream());
    pw.println("<?xml version='1.0' encoding='utf-8'?>");
    pw.println("<request>");
    pw.println("<verificationRequest>");
    pw.println("<transactionId>" + transactionId + "</transactionId>");
    pw.println("<messageCounter>" + Integer.toString(messageCounter) +
        "</messageCounter>");
    pw.println("</verificationRequest>");
    pw.println("</request>");
    pw.flush();
    resultVector = XMLParserResult(is);
    for (int i = 0; i < resultVector.size(); i++) {
        ParserResult parserResult = (ParserResult) resultVector.get(i);
        if (parserResult.getElementName().equals("reason"))

```

```

    {
        reason = parserResult.getValue();
    }
    else if (parserResult.getElementName().equals("amount"))
    {
        amount = Double.parseDouble(parserResult.getValue());
    }
    else if (parserResult.getElementName().equals("fee"))
    {
        fee = Double.parseDouble(parserResult.getValue());
    }
}
resultVector.clear();
if (reason.equals(""))
{
    verificationResult = new VerificationResult(amount, fee);
}
else
{
    verificationResult = new VerificationResult(reason, fee);
}
return verificationResult;
}
}

```

The TransferResult.java:

```

package lightweightcurrency.client;

public class TransferResult
{
    private String result;
    private double fee;

    public TransferResult()
    {
        result = "";
        fee = 0.0;
    }

    public TransferResult(String result, double fee)
    {
        this.result = result;
        this.fee = fee;
    }

    /* getter methods */
    public String getResult() { return result; }
    public double getFee() { return fee; }

    /* setter methods */
    public void setResult(String result) { this.result = result; }
    public void setFee(double fee) { this.fee = fee; }
}

```

```

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class XMLHandler extends DefaultHandler
{
    StringBuffer textBuffer;
    String responseType;
    static Vector results = new Vector();
    ParserResult result;

    //=====
    // SAX DocumentHandler methods
    //=====

    public void startDocument()
    throws SAXException
    {
        System.out.println("StartDocument");
    }

    public void endDocument()
    throws SAXException
    {
        System.out.println("EndDocument");
    }

    public void startElement(String namespaceURI,
        String sName, // simple name
        String qName, // qualified name
        Attributes attrs)
    throws SAXException
    {
        String eName = sName; // element name
        if ("".equals(eName)) eName = qName; // not namespaceAware

        if (qName.equals("transferResult") ||
            qName.equals("verificationResult"))
        {
            responseType = qName;
        }
        if (attrs != null)
        {
            for (int i = 0; i < attrs.getLength(); i++)
            {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                String attrsValue = (String) attrs.getValue(i);
                result = new ParserResult(responseType, aName, attrsValue);
                results.addElement(result);
            }
        }
    }
}

```

```

public void endElement(String namespaceURI,
                      String sName, // simple name
                      String qName // qualified name
                      )
throws SAXException
{
    if (textBuffer != null)
    {
        String value = textBuffer.toString();
        if(!value.trim().equals(""))
        {
            result = new ParserResult(responseType, qName, value);
            textBuffer = null;
            results.addElement(result);
        }
    }
}

public void characters(char buf[], int offset, int len)
throws SAXException
{
    String s = new String(buf, offset, len);
    textBuffer = new StringBuffer(s);
}

public Vector getResult() throws Exception
{
    return results;
}
}

```

### The Client.java:

```

import java.io.*;
import java.sql.*;
import java.net.*;
import java.util.Random;
import java.util.Vector;
import java.security.*;
import java.security.interfaces.*;
import java.security.cert.Certificate;
import javax.net.*;
import javax.net.ssl.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

/* This Class handles all operations for this client */
public class Client
{
    private SSLSocket socket;

```

```

/* Fee Setting */
private double transferFeeRate;
private double verificationFeeRate;

/* DBSetting parameters */
private String DBSystemName;
private String DBDriverName;
private String DBName;
private String DBPort;
private String DBUserName;
private String DBPassword;

/* The setting of account Table */
private String accountTable;
private String accountId;
private String accountAmount;

/* The setting of deposit table */
private String depositTable;
private String depositId;
private String depositTransactionId;
private String depositAmount;

