Micro-payment exchange system

Yiyao Hao
MICRO-PAYMENT EXCHANGE SYSTEM

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
Yiyao Hao
March 2005
MICRO-PAYMENT EXCHANGE SYSTEM

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

by
Yiyao Hao
March 2005

Approved by:

Dr. David Turner, Chair, Computer Science

Dr. George M. Georgiou

Dr. Tong Lai Yu

2/21/2005 Date
ABSTRACT

The micro-payment exchange market is where the buying and selling of different lightweight currencies takes place.

This project prototypes an exchange service for LCP banks and other users with large currency reserves, using inflexible exchange contracts.

This project will deliver three products, a prototype of currency exchange service with human interface for exchange owners, a lightweight currency bank that provides an administrative interface to bank managers to invoke operations on the currency exchange service, and a protocol specification for Lightweight Currency Protocol user agents to communicate with currency exchange services.
ACKNOWLEDGMENTS

I thank the faculty of Computer Science department for giving me an opportunity to pursue my M.S. in Computer Science at California State University, San Bernardino. I express my sincere appreciation to my graduate advisor, Dr. David Turner who offered me this project and directed me through this entire effort. I also thank my other committee members, Dr. George M. Georgiou and Dr. Tong Lai Yu for their valuable input.
# TABLE OF CONTENTS

ABSTRACT ........................................................................................................ iii

ACKNOWLEDGMENTS ................................................................. iv

LIST OF TABLES ................................................................. vii

LIST OF FIGURES ................................................................. viii

CHAPTER ONE: INTRODUCTION

<table>
<thead>
<tr>
<th>Purpose of the Project</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exchange Market Participants</td>
<td>2</td>
</tr>
<tr>
<td>Lightweight Currency Banks</td>
<td>2</td>
</tr>
<tr>
<td>Bank Customers</td>
<td>3</td>
</tr>
<tr>
<td>Exchange Service</td>
<td>3</td>
</tr>
<tr>
<td>Lightweight Currency Issuers</td>
<td>4</td>
</tr>
<tr>
<td>Two Types of Exchange Models</td>
<td>4</td>
</tr>
<tr>
<td>Partial Exchange Model</td>
<td>4</td>
</tr>
<tr>
<td>Total Exchange Model</td>
<td>4</td>
</tr>
<tr>
<td>Project Products</td>
<td>4</td>
</tr>
</tbody>
</table>

CHAPTER TWO: MICRO-PAYMENT EXCHANGE SYSTEM STRUCTURE .......... 7

CHAPTER THREE: PRODUCT FUNCTIONS

| Exchange Service | 9 |
| Lightweight Currency Bank | 13 |

CHAPTER FOUR: DATABASE DESIGN

| Exchange Service Database | 16 |
| Lightweight Currency Bank Database | 18 |
CHAPTER FIVE: SYSTEM IMPLEMENTATION

Communication Messages ........................................ 19
  Operations' Attributes ..................................... 19
  Response Class .............................................. 20
  Exchange Service Operations Sequence .................. 21
  Security ..................................................... 26

CHAPTER SIX: SYSTEM VALIDATION

  Exchange Service Unit Test .............................. 27
  Lightweight Currency Bank Test ......................... 27
  System Test .................................................. 30

CHAPTER SEVEN: CONCLUSION AND FUTURE DIRECTIONS

  Conclusion .................................................... 31
  Future Directions .......................................... 31

APPENDIX A: EXCHANGE WEB SERVICE DEPLOYMENT
  DESCRIPTOR FILE ........................................... 33

APPENDIX B: EXCHANGE SERVICE DATABASE FILE ............. 35

APPENDIX C: EXCHANGE SERVICE RESPONSE CLASS .......... 38

APPENDIX D: EXCHANGE SERVICE EXCHANGE CLASS .......... 40

APPENDIX E: EXCHANGE SERVICE JUNIT TEST CLASS ........ 50

APPENDIX F: A PAPER ON 2005 INTERNATIONAL
  CONFERENCE ON INFORMATION TECHNOLOGY
  CODING AND COMPUTING .................................. 55

REFERENCES ................................................................ 61
LIST OF TABLES

Table 1. Lightweight Currency Bank Functions Test Results ........................................... 27
Table 2. System Test Results ......................................................................................... 30
CHAPTER ONE
INTRODUCTION

The micro-payment exchange market is where the buying and selling of different lightweight currencies takes place.

Purpose of the Project

The purpose of this project is to investigate the infrastructure requirements for a currency exchange market for holders of currencies issued through the Lightweight Currency Protocol [3].

The Lightweight Currency Protocol was designed as a means for entities to issue currencies for the purpose of cultivating low-value business transactions between collaborating nodes in a peer-to-peer market [4]. Some possible business activities include buying and selling storage contracts, buying and selling media streaming contracts, payment-based email, buying and selling documents, etc [3][4]. In all of these markets, it is advantageous to transact in multiple currencies for the following reasons [4].
• A single currency means an entity has a monopoly, which allows the entity to extract larger than necessary profits from operating the currency.

• Multiple currencies are safer, because entities can avoid currencies that loose value through mismanagement or loss of private key.

• Multiple currencies remove a single point of failure, and thus make the markets more resilient to failure or attack.

Thus, it is advantageous for transacting entities to rely on several different currencies. However, multiple currencies result in a new problem, which is that buyers may not hold currencies required by sellers. Clearly, a currency exchange mechanism is required to enable markets to form around a multiple currency paradigm. The next section introduces this kind of currency exchange.

Exchange Market Participants

There are four types of market participants—lightweight currency banks, bank customers, exchange service and lightweight currency issuers.

Lightweight Currency Banks

They buy and sell large amounts of currencies from and to each other, and earn profits in the trading. The amount
of currency exchanges between them is large and fixed, and there is no partial exchange between banks. So the exchanges between banks are major trades in the market.

**Bank Customers**

Bank customers require other currencies in the course of doing business. Some even have their own trading account in the exchange service if their requirements are large. Other types of customers are individuals who exchange other currencies to make purchases in other lightweight currency markets, which only accept a certain type of currency. Normally, the amount of currency involved in the trading is small and some customers want partial exchange when they find good offers but not the exactly matched amount. So lightweight currency banks can handle their requests. The individuals can get exchange services from their lightweight currency banks.

**Exchange Service**

It acts as intermediaries between banks. Bank managers call it to find out where they can get the best price for currencies. Such arrangements are beneficial since it affords anonymity to the buyer/seller. Exchange service earns profit by charging a commission on the transactions they arrange and other related operations.
Lightweight Currency Issuers

Issuers act on behalf of governments, manage the currency transferring among the major currency holders, and sometimes participate in the exchange market to influence the value of their currencies.

Two Types of Exchange Models

Partial Exchange Model

This model is mainly for small amount currency exchanges for individuals or some companies. Partial exchange is allowed and exchanges are managed through lightweight currency banks. In this case, buyers are bank customers and the seller is the bank.