/* The backup Table saves the last response for each client. Once some
   unpredictable situation such as data lost and middle hackers attack
   happened the client can send same request XML document and messageCounter.
   Then, the issuer will check the table first and send the response back
   to the client again */
private String backupTable;
private String backupId;
private String backupRequestType;
private String backupTransactionId;
private String backupMessageCounter;
private String backupResponse;

/* I/O handler */
BufferedReader br;
PrintWriter pw;

/*Request user public key */
private String senderId;

/*DB connection requirement */
private Connection connection;
private Statement statement;
private ResultSet senderResultSet;
private ResultSet depositCheckResult;

/* Store the XMLParser result */
private Vector resultVector;

/* Either transferRequest or verificationRequest */

```

```

private String requestType;

/* transferRequest(verificationRequest) function parameters */
private String recipient;
private String transactionId;
private double amount;
private int messageCounter;

/* This constructor sets the defaults setting for the class */
Client(SSLSocket socket)
{
    this.socket = socket;
    transferFeeRate = 0;
    verificationFeeRate = 0;

    /* account table default setting */
    accountTable = "account";
    accountId = "id";
    accountAmount = "amount";

    /* deposit table default setting */
    depositTable = "deposit";
    depositId = "id";
    depositTransactionId = "transactionid";
    depositAmount = "amount";

    /* backup table default setting */
    backupTable = "backup";
    backupId = "id";
    backupRequestType = "requesttype";
    backupTransactionId = "transactionid";
    backupMessageCounter = "messagecounter";
    backupResponse = "response";
}

/* Get the default Database setting. The default setting is:
    DB System: postgresql
    Driver: postgresql JDBC Driver
    Access DB Name: lcp1
    Port: 5432
    UserName: tester
    Password: 1234
*/
void getDefaultDBSetting()
{
    DBSystemName = "postgresql";
    DBDriverName = "org.postgresql.Driver";
    DBName = "lcp1";
    DBPort = "5432";
    DBUserName = "tester";
    DBPassword = "1234";
}

```

```

/* This method allows developer to modify the DBSetting. For instance,
   a developer can use MySQL, Oracle or any other DB system with different
   Setting. */
void DBSetting(String DBSystemName, String DriverName, String DBName,
               String DBPort, String DBUserName, String DBPassword)
{
    this.DBSystemName = DBSystemName;
    this.DBDriverName = DBDriverName;
    this.DBName = DBName;
    this.DBName = DBName;
    this.DBPort = DBPort;
    this.DBUserName = DBUserName;
    this.DBPassword = DBPassword;
}

/* Modify the transferRequest fee rate. */
void setTransferFeeRate(double transferFeeRate)
{
    this.transferFeeRate = transferFeeRate;
}

/* Modify the verificationRegeust fee rate. */
void setVerificationFeeRate(double verificationFeeRate)
{
    this.verificationFeeRate = verificationFeeRate;
}

/* This method allows a developer to set the account table by using
   different from the original table setting. */
void setAccountTable(String accountTable, String accountId,
                    String accountAmount)
{
    this.accountTable = accountTable;
    this.accountId = accountId;
    this.accountAmount = accountAmount;
}

/* This method allows a developer to set the deposit table by using
   different from the original table setting. */
void setDepositTable(String depositTable, String depositId,
                    String depositTransactionId, String depositAmount)
{
    this.depositTable = depositTable;
    this.depositId = depositId;
    this.depositTransactionId = depositTransactionId;
    this.depositAmount = depositAmount;
}

/* This method allows a developer to set the backup table by using
   different from the original table setting. */
void setBackupTable(String backupTable, String backupId,
                   String backupRequestType, String backupTransactionId,
                   String backupMessageCounter, String backupResponse)

```

```

{
    this.backupTable = backupTable;
    this.backupId = backupId;
    this.backupRequestType = backupRequestType;
    this.backupTransactionId = backupTransactionId;
    this.backupMessageCounter = backupMessageCounter;
    this.backupResponse = backupResponse;
}

void run() throws Exception
{
    /* Try to get the client public key from SSL handshake */
    socket.startHandshake();
    Certificate[] clientCerts =
        socket.getSession().getPeerCertificates();
    senderId = clientCerts[0].getPublicKey().toString();

    /* Prepare Database connection setting */
    Class.forName(DBDriverName);
    connection = DriverManager.getConnection( "jdbc:" + DBSystemName +
        "://127.0.0.1:" + DBPort + "/" + DBName + "?user=" + DBUserName +
        "&password=" + DBPassword);
    statement = connection.createStatement();
    statement.setEscapeProcessing(false);

    /* If the client doesn't exist in database, the system forbid any
    later operation. */
    if (!userAccountCheck(senderId))
    {
        try {
            socket.close();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
    else
    {
        /* Open a temp file to store the XML document */
        String random = Long.toString(new Random().nextLong());
        File file = new File("temp" + random + ".xml");
        FileWriter fw = new FileWriter("temp" + random + ".xml");
        br = new BufferedReader(new InputStreamReader(
            socket.getInputStream()));
        pw = new PrintWriter(socket.getOutputStream());

        /* leave the loop when we read the end of the document */
        while(true) {
            String line = br.readLine().trim();
            System.out.println(line);
            fw.write(line);
            int len = line.length();
            if (len > 9 &&
                line.substring(len-10,len).equals("</request>"))

```

```

    {
        try {
            fw.close();
            break;
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}

/* Handle the XML parse operation */
DefaultHandler handler = new XMLHandler();
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
saxParser.parse(file, handler);
file.delete();

/* Pull out the parser result from a result vector */
resultVector = ((XMLHandler)handler).getResult();
ParserResult parserResult = new ParserResult();
if (resultVector.size() > 0)
{
    parserResult = (ParserResult) resultVector.get(0);
    requestType = parserResult.getType();
}
else
{
    System.out.println("ResultVector is empty!");
    return;
}
for (int i = 0; i < resultVector.size(); i++) {
    parserResult = (ParserResult) resultVector.get(i);
    String elementName = parserResult.getElementName();
    if (!parserResult.getType().equals(requestType))
    {
        System.out.println("XML format incorrect!");
        return;
    }
    if (elementName.equals("recipient"))
    {
        recipient = parserResult.getValue();
    }
    else if (elementName.equals("amount"))
    {
        amount = Double.parseDouble(parserResult.getValue());
    }
    else if (elementName.equals("transactionId"))
    {
        transactionId = parserResult.getValue();
    }
    else if (elementName.equals("messageCounter"))
    {
        messageCounter = Integer.parseInt(parserResult.getValue());
    }
}

```

```

    }
    System.out.println(resultVector.size());
    System.out.println("First: " + parserResult.getType() + " Second: " +
        parserResult.getElementName() + " Third: " +
        parserResult.getValue());
}

/* Analysis the request type, then call a appropriate method */
if (requestType.equals("transferRequest"))
{
    transferRequest(recipient, amount, transactionId, messageCounter);
}
else if (requestType.equals("verificationRequest"))
{
    verificationRequest(transactionId, messageCounter);
}
}
}

/* This method deals with the client transferRequest message */
void transferRequest(String recipient, double amount, String transactionId,
    int messageCounter) throws SQLException
{
    String query = "SELECT * FROM " + accountTable + " WHERE " +
        accountId + "=" + recipient + ",";
    String result = "";
    String update = "";
    double senderBalance;
    double recipientBalance;
    double fee;
    ResultSet recipientResultSet;

    if (isSameRequest("transferRequest", transactionId, messageCounter)) {
        return;
    }

    update = "DELETE FROM " + backupTable + " WHERE " +
        backupId + "=" + senderId + ",";
    statement.executeUpdate(update);

    /* If the transactionId already exists in deposit table,
    issuer send the fail message back to sender */
    if (depositTableCheck(recipient, transactionId))
    {
        transferResponse("same transactionId exists", 0);
        return;
    }

    recipientResultSet = statement.executeQuery(query);
    senderBalance = senderResultSet.getDouble(accountAmount);
    /* If the sender doesn't has enough funds, this transaction fail. */
    if (senderResultSet.getDouble(accountAmount) <

```