Total Exchange Model

This model deals with large amount of currency trading, which mostly happens between lightweight currency banks or institutions with large currency requirements. In this model, exchange action only executes between the exactly matched offers (that is, no partial exchange is allowed) and trading is managed by the exchange service.

Project Products

According to the characteristics of the two types of exchange models, we need to build two kinds of micro-
payment exchange systems to really enable markets to form around a multiple lightweight currency paradigm. This project builds an exchange system prototype for the second type of exchange model, where banks or other users of large currency reserves offer and accept inflexible exchange contracts through the exchange service. In this project, the involved entities are limited to lightweight currency banks, exchange service, and currency issuers. Other lightweight currency institutions or users are not considered in this project.

The managers of lightweight currency banks monitor the status of their currency reserves, and make deals through the exchange service to obtain currencies they lack. The basic function of the exchange system is to transfer currencies between currency holders. Issuers who maintain the accounts of currency holders manage all currencies. So issuers will be involved in the exchange system. The functions of the issuer have already been developed by Chiao [2].

This project has delivered three products:

- A prototype of currency exchange service with human interface for exchange owners.
• A lightweight currency bank that provides an administrative interface to bank managers to invoke operations on the currency exchange service.

• A protocol specification for Lightweight Currency Protocol user agents to communicate with currency exchange services.
CHAPTER TWO

MICRO-PAYMENT EXCHANGE SYSTEM

STRUCTURE

The micro-payment exchange system is a service extension to the bank managers to exchange large amounts of currencies with other banks and managed by the exchange service owners.

Figure 1. Prototype of Micro-payment Exchange System

Figure 1 illustrates the main components in this project. Bank managers log into the bank system from an administrator interface. They can view the bank currency reserves, and make decisions regarding which currencies to sell and which currencies to buy. In the bank interface, they specify the exchange operations, which the bank system translates into soap messages and delivers to the exchange service. The exchange service accesses the requests from banks and sends back responses to the banks to indicate
whether the requests are successfully processed. The owner (or manager) of the exchange service uses a browser to login to the administration interface of the exchange system. The exchange service owner can study customer transactions, manage the exchange currency reserves, and sets service fees. Whenever the operation is to transfer currencies between banks and the exchange service, the issuer will participate in the transaction.

Lightweight currency banks communicate with the exchange service by sending and receiving secured soap messages. A Java Servlet in the exchange server handles the requests from the bank managers. All entities communicate with the issuer through Lightweight Currency Protocol, which has already been built. Figure 2 shows the communication type between entities in the exchange system.

![Figure 2. Communication Diagram](image-url)
CHAPTER THREE
PRODUCT FUNCTIONS

Exchange Service

In this project, Axis is installed to publish the exchange web service. Web Service Deployment Descriptor [1] (Appendix A) is used to deploy the service. The rpc-style soap messages are sent and received between the client (bank) and the server (exchange). Axis matches each request message to a deployed exchange service operation, which is configured in the Web Service Deployment Descriptor file, and then sends back a response. The type of the return object(s) is defined in a Web Service Definition Language file [1]. Any Lightweight Currency Protocol agents who want to communicate with the exchange service can generate stub code from the exchange Web Service Definition Language file, and through the stub interface, invoke operations on remote objects as if they were local.

Because this system is a commercial service, processing fees (commissions) are charged to all operations provided by the exchange service. Due to the existence of multiple currencies, payments can be in any currency accepted by the exchange service. The exchange service reserves the right to change the amount and type of the
payment paid for each operation. When a user opens an account, the minimum deposit is required by the exchange service, the amount of which is one operation’s processing fee. The user account balance will be monitored from logging-in till logging-out. Whenever the user account balance is lower than one operation’s processing fee or the required amount in his related offers, no operation can be executed except for deposit.

In this project, there are two types of users: lightweight currency bank managers and the exchange service owners.

Lightweight currency bank managers log into their bank websites to get exchange services. First, they must open accounts in the exchange service, then post offers, cancel offers, search offers, accept offers and transfer currencies. In this system, we require the users to deposit a sufficient amount of currency, enough to perform an operation or post an offer, in the exchange service in advance. If the amount of currencies in the user account is enough for the operation fees or enough for paying his accepted offers or enough for the amount of the offer to be posted, the exchange service only needs to access the user account balance record, but no extra transfer operations are needed. This solution simplifies the system structure
and enhances its efficiency. However, we must see that in this way, the exchange service will hold a part of users' currencies. Sometimes, users may want to get their currencies back so the withdraw function must be implemented in the system.

After processing each operation mentioned above, the balance of the bank account in the exchange service should be different from the balance before, because of the commission charged by the exchange service. So the system also need to provide view account and view commission functions, which are free, for the bank managers to check any change of the account and the exchange commission.

Figure 3. Use Case Diagram For Bank Manager
Exchange service owners can log into the exchange service from a human interface to set processing fees, monitor offers and transactions, and manage the profits of the service by checking balances, or transferring profit to other LCP markets. (See Figure 4)

Figure 4. Use Case Diagram For Exchange Owner

From analysing the bank manager requirements, we see that the exchange service needs to have ten operations to meet the needs of the banks. They are open account, view commission, view account (includes view balance, view offer and cancel offer), post offer, search offer, accept offer, deposit and withdraw. When the same user accepts offers, the exchange system will check the email address of each offer's poster and send emails to notice them their offers have been bought and give a transaction report. The bank
managers send requests for these operations from their bank servers, and then get response from the exchange service. The bank servers process the response and show the response to bank managers. So there is no direct human interface for bank managers to access the exchange service.

However, for the exchange owners, they need to monitor the work of the exchange service. So the only human interface offered by the exchange service is for the exchange owners. Except that they access the same database, the exchange owner web application has no interaction with the exchange service. The issuer is also needed when the exchange owners want to transfer their business profits to other markets. (See Figure 5)

![Diagram](https://example.com/diagram.png)

Figure 5. Entities Relation Diagram For Exchange

**Lightweight Currency Bank**

A lightweight currency bank provides its managers the exchange service functions. First, the bank needs to get
the exchange service-binding file, the Web Service Definition Language file. [1] Then the bank implements the interface to get the necessary parameters, which are needed by the corresponding exchange service operation. In the detail implementation, the jsp page [9] to collect the required data for each operation and the corresponding servlet [5] is invoked to check the given data, and then the servlet either gives the error response or continues the process to connect the exchange service.

Bank managers get exchange services through sending rpc-style soap messages from their banks servers to request the exchange services. Different requests are sent from banks to the exchange service to execute operations, such as open account, post offer, cancel offer. The exchange service accesses the requests and sends responses to the banks. The bank servers parse the exchange service response and generate a jsp page [9] to show the contents of the response to the bank managers. Whenever operations are about transferring currencies between banks and the exchange service, such as, deposit and withdraw, the issuer will be invoked to send back the response after receiving the pay or verify requests from either a bank or the exchange service. Figure 6 shows the entities involved in the
exchange system depending on bank manager actions and operations between the entities.

Figure 6. Entities Relation Diagram For Bank
CHAPTER FOUR
DATABASE DESIGN

There are two main web applications in this project. They are the exchange service with a human interface for exchange owners and the bank with the interface for the bank managers. So two databases [7] are needed. One is for the exchange service and the other is for the bank.

Exchange Service Database

From the discussion of the previous chapters, we can see that bank managers and exchange owners have different relations with the exchange service. So they are regarded as two tables in the exchange service database. Table user keeps the record of bank managers, and table eadmin refers to exchange owners. Because the exchange service is a market for users to exchange different types of lightweight currencies through posting or searching the offers, offer and currency are two other important tables in the database. In order to keep track of each user's account, table transaction is needed. One user should have at least one currency; table uc is used to show the user's currency balance. Figure 7 and figure 8 show the entities and their
relations. Appendix B is exchange service database creation file.

Figure 7. Entity Relation Diagram For the Exchange Service Database
Lightweight Currency Bank Database

Lightweight currency bank database is quite simple. There is a table, admin, to check whether the user is administrator of the bank and a table named account, which keeps that bank’s exchange service username and password.
CHAPTER FIVE
SYSTEM IMPLEMENTATION

This chapter explains the details of the system. It includes the attributes of request and response messages between the lightweight currency bank and the exchange service [1], the sequence of each operation, and necessary web pages [9].

Communication Messages

The messages between the bank and the exchange service can be divided into two kinds, the operation request messages and the response massages. In this project, the return types of all the operation requests are in the same style, a response bean. The wsdl, a machine-readable file [1], shows the outline of these messages. We will see these messages in detail.

Operations’ Attributes

- Open Account (int payid, String username, String passwd, String email)
- Get Balance (String username, String passwd)
- Get Offer (String username, String passwd)
- Get Commission ()
• Post Offer (String username, String passwd, double sellamnt, String selltype, double buyamnt, String buytype, String commission)

• Cancel Offer (String username, String passwd, int[] offerids, String commission)

• Search (String username, String passwd, String buytype, String selltype, String commission)

• Accept Offer (String username, String passwd, int[] offerids, String commission)

• Deposit (String username, String passwd, int payid, String currencytype)

• Withdraw (String username, String passwd, double amount, String currencytype, String payee, String commission)

Response Class

In the response, there is a Boolean variable named success, which indicates whether the operation is performed successfully, a string variable named errorMessage and an array of string called successMessage. If the success is false, the errorMessage is set to a value to show the reason why the operation failed. Otherwise, the successMessage is set to an array type value to show the result of the operation. For example, if the request
operation is search, and the success value in response is true, the content of successMessage is an array of offers, which satisfy the search requirements. Appendix C shows the response class and appendix D shows the content of successMessage of each operation. [1][6]

Exchange Service Operations Sequence

According to the need of the exchange service users, ten operations are implemented. They are open account, get balance, get offers, get commission, post offers, cancel offers, search, accept offers, deposit, and withdraw. The following part gives a sequence diagram to each operation.

- Open Account

![Sequence Diagram For Opening Account](image)

Figure 9. Sequence Diagram For Opening Account
• Get Balance

![Sequence Diagram For Getting Balance](image_url)

Figure 10. Sequence Diagram For Getting Balance

• Get Offers

![Sequence Diagram For Getting Offers](image_url)

Figure 11. Sequence Diagram For Getting Offers
• Get Commission

Figure 12. Sequence Diagram For Getting Commission

• Post Offers

Figure 13. Sequence Diagram For Posting Offers
• Cancel Offers

Figure 14. Sequence Diagram For Cancelling Offers

• Search

Figure 15. Sequence Diagram For Searching
• Accept Offers

Figure 16. Sequence Diagram For Accepting Offers

• Deposit

Figure 17. Sequence Diagram For Deposit
• Withdraw

![Sequence Diagram For Withdraw](image)

Figure 18. Sequence Diagram For Withdraw

Security

The server machines of banks and exchange services should be configured to highly secured. IP tables packet filtering is set up to accept only necessary packets.

The exchange system will be secured through the use of SSL/TLS. All operations in the exchange system will be done over SSL, which is layered beneath the application protocols (such as HTTP) and above the connection protocol TCP/IP. SSL is used by HyperText Transmission Protocol Secure (HTTPS) access method. The currency exchange client will authenticate by presenting a username and password when it invokes a system operation.
CHAPTER SIX
SYSTEM VALIDATION

In this project, the system validation test is done in two steps. First, test each exchange service operation by using the junit test framework. After all operations can work properly, the bank web application is implemented and tested.

Exchange Service Unit Test

In this Unit test, ten operations of the exchange service are tested to ensure that they operate correctly. The unit testing code is in Appendix E.

Lightweight Currency Bank Test

This part of testing focuses on the performance of each function in the Lightweight Currency bank. Table 1 shows the test results.

Table 1. Lightweight Currency Bank Functions Test Results

<table>
<thead>
<tr>
<th>Functions</th>
<th>Tests Performed</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>• Check the page redirect correctly [8][9]</td>
<td>Pass</td>
</tr>
<tr>
<td>Open Account</td>
<td>• Check the button works properly</td>
<td>Pass</td>
</tr>
</tbody>
</table>

27
<table>
<thead>
<tr>
<th>Functions</th>
<th>Tests Performed</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Check the page redirect correctly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the issuer's remote-pay page is got and all attributes are set in the session correctly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the open account attributes are in right types</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the exchange service response is interpreted correctly</td>
<td></td>
</tr>
<tr>
<td>View Account</td>
<td>• Check the exchange response is interpreted correctly</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>• Check offerids are got</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the page redirect correctly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the exchange response is interpreted correctly</td>
<td>Pass</td>
</tr>
<tr>
<td>Cancel Offer</td>
<td>• Check the page redirect correctly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the exchange response is interpreted correctly</td>
<td>Pass</td>
</tr>
<tr>
<td>Post offer</td>
<td>• Check the page redirect correctly</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>• Check the post offer attributes are in right types</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the exchange response is interpreted correctly</td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td>• Check the page redirect correctly</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>• Check the search attributes are in right types</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the exchange response is interpreted correctly</td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td>Tests Performed</td>
<td>Results</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Accept Offer</td>
<td>• Check offerids are got</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>• Check the page redirect correctly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the exchange response is interpreted correctly</td>
<td></td>
</tr>
<tr>
<td>Deposit</td>
<td>• Check the page redirect correctly</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>• Check the issuer's remote-pay page is got and all attributes are set in the session correctly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the exchange service response is interpreted correctly</td>
<td></td>
</tr>
<tr>
<td>Withdraw</td>
<td>• Check the page redirect correctly</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>• Check the withdraw attributes are in right types</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Check the exchange service response is interpreted correctly</td>
<td></td>
</tr>
<tr>
<td>View Commission</td>
<td>• Check the exchange service response is interpreted correctly</td>
<td>Pass</td>
</tr>
<tr>
<td>Logout</td>
<td>• Verify the user remove from session after logout.</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>• Check the page redirect to proper page after logout.</td>
<td></td>
</tr>
</tbody>
</table>
System Test

System testing is the testing process that uses real data, which the system is intended to manipulate, to test the system. First all subsystem will be integrated into one system. Then test the system by using a variety of data to see the overall result.
System testing is the following steps:

Table 2. System Test Results

<table>
<thead>
<tr>
<th>System Testing</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Install the exchange system.</td>
<td>Pass</td>
</tr>
<tr>
<td>2. Start up all servers such as Tomcat server, PostgreSQL database server.</td>
<td>Pass</td>
</tr>
<tr>
<td>3. Running testing by using real data on all forms and reports.</td>
<td>Pass</td>
</tr>
</tbody>
</table>
CHAPTER SEVEN
CONCLUSION AND FUTURE DIRECTIONS

Conclusion

Micro-payment exchange system solves the problem that exits in the multiple currency markets, which is that buyers may hold currencies, but not those required by a particular sellers. Thus, the exchange system enables the multiple currency paradigm to run in the real world.