```

    (amount + transferFeeRate * amount))
    {
        transferResponse("not enough funds", 0);
        return;
    }
    /* If the recipient is inexist in the database, then the system create
    one account for recipient. */
    else if (!recipientResultSet.next())
    {
        update = "INSERT INTO " + accountTable + " VALUES(" +
            recipient + "," + amount + ");";
        statement.executeUpdate(update);
    }
    /* If the recipient already exists, just update the recipient data. */
    else
    {
        recipientBalance = recipientResultSet.getDouble(accountAmount);
        update = "UPDATE " + accountTable + " SET " +
            accountAmount + "=" + (recipientBalance + amount) +
            " WHERE " + accountId + "=" + recipient + ",";
        statement.executeUpdate(update);
    }
    /* update sender account information */
    update = "UPDATE " + accountTable + " SET " + accountAmount + "="
        + (senderBalance - amount * (1 + transferFeeRate)) +
        " WHERE " + accountId + "=" + senderId + ",";
    statement.executeUpdate(update);

    /* insert new data inside deposit table */
    update = "INSERT INTO " + depositTable + " VALUES(" +
        recipient + "," + transactionId + "," +
        amount + ");";
    statement.executeUpdate(update);
    recipientResultSet.close();
    fee = amount * transferFeeRate;
    transferResponse("success", fee );
}

/* This method deals with the client verificationRequest message */
void verificationRequest(String transactionId, int messageCounter)
throws SQLException
{
    String query = "";
    String update = "";
    ResultSet verificationResult;
    if (isSameRequest("verificationRequest", transactionId, messageCounter))
    {
        return;
    }
    update = "DELETE FROM " + backupTable + " WHERE " +
        backupId + "=" + senderId + ",";
    statement.executeUpdate(update);
}

```

```

query = "SELECT * FROM " + depositTable + " WHERE " +
        depositId + "=" + senderId + " AND " +
        depositTransactionId + "=" + transactionId + ",";
verificationResult = statement.executeQuery(query);
if (!verificationResult.next())
{
    verificationResponse("inexistent transactionId", 0);
}
else
{
    update = "DELETE FROM " + depositTable + " WHERE " +
            depositId + "=" + senderId + " AND " +
            depositTransactionId + "=" + transactionId + ",";
    statement.executeUpdate(update);
    double amount = verificationResult.getDouble(depositAmount);
    double fee = amount * verificationFeeRate;
    verificationResponse(amount, fee);
}
verificationResult.close();
}

/* Check if the request is same as the last request. If it's same,
just directly send the backup response message back to the client. */
boolean isSameRequest(String requestType, String transactionId,
int messageCounter)
{
    String query = "SELECT * FROM " + backupTable + " WHERE " +
            backupId + "=" + senderId + " AND " +
            backupRequestType + "=" + requestType + " AND " +
            backupTransactionId + "=" + transactionId + " AND " +
            backupMessageCounter + "=" + messageCounter + ",";
    try {
        ResultSet backupCheckResult = statement.executeQuery(query);
        if (backupCheckResult.next())
        {
            sendBackupResponse(backupCheckResult.getString(backupResponse));
            backupCheckResult.close();
            return true;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}

/* Once the request message is same as the last message. isSameRequest
function will call this method to send out the response message. */
void sendBackupResponse(String response)
{
    pw.println(response);
    pw.flush();
    try {
        resultVector.clear();
    }
}

```

```

senderResultSet.close();
connection.close();
socket.close();
} catch (Exception e) {
    e.printStackTrace();
}
}

/* Prepare the success verificationRequest message. Then send it
back to the client */
void verificationResponse(double amount, double fee)
{
    String response = "<?xml version='1.0' encoding='utf-8'?>\n" +
        "<response>\n" +
        "<verificationResult>\n" +
        "<amount>" + Double.toString(amount) + "</amount>\n" +
        "<fee>" + Double.toString(fee) + "</fee>\n" +
        "</verificationResult>\n" +
        "</response>";

    pw.println(response);
    pw.flush();

    String update = "INSERT INTO " + backupTable + " VALUES(" +
        senderId + ", " + "verificationRequest" + ", " +
        "" + transactionId + ", " +
        messageCounter + ", " + response + ")";

    try {
        statement.executeUpdate(update);
    } catch (SQLException e) {
        e.printStackTrace();
    }

    try {
        resultVector.clear();
        senderResultSet.close();
        statement.close();
        connection.close();
        socket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/* Prepare the fail verificationRequest message. Then send it back
to the client. */
void verificationResponse(String reason, double fee)
{
    String response = "<?xml version='1.0' encoding='utf-8'?>\n" +
        "<response>\n" +
        "<verificationResult reason='" + reason + "'>\n" +
        "<fee>" + Double.toString(fee) + "</fee>\n" +
        "</verificationResult>\n" +
        "</response>";
}

```