The web service design of the micro-payment exchange service gives the flexibility to the lightweight currency banks. They can implement their own exchange service interfaces according to their special requirements. It also provides interoperability between various software applications running on various platforms.

In the project practice, I learned a lot, such as, JavaServer Page, java servlet programming, web services, Postgre Database, java mail. The knowledge will be the great help for my future career or education.

Future Directions

The micro-payment exchange system is designed only for the exchange activities among the lightweight currency
banks. But the exchange service is not only limited to the banks, any lightweight currency market can get the exchange service. They implement their own exchange function interface at their local servers. What they need is to generate the necessary stub classes from wsdl document provided by the exchange service.

For the lightweight currency banks, they can implement an interface for bank customers to exchange lightweight currencies in the bank. This can be another extension function of the lightweight currency banks.
APPENDIX A

EXCHANGE WEB SERVICE

DEPLOYMENT DESCRIPTOR FILE
<deployment xmlns="http://xml.apache.org/axis(wsdd/
</service>
</deployment>
APPENDIX B

EXCHANGE SERVICE DATABASE FILE
package exchange;
import java.io.*;
import java.util.*;
import java.sql.*;
public class CreatedbSequence {
/**
 * This method creates the model database.
 * Note: this method deletes any existing data.
 */
public static void main (String argv[]) throws Exception {
    // Get database connection, and create statement object.
    Class.forName("org.postgresql.Driver");
    String jdbcUrl = "jdbc:postgresql://localhost/exchange";
    Connection connection =
        DriverManager.getConnection(jdbcUrl, "exchange", "exchange");
    if (connection == null) {
        throw new Exception("Can not get database connection.");
    }
    Statement statement = connection.createStatement();
    String sql = null;
    // Drop existing tables.
    statement.executeUpdate("drop sequence sequenceid");
    sql = "drop table uc";
    statement.executeUpdate(sql);
    sql = "drop table offers";
    statement.executeUpdate(sql);
    sql = "drop table transaction";
    statement.executeUpdate(sql);
    sql = "drop table currencies";
    statement.executeUpdate(sql);
    sql = "drop table administrator";
    statement.executeUpdate(sql);
    sql = "drop table users";
    statement.executeUpdate(sql);
    // Create tables.
    // Create users table.
    sql = "create table users (" + "userid varchar(255) primary key, " +
        "password varchar(255) not null," +
        "mailaddr varchar(30) not null)";
    statement.executeUpdate(sql);
    // Create administrators table.
    sql = "create table administrator (" + 
        "adminid varchar(255) primary key, " +
        "password varchar(255) not null," +
        "userid varchar(255) not null," +
        "FOREIGN KEY (userid) REFERENCES users {userid})";
    statement.executeUpdate(sql);
    // Create currencies table.
    sql = "create table currencies (" +
        "currtype varchar(25) primary key, " +
        "commission double precision not null," +
        "date date)";
    statement.executeUpdate(sql);
    // Create sequence for offers, transaction tables
    sql = "create sequence sequenceid start 1 maxvalue 2147483647";
    statement.executeUpdate(sql);
}
// Create user_currency table
sql = "create table uc {
" + 
  "userid varchar(255) not null," +
  "currtype varchar(25) not null," +
  "amount double precision not null," +
  "primary key (userid, currtype)," +
  "FOREIGN KEY (currtype) REFERENCES currencies (currtype)," +
  "FOREIGN KEY (userid) REFERENCES users (userid))";
statement.executeUpdate(sql);

// Create offers table.
sql = "create table offers (" + 
  "offerid int4 primary key, " +
  "sellamnt double precision not null," +
  "selltype varchar(25) not null," +
  "buyamnt double precision not null," +
  "buytype varchar(25) not null," +
  "userid varchar(255) not null," +
  "status varchar(25) not null," +
  "date date," +
  "FOREIGN KEY (selltype) REFERENCES currencies (currtype)," +
  "FOREIGN KEY (buytype) REFERENCES currencies (currtype)," +
  "FOREIGN KEY (userid) REFERENCES users (userid))";
statement.executeUpdate(sql);

// Create transaction table
sql = "create table transaction(" + 
  "transid int4 primary key, " +
  "userid varchar(255) not null, " +
  "currtype varchar(25) not null, " +
  "amount double precision not null, " +
  "credit varchar(10) not null, " +
  "date date," +
  "FOREIGN KEY (userid) REFERENCES users (userid)," +
  "FOREIGN KEY (currtype) REFERENCES currencies (currtype)," +
  "FOREIGN KEY (currencytype) REFERENCES currencies (currencytype)";
statement.executeUpdate(sql);

// Add exchange user.
sql = "insert into users (" +
  "userid, password, mailaddr) values ('exchange', 'exchange', 'none')";
statement.executeUpdate(sql);
if (statement != null) statement.close();
System.out.println("success create users table");

// Add admins user.
sql = "insert into administrator (" +
  "adminid, password, userdir) values ('eadmin', 'eadmin', 'exchange')";
statement.executeUpdate(sql);
if (statement != null) statement.close();
System.out.println("success create admin table");

// Add currencies: "ccat.ias.csusb.edu" and "lcpex.ias.csusb.edu".
sql = "insert into currencies (" +
  "currtype, commission, date) values ('ccat.ias.csusb.edu', 1, now())";
statement.executeUpdate(sql);
if (statement != null) statement.close();
System.out.println("success create currencies table");
sql = "insert into currencies (" +
  "currtype, commission, date) values ('lcpex.ias.csusb.edu', 1, now())";
statement.executeUpdate(sql);
if (statement != null) statement.close();
System.out.println("success create currencies table");

if (connection != null) connection.close();
}
APPENDIX C

EXCHANGE SERVICE RESPONSE

CLASS
package exchange;
import java.util.Vector;

public class Response {
    private boolean success;
    private String errorMessage;
    private String[] successMessage;
    private String commission;

    // Get
    public boolean getsuccess() {
        return success;
    }
    public String geterrorMessage() {
        return errorMessage;
    }
    public String[] getsuccessMessage() {
        return successMessage;
    }
    public String getcommission() {
        return commission;
    }

    // Set
    public void setSuccess(boolean s) {
        success = s;
    }
    public void seterrorMessage(String e) {
        errorMessage = e;
    }
    public void setsuccessMessage(String[] sm) {
        successMessage = sm;
    }
    public void setcommission(String c) {
        commission = c;
    }
}
APPENDIX D
EXCHANGE SERVICE EXCHANGE
CLASS
package exchange;
import java.util.Vector;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.lang.reflect.Array;

public class Exchange {
    public Exchange() {}
    public static Response openAccount(String username, String password, String mailaddr, String currtype, int payId) {
        User user = null;
        if (username == null || password == null || currtype == null) {
            resp.setSuccess(false);
            resp.seterrorMessage("Incomplete open account information.");
            return resp;
        }
        if (payId <= 0) {
            resp.setSuccess(false);
            resp.seterrorMessage("Transaction ID is invalid");
            return resp;
        }
        user = UserDB.find(username, password);
        if (user != null) {
            resp.setSuccess(false);
            resp.seterrorMessage("username exists");
            return resp;
        }
        double payAmnt = Payment.verify(currtype, payId);
        if (Payment.error != null) {
            Payment.error = null;
            if (payAmnt == 0) {
                System.out.println("Open account payment error. id: "+ username + "payid: "+ payId);
                resp.setSuccess(false);
                resp.seterrorMessage("Payment error. Please contact exchange service.");
                return resp;
            }
        }
        if (!payAmnt > 0)) {
            System.out.println("Open account payment error. id: "+ username + "payid: "+ payId);
            resp.setSuccess(false);
            resp.seterrorMessage("No deposit was found.");
            return resp;
        }
        UserDB.create(username, password, mailaddr);
        Currency curr = Currency.findcomm(currtype);
        double comm = curr.get_comm();
        int transid = TransactionDB.addNew(username, currtype, payAmnt, "deposit",
        "openAccount"+payId, comm, currtype);
        double amnt = payAmnt - comm;
        BalanceDB.increase(username, currtype, amnt);
        BalanceDB.increase(Constants.exchangeName, currtype, comm);
        String c = comm + " + currtype;
        if (amnt < 0) c = c + " Your account balance is not enough. Please deposit more currency."
        String s = transid + "username=" + payAmnt + "currtype=" + String[] sm = ["depositid userid amount currtype", s];
        resp.setSuccess(true);
        resp.setsuccessMessage(sm);
    }
}
public static Response withdraw(String username, String password, String currtype, double curramnt, String commtype, String bankName) {
    Response resp = new Response();
    if (username == null || password == null || currtype == null || bankName == null || commtype == null) {
        resp.setSuccess(false);
        resp.setErrorMessage("Incomplete information for posting offer.");
        return resp;
    }
    if (curramnt <= 0) {
        resp.setSuccess(false);
        resp.setErrorMessage("Currency amount should be greater than zero.");
        return resp;
    }

    User user = null;
    user = UserDB.find(username, password);
    if (user == null) {
        resp.setSuccess(false);
        resp.setErrorMessage("username or password is not correct.");
        return resp;
    }

    String error = User.checkBalance(username, curramnt, currtype, commtype, 1);
    System.out.println("Exchange.java: check balance error: " + error);
    if (error != null) {
        resp.setSuccess(false);
        resp.setErrorMessage(error);
        return resp;
    }

    System.out.println("Exchange.java: check balance");
    int payid = Payment.pay(Constants.exchangeName, Constants.exchangePw, bankName, currtype, curramnt);
    if (Payment.error != null) {
        Payment.error = null;
        System.out.println("Payment error: " + Payment.error);
        if (payid == 0) {
            resp.setSuccess(false);
            resp.setErrorMessage("Payment error. Please try again later.");
            return resp;
        }
    }
    if (!payid > 0) {
        resp.setSuccess(false);
        resp.setErrorMessage("Your issuer account error.");
        return resp;
    }

    System.out.println("payid from currency issuer: " + payid);
    Currency curr = Currency.findcomm(commtype);
    double comm = curr.get_comm();
    BalanceDB.decrease(username, currtype, curramnt);
    BalanceDB.decrease(username, commtype, comm);
    BalanceDB.increase(Constants.exchangeName, commtype, comm);
    int withdrawid = TransactionDB.addNewf(username, currtype, curramnt, "withdraw",
            "withdraw" + payid, comm, commtype);
    System.out.println("withdrawid: " + withdrawid);

    String c = comm + " " + commtype;
    String s = withdrawid + "" + username + "" + curramnt + "" + currtype + "" withdraw" + payid;
    String[] sm = {"withdraw userid amount currencytype descript", s};

    resp.setSuccess(true);
    resp.setsuccessMessage(sm);
    resp.setcommission(c);
    return resp;
}
public static Response deposit(String username, String password, String currtype, int payid)
{
    Response resp = new Response();
    if (username == null || password == null || currtype == null)
    {
        resp.setSuccess(false);
        resp.setErrorMessage("Incomplete information for deposit.");
        return resp;
    }
    if (payid <= 0)
    {
        resp.setSuccess(false);
        resp.setErrorMessage("Transaction ID is invalid");
        return resp;
    }

    User user = null;
    user = UserDB.find(username, password);
    if (user == null)
    {
        resp.setSuccess(false);
        resp.setErrorMessage("username or password is not correct.");
        return resp;
    }

    double payAmnt = Payment.verify(currtype, payid);
    if (Payment.error != null)
    {
        Payment.error = null;
        System.out.println("Payment error: " + Payment.error);
        if (payAmnt == 0)
        {
            System.out.println("deposit error. user: " + username + "transid: " + payid);
            resp.setSuccess(false);
            resp.setErrorMessage("Payment error. Please check your issuer account.");
            return resp;
        }
    }
    if (!payAmnt > 0)
    {
        resp.setSuccess(false);
        resp.setErrorMessage("No deposit was found.");
        return resp;
    }

    Currency curr = Currency.findcommlcurrtype(currtype);
    double comm = curr.get_comm();
    double amnt = payAmnt - comm;

    BalanceDB.increase(username, currtype, amnt);
    BalanceDB.increase(Constants.exchangeName, currtype, comm);
    int depositid = TransactionDB.addNew(username, currtype, payAmnt, "deposit",
        "deposit+payId, comm, currtype");