```

pw.println(response);
pw.flush();

String update = "INSERT INTO " + backupTable + " VALUES(" +
    senderId + "," + "verificationRequest" + "," +
    "" + transactionId + "," +
    messageCounter + "," + "response + ")";

try {
    statement.executeUpdate(update);
} catch (SQLException e) {
    e.printStackTrace();
}

try {
    resultVector.clear();
    senderResultSet.close();
    statement.close();
    connection.close();
    socket.close();
} catch (Exception e) {
    e.printStackTrace();
}

}

/* Prepare the transferResponse message. Then send it back to the client. */
void transferResponse(String result, double fee)
{
    String response = "<?xml version='1.0' encoding='utf-8'?'>\n" +
        "<response>\n" +
        "<transferResult>\n" +
        "<result>" + result + "</result>\n" +
        "<fee>" + Double.toString(fee) + "</fee>\n" +
        "</transferResult>\n" +
        "</response>";

    pw.println(response);
    pw.flush();

    String update = "INSERT INTO " + backupTable + " VALUES(" +
        senderId + "," + "transferRequest" + "," +
        "" + transactionId + "," +
        messageCounter + "," + "response + ")";

    try {
        statement.executeUpdate(update);
    } catch (SQLException e) {
        e.printStackTrace();
    }

    try {
        resultVector.clear();
        senderResultSet.close();
        statement.close();
        connection.close();
        br.close();
        socket.close();
    }
}

```

```

    } catch (Exception e) {
        e.printStackTrace();
    }
}

/* Check if the client owns a account.*/
boolean userAccountCheck(String id)
{
    String query = "SELECT * FROM " + accountTable + " WHERE " +
        accountId + "=" + id + ",";
    try {
        senderResultSet = statement.executeQuery(query);

        if (!senderResultSet.next())
        {
            System.out.println("This id is inexistent.");
            return false;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return true;
}

/* Check if the same id and transactionId already exist in the
deposit table */
boolean depositTableCheck(String id, String transactionId)
{
    String query = "SELECT * FROM " + depositTable + " WHERE " +
        depositId + "=" + id + " AND " +
        depositTransactionId + "=" + transactionId + ",";
    try {
        depositCheckResult = statement.executeQuery(query);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    try {
        if (!depositCheckResult.next())
        {
            System.out.println("Test here");
            return false;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return true;
}
}

```

## The ClientConnectionHandler.java:

```
import java.io.*;
import java.security.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;

public class ClientConnectionHandler {
    String keystorePath;
    String keystorePass;
    SSLServerSocketFactory ssf;
    ServerSocket serverSocket;
    int port;
    static TrustManager[] trustAllCerts = new TrustManager[] {
        new X509TrustManager()
        {
            public java.security.cert.X509Certificate[]
            getAcceptedIssuers()
            {
                return new java.security.cert.X509Certificate[0];
            }
            public void checkClientTrusted(
                java.security.cert.X509Certificate[] certs,
                String authType) { }

            public void checkServerTrusted(
                java.security.cert.X509Certificate[] certs,
                String authType) { }
        }
    };

    ClientConnectionHandler(String keystorePath, String keystorePass, int port)
    {
        this.keystorePath = keystorePath;
        this.keystorePass = keystorePass;
        this.port = port;
    }

    public void run() throws Exception
    {
        KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
        KeyStore ks = KeyStore.getInstance("JKS");
        char[] passphrase = keystorePass.toCharArray();
        ks.load(new FileInputStream(keystorePath), passphrase);
        kmf.init(ks, passphrase);
        SSLContext ctx = SSLContext.getInstance("TLS");
        ctx.init(kmf.getKeyManagers(), trustAllCerts, null);
        ssf = ctx.getServerSocketFactory();
        serverSocket = (SSLServerSocket) ssf.createServerSocket(port);
        ((SSLServerSocket)serverSocket).setNeedClientAuth(true);
    }
}
```

```

public ServerSocket getServerSocket()
{
    return serverSocket;
}
}

```

### The SSLSimpleServer.java:

```

import java.io.*;
import java.sql.*;
import java.net.*;
import java.util.Random;
import java.util.Vector;
import java.security.*;
import java.security.interfaces.*;
import java.security.cert.Certificate;
import javax.net.*;
import javax.net.ssl.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class SSLSimpleServer extends Thread {
    public static void main(String[] args) throws Exception {
        ClientConnectionHandler connectionHandler =
            new ClientConnectionHandler("/root/.keystore", "changeit", 8888);
        connectionHandler.run();
        ServerSocket ss = connectionHandler.getServerSocket();
        while (true) {
            new SSLSimpleServer(ss.accept()).start();
        }
    }

    private SSLSocket sock;
    public SSLSimpleServer(Socket s) throws Exception {
        sock = (SSLSocket) s;
    }

    public void run() {
        try {
            Client client = new Client(sock);
            client.getDefaultDBSetting();
            client.run();
        } catch (Exception e) {
            System.out.println("Errors");
        }
    }
}

```

APPENDIX D  
ACRONYMS AND ABBREVIATIONS

Terminology	Definition
ANT	Apache Ant. A Java-based build tool.
HTTP	Hypertext Transfer Protocol. The client/server protocol that defines how the messages are formatted and transmitted over World Wide Web.
HTTPS	The HTTP protocol layered over the Secure Socket Layer protocol to allow secure data transfer through a secure channel.
Java	An object-oriented language. Java programs are capable of running most popular platforms without recompilation.
Java Servlet	A Java program that extends the functionality of a Web server by generating dynamic content and by interacting with Web clients using a request-response model.
JSP	Java Server Page. An extension to the Java Servlet. JSP is available to display dynamic content on a web page.
Java Applet	An executable, Java component that executes in a Web browser or in any environment that supports the applet programming model.
Struts	A Java open source web application developing framework which helps programmers build robust systems.
iBTAIS Database Layer	A Java open source database connection bridge framework.
JDBC	Java Database Connectivity. A programming interface that lets Java programs access a database via SQL language.
LCP	Lightweight Currency Protocol. A protocol defines the format and standard for exchanging lightweight currency.
SSL	An acronym for Secure Socket Layer (SSL)—a protocol that allows communication between a Web browser and a server to be encrypted for privacy.
JSDK	Java Standard Develop Kit. It is used to compile and run the Java programs.
Jakarta Tomcat	An open source web server and container.

## REFERENCES

- [1]. Brett McLaughlin. *Programming Jakarta Struts*, O'Reilly and Associates, 2003.
- [2]. Clinton Begin. *iBATIS Database Layer Developer Guide*. <http://www.ibatis.com>, Oct 12, 2003
- [3]. David A. Turner. "Collaborative Research: Launching the P2P Resource Market", 2003.
- [4]. David A. Turner, Daniel Havey, and John Ewart. "Allocating Resources in Storage Cooperatives with Pseudo Currencies. Accepted for presentation at International Conference on Computer Science and its Applications", July 2003.
- [5]. David A. Turner and Keith W. Ross. "A Lightweight Currency Paradigm for the P2P Resource Market". 2003.
- [6]. David A. Turner and Keith W. Ross. "The Lightweight Currency Protocol Internet Draft", September 18, 2003.
- [7]. Ghaly Kothapalli. *Teach Yourself EJB in 21 Days*. SAMS, 2001.
- [8]. Jayson Falkner, et al. *Beginning JSP Web Development*. Wrox Press Inc, 1st Edition, August 2001.
- [9]. Jason Hunter and William Crawford. *Java Servlet Programming Second Edition*. O'Reilly and Associates, 2002
- [10]. Meng-Hsi Tsai. "IEPIS II (International Extension Programs Information System Version II) - An MVC Model 2 Design Using Apache Jakarta Struts", Master project of Computer Science Department, CSUSB, 2003.
- [11]. Petra J. Recter, et al. *How to Program JAVA, 5th Edition*. 2003.
- [12]. Petra J. Recter, et al. *How to Program Advanced Java 2 Platform*. 2002.

- [13]. PostgreSQL 7.3.2 Documentation.  
<http://www.postgresql.com/docs/>, September 2002.
- [14]. Scott Oaks. Java Security, 2nd Edition. O'Reilly and Associates, 2001.
- [15]. SSL 3.0 Specification.  
<http://wp.netscape.com/eng/ssl3/>, November 1996.