    String balanceError = null;
    if (amnt < 0)
    {
        balanceError = User.checkBalance(username, 0, currtype, currtype, 1);
        String c = comm + " + currtype;
        if (balanceError != null) c = c + "\n Your account balance is not enough. Please deposit more currency.";
        String s = depositid + "+username=" + username + "+ payId + " + currtype=" + currtype + " deposit" + payId;
        String[] sm = {"depositid userid amount currencytype description", s};
        resp.setSuccess(true);
        resp.setSuccessMessage(sm);
        resp.setCommission(c);
        return resp;
    }

    public static Response acceptOffer(String username, String password, String commtype, int[] offerid)
    {
        Response resp = new Response();
        if (username == null || password == null || commtype == null || offerid == null || offerid.length == 0)
        {
            resp.setSuccess(false);
            resp.setErrorMessage("Incomplete information for accepting offer.");
        }
boolean changeSuccess = OfferDB.changeStatus(offerid, "unavailable");
if (!changeSuccess)
    resp.setSuccess(false);
    resp.seterrorMessage("invalid offer id or some of your selected offers have been sold out.<br>"+"Please do another search.");
    return resp;
}
Offer o = (Offer) OfferDB.getBuy(offerid);
String buytype = o.getbuyType();
double totbuy = o.getbuyAmnt();
String selltype = o.getsellType();
double totsell = o.getsellAmnt();
System.out.println("OfferDB.getBuy "+ totbuy);
int num = offerid.length;
String error = UserDB.checkBalance(username, totbuy, buytype, commtype, num);
if (error != null) {
    OfferDB.changeStatus(offerid, "available");
    resp.setSuccess(false);
    resp.seterrorMessage(error);
    return resp;
}
Currency c = Currency.findcommm(commtype);
double comm = c.get_comm();
comm = comm * num;
BalanceDB.increase(username, commtype, comm);
BalanceDB.decrease(username, selltype, totsell);
String[] sm = new String[4];
String oid = "";
for (int i=0; i<num; i++){
    int id = offerid[i];
    Offer off = OfferDB.getOffer(id);
    String seller = off.getUserid();
    double sellamnt = off.getsellAmnt();
    double buyamnt = off.getbuyAmnt();
    String date = off.getDate();
    BalanceDB.increase(seller, buytype, buyamnt);
    int depositid = TransactionDB.addNew(seller, buytype, buyamnt, "deposit", "accepted"+id, 0.0, commtype);
    //send email to sellers
    String mail = UserDB.mailAddr(seller);
    String text = "Hello, User: "+seller+":";
    "The following offers in your account have been sold.
    "; String offer = "Offerid "+id+" sell "+sellamnt+","+selltype+" buy "+buyamnt+","+buytype+" posted date "+date+".
    "; String trans = "buyamnt" +buyamt+" has been deposited in your account.
    "; Deposit id: "+depositid+" commission : 0.0
    "; Sincerely,"
    lcpex;
    text = text + offer + trans;
    Mail.sendTo(mail, text);
}
BalanceDB.increase(username, selltype, totsell);
int depositid = TransactionDB.addNew(username, selltype, totsell, "deposit", "accept"+oid, comm, commtype);
int withdrawid = TransactionDB.addNew(username, buytype, totbuy, "withdraw", "accept"+oid, 0.0, commtype);
sm[0] = "depositid userid amount currencytype description";
sm[1] = depositid "+username" +"totsell" +"+selltype"+ acceptoffers"+oid;
sm[2] = "withdrawid userid amount currencytype description";
sm[3] = withdrawid +"+username" +"totbuy" +"+buytype"+ acceptoffers"+oid;

String commission = comm +"+"+commmtype;
resp.setSuccess(true);
resp.setsuccessMessage(sm);
resp.setcommission(commission);
return resp;
}

public static Response search(String username, String password, String selltype, String buytype, String commtype){
Response resp = new Response();
if (username == null || password == null || commtype == null || buytype == null){
resp.setSuccess(false);
resp.seterrorMessage("Incomplete information for searching offer.");
return resp;
}
User user = null;
user = UserDB.find(username, password);
if (user == null){
resp.setSuccess(false);
resp.seterrorMessage("username or password is not correct.");
return resp;
}
Balance b = BalanceDB.check_balance(username, commtype);
if (b == null){
resp.setSuccess(false);
resp.seterrorMessage("Zero balance in commission type.");
return resp;
}
Currency c = Currency.findcomm(commtype);
double comm = c.get_comm();
double amnt = b.get_amnt();
if (comm > amnt){
resp.setSuccess(false);
resp.seterrorMessage("The balance of commission type is not enough.");
return resp;
}
BalanceDB.decrease(username, commtype, comm);
BalanceDB.increase(ExchangeName, commtype, comm);
Vector offers = new Vector();
offers = OfferDB.search(selltype, buytype, username);
if (offers == null){
resp.setSuccess(false);
resp.seterrorMessage("no offer was found.<br> Charge commission "+ comm +
commmtype);
return resp;
}
int s = offers.size();
String[] msg = new String[s+1];
msg[0] = "offerid sell buy";
for (int i=0; i<s; i++){
Offer offer = (Offer) offers.elementAt(i);
String m = offer.getofferid()+"+offer.getsellAmnt()+offer.getsellType()+"+
offer.getbuyAmnt()+offer.getbuyType();
msg[i+1] = m;
}
String commission = comm +"+"+commmtype;
resp.setSuccess(true);
return resp;
resp.setSuccessMessage(msg);
resp.setcommission(commission);
return resp;
}

public static Response cancelOffer(String username, String password, String comtype, int[] offerid){
Response resp = new Response();
if (username == null || password == null || comtype == null || offerid == null || offerid.length == 0){
resp.setSuccess(false);
resp.seterrorMessage("Incomplete information for cancelling offer.");
return resp;
}
User user = null;
user = UserDB.find(username, password);
if (user == null){
resp.setSuccess(false);
resp.seterrorMessage("username or password is not correct.");
return resp;
}
Balance b = BalanceDB.check_balance(username, comtype);
if (b == null){
resp.setSuccess(false);
resp.seterrorMessage("Zero balance in commission type.");
return resp;
}
Currency c = Currency.findcommt(comtype);
double comm = c.get_comm();
int l = offerid.length;
System.out.println("offerid length:"+l+" offerid[0]:");
double com = comm * l;
double amnt = b.get_amnt();
if (com > amnt){
resp.setSuccess(false);
resp.seterrorMessage("The balance of commission is not enough to cancel "+l+" offers.");
return resp;
}
String[] sm = new String[l+1];
sm[0] = "depositid userid amount currencytype description";
double realComm = 0;
for (int i=0; i<l; i++){
int oid = offerid[i];
Offer offer = (Offer) OfferDB.getOffer(oid);
if (offer != null){
String status = offer.getStatus();
if (status.equals("available")){
double sellamnt = offer.getsellAmnt();
String selltype = offer.getsellType();
OfferDB.delete(oid);
BalanceDB.increase(username, selltype, sellamnt);
int depositid = TransactionDB.addNew(username, selltype, sellamnt, "deposit", "cancelOffer"+oid, com, comtype);
sm[i+1] = depositid="+username+"+sellamnt+"+selltype+"cancelOffer"+oid;
realComm += comm;
}
}
if(realComm == 0){
resp.setSuccess(false);
resp.seterrorMessage("Cannot cancel your selected offers. &lt;br&gt; Please check your offers' id or status.");
return resp;
}
BalanceDB.decrease(username, commtype, realComm);
BalanceDB.increase(Constants.exchangeName, commtype, realComm);
String commission = realComm + " + commtype + " <br> " + "Please check the offers' status is available <br> " + "if the number of cancelled offers is different from that of you selected."
resp.setSuccess(true);
resp.setsuccessMessage(sm);
resp.setcommission(commission);
return resp;
}

public static Response getCommission()
{
Response resp = new Response();
Vector comm = new Vector();
try{
    comm = Currency.getAll();
} catch(Exception e){
    resp.setSuccess(false);
    resp.seterrorMessage("Exchange system database error. Please try check commission again later.");
    return resp;
}
int s = comm.size();
System.out.println("get commission size: "+ s);
String[] sm = new String[s+1];
for (int i=0; i<s; i++){
    Currency c = (Currency) comm.elementAt(i);
    String l = c.get_type()+" + c.get_comm();
    sm[i+1] = l;
}
resp.setsuccessMessage(sm);
resp.setSuccess(true);
resp.setcommission(" ");
return resp;
}

public static Response getBalance(String username, String password)
{
Response resp = new Response();
if (username == null || password == null){
    resp.setSuccess(false);
    resp.seterrorMessage("Incomplete parameters.");
    return resp;
}
User user = null;
user = UserDB.find(username, password);
if (user == null){
    resp.setSuccess(false);
    resp.seterrorMessage("username or password is not correct.");
    return resp;
}
Vector balances = new Vector();
try{
    balances = (Vector) BalanceDB.check_balance(username);
} catch(Exception e){
    resp.setSuccess(false);
    resp.seterrorMessage("Exchange system database error. Please try check balance later.");
    return resp;
}
if (balances == null){
    System.out.println("Get balance error. userid: "+username);
    resp.setSuccess(false);
    resp.seterrorMessage("No balance was found.");
    return resp;
}
int s = balances.size();
String[] sm = new String[s+1];
System.out.println("getOffer size: "+ sm.length);
sm[0] = "currencytype amount";
for (int i=0; i<s; i++){
    Balance b = balances.elementAt(i);
    String t = b.get_type()+"\"+b.get_amnt();
    sm[i+1] = t;
}
resp.setSuccess(true);
resp.setsuccessMessage(sm);
return resp;
}

public static Response getOffer(String username, String password){
    Response resp = new Response();
    if (username == null || password == null){
        resp.setSuccess(false);
        resp.setErrorMessage("Incomplete parameters.");
        return resp;
    }

    User user = UserDB.find(username, password);
    if (user == null){
        resp.setSuccess(false);
        resp.setErrorMessage("username or password is not correct.");
        return resp;
    }

    Vector offers = new Vector();
    try{
        offers = OfferDB.find(username);
    } catch (Exception e){
        resp.setSuccess(false);
        resp.setErrorMessage("Exchange system error. Please try again later.");
        return resp;
    }

    resp.setSuccess(true);
    resp.setcommission(" ");
    if (offers == null){
        String[] sm = {"offerid sell buy status date"};
        resp.setsuccessMessage(sm);
        return resp;
    }

    int s = offers.size();
    String[] sm = new String[s+1];
    System.out.println("getOffer size: "+ sm.length);
    sm[0] = "offerid sell buy status date";
    for (int i=0; i<s; i++){
        Offer o = (Offer) offers.elementAt(i);
        String t = o.getOfferId()+"\"+o.getSellAmnt()+o.getSellType()
            +"\"+o.getBuyAmnt()+o.getBuyType()+"\"+o.getStatus()+"\"+o.getDate();
        sm[i+1] = t;
    }
    resp.setsuccessMessage(sm);
    return resp;
}

public static Response postOffer(String username, String password, double buyamnt, String buytype, double sellamnt, String selltype, String commtype){
    Response resp = new Response();
    if (username == null || password == null || buytype == null || selltype == null
        || commtype == null){
        resp.setSuccess(false);
        resp.setErrorMessage("Incomplete information for posting offer.");
        return resp;
    }

    User user = UserDB.find(username, password);
    if (user == null){
        resp.setSuccess(false);
        resp.setErrorMessage("username or password is not correct.");
        return resp;
    }

    Double buyPrice = Double.parseDouble(buyamnt);
    Double sellPrice = Double.parseDouble(sellamnt);
    String buyCurrency = buytype;
    String sellCurrency = selltype;
    String commissionType = commtype;

    resp.setSuccess(true);
    resp.setcommission(commissionType);
    resp.setsuccessMessage("success");
    return resp;
}
if (buyamnt <= 0 || sellamnt <= 0) {
    resp.setSuccess(false);
    resp.setErrorMessage("Currency amount should be greater than zero.");
    return resp;
}

User user = null;
user = UserDB.find(username, password);
if (user == null) {
    resp.setSuccess(false);
    resp.setErrorMessage("username or password is not correct.");
    return resp;
}
String error = User.checkBalance(username, sellamnt, selltype, commtype, 1);
if (error != null) {
    resp.setSuccess(false);
    resp.setErrorMessage(error);
    return resp;
}

double comm = 0;
int offerid = 0;
int withdrawid = 0;
try {
    Currency c = Currency.findcommlcommtype);
    comm = c.get_comm();
    BalanceDB.decrease(username, commtype, comm);
    BalanceDB.decrease(username, selltype, sellamnt);
    BalanceDB.increase(ExchangeName, commtype, comm);
    offerid = OfferDB.post(sellamnt, selltype, buyamnt, buytype, username, "available");
    withdrawid = TransactionDB.addNew(username, selltype, sellamnt, "withdraw",
"postOffer"+offerid, comm, commtype);
} catch (Exception e) {
    resp.setSuccess(false);
    resp.setErrorMessage("Exchange system database error. Please try post offer
lister.");
    return resp;
}

String commission = comm + " " + commtype;
String s = withdrawid + "+username+" +sellamnt+" +selltype+
postOffer"+offerid;
String[] sm = {"withdraw userid amount currencytype description", s};
resp.setSuccess(true);
resp.setSuccessMessage(sm);
resp.setCommission(commission);
return resp;
APPENDIX E

EXCHANGE SERVICE JUNIT TEST

CLASS
package exchange;
import java.util.Vector;
import junit.framework.TestCase;
import junit.framework.Test;
import junit.framework.TestSuite;

public class TestRunner extends junit.framework.TestCase {
    public TestRunner(java.lang.String name) {
        super(name);
    }

    public static Test suite() {
        TestSuite suite = new TestSuite();
        suite.addTest(new TestRunner("testOpenAccount"));
        suite.addTest(new TestRunner("testDeposit"));
        suite.addTest(new TestRunner("testPostOffer"));
        suite.addTest(new TestRunner("testWithdraw"));
        suite.addTest(new TestRunner("testSearchOffer"));
        suite.addTest(new TestRunner("testCancelOffer"));
        return suite;
    }

    protected void setUp() throws Exception {
    }

    protected void tearDown() throws Exception {
        static final String bankName = "bank1";
        static final String bankPw = "bank1";
        static final String exchangeName = "exchange";
        static final String exchangePw = "exchange";

        public void testOpenAccount() {
            // Open an account for bank X.
            String name = "x";
            String pw = "x";
            String mailaddr = "yhaocsci.csusb.edu";
            String currtype = "lcpex.ias.csusb.edu";
            double curramnt = 200;

            try{
                currency.CurrencySoapBindingStub bindingc = (currency.CurrencySoapBindingStub)
                        new currency.CurrencyServiceLocator().getCurrency();
                assertNotNull("Bindingc is null.", bindingc);
                bindingc.setTimeout(60000);

                int paymentId = bindingc.pay(bankName, bankPw, exchangeName, curramnt);
                assertTrue("paymentId not positive", paymentId > 0);
                assertEquals(3, paymentId);

                exchange.ExchangeSoapBindingStub bindinge;
                bindinge = (exchange.ExchangeSoapBindingStub)
                        new exchange.ExchangeServiceLocator().getExchange();
                assertNotNull("bindinge is null", bindinge);
                // Time out after a minute
                bindinge.setTimeout(60000);
                System.out.println("Test OpenAccount");

                Response resp =
                        (Response) bindinge.openAccount(name, pw, mailaddr, currtype, paymentId);
        }
    }
}

51
assertNotNull(resp);
assertTrue(resp.isSuccess());
String comm = resp.getCommission();
assertEquals("1.0 lcpex.ias.csusb.edu", comm);
String[] sm = resp.getSuccessMessage();
assertEquals("1 x 200.0 lcpex.ias.csusb.edu", sm[1]);
}
}

public void testDeposit()
{
    // bank X deposits currency to exchange.
    String name = "x";
    String pw = "x";
    String currtype = "lcpex.ias.csusb.edu";
    double curramnt = 100;
    try{
        currency.CurrencySoapBindingStub bindingc = (currency.CurrencySoapBindingStub)
                new currency.CurrencyServiceLocator().getCurrency();
        assertNotNull("Bindingc is null.", bindingc);
        bindingc.setTimeout(60000); // one minute
        int paymentld = bindingc.pay(bankName,
                bankPw, exchangeName, currant);
        assertTrue("paymentld not positive", paymentld > 0);
        assertEquals(4, paymentld);
        exchange.ExchangeSoapBindingStub bindinge;
        bindinge = (exchange.ExchangeSoapBindingStub)
                new exchange.ExchangeServiceLocator().getExchange();
        assertNotNull("bindinge is null", bindinge);
        // Time out after a minute
        bindinge.setTimeout(60000);
        System.out.println("Test Deposit");
        Response resp =
                (Response) bindinge.deposit(name, pw, currtype, paymentld);
        assertNotNull(resp);
        assertTrue(resp.isSuccess());
        String comm = resp.getCommission();
        assertEquals("1.0 lcpex.ias.csusb.edu", comm);
        String[] sm = resp.getSuccessMessage();
        assertEquals("2 x 100.0 lcpex.ias.csusb.edu depositd", sm[1]);
    }catch(Exception e){
        throw new RuntimeException(e);
    }
}

public void testPostOffer()
{
    // bank X deposits currency to exchange.
    String name = "x";
    String pw = "x";
    String commtype = "lcpex.ias.csusb.edu";
    String selltype = "lcpex.ias.csusb.edu";
    double sellamnt = 10;
    String buytype = "ccat.ias.csusb.edu";
    double buynamt = 5;
    try{
        exchange.ExchangeSoapBindingStub bindinge;
        bindinge = (exchange.ExchangeSoapBindingStub)
                new exchange.ExchangeServiceLocator().getExchange();

assertNotNull("bindinge is null", bindinge);
// Time-out after a minute
bindinge.setTimeout(60000);
System.out.println("Test PostOffer");

Response resp =
(Response) bindinge.postOffer(name, pw, buyamnt, buytype, sellamnt,
selltype, commtype);

assertNotNull(resp);
assertTrue(resp.isSuccess());
String comm = resp.getCommission();
String[] sm = resp.getSuccessMessage();
assertEquals("1.0 lcpex.ias.csusb.edu postOffer3", sm[l]);
}
}

public void testWithdraw()
{
// bank X withdraws currency from exchange.
String name = "x";
String pw = "x";
String currtype = "lcpex.ias.csusb.edu";
double curramnt = 5;
String commtype = "lcpex.ias.csusb.edu";
int payId;
double payAmnt;
try{
exchange.ExchangeSoapBindingStub bindinge;
bindinge = (exchange.ExchangeSoapBindingStub)
new exchange.ExchangeServiceLocator().getExchange();
// Time-out after a minute
bindinge.setTimeout(60000);
System.out.println("Test Withdraw");

Response resp =
(Response) bindinge.withdraw(name, pw, currtype, curramnt, commtype,
bankName);
assertNotNull(resp);
assertTrue(resp.isSuccess());
String comm = resp.getCommission();
assertEquals("1.0 lcpex.ias.csusb.edu", comm);
String[] sm = resp.getSuccessMessage();
assertEquals("5 x 5.0 lcpex.ias.csusb.edu withdraw5", sm[l]);
currency.CurrencySoapBindingStub bindingc = new
currency.CurrencySoapBindingStub();
currency.CurrencyServiceLocator currencyServiceLocator = new
currency.CurrencyServiceLocator();
bindingc = (currency.CurrencySoapBindingStub)
currencyServiceLocator.getCurrency();
bindingc.setTimeout(60000);
payAmnt = bindingc.verify(bankName, bankPw, 5);
assertEquals(curramnt, payAmnt, 0.0);
}
catch(Exception e){
throw new RuntimeException(e);
}
}

public void testSearchOffer()
{
// bank X deposits currency to exchange.
String name = "x";
String pw = "x";
String selltype = lcpex.ias.csusb.edu";
String buytype = "ccat.ias.csusb.edu";
String commtype = "lcpex.ias.csusb.edu";

try{
    exchange.ExchangeSoapBindingStub bindings;
    bindings = (exchange.ExchangeSoapBindingStub)
        new exchange.ExchangeServiceLocator().getExchange();
    assertNotNull("bindinge is null", bindings);
    // Time out after a minute
    bindings.setTimeout(60000);
    System.out.println("Test SearchOffer");
    bindings = (exchange.ExchangeSoapBindingStub)
        new exchange.ExchangeServiceLocator().getExchange();
    assertNotNull("bindinge is null", bindings);
    // Time out after a minute
    bindings.setTimeout(60000);
    System.out.println("Test Search");

    Response resp =
        (Response) bindings.searchOffer(name, pw, selltype, buytype, commtype);
    assertNotNull(resp);
    assertTrue(resp.isSuccess());
    String comm = resp.getCommission();
    assertEquals("1.0 lcpex.ias.csusb.edu", comm);
APPENDIX F

A PAPER ON 2005 INTERNATIONAL
CONFERENCE ON INFORMATION
TECHNOLOGY CODING AND
COMPUTING
An Exchange Protocol for Alternative Currencies

Yiyao Hao, Daniel M. Havey and David A. Turner
Department of Computer Science
California State University, San Bernardino
San Bernardino, CA 92407
(hyy92507@yahoo.com, dhavey@yahoo.com, dturner@csusb.edu)

Abstract

Alternative currencies are currencies issued by individuals or entities other than national governments for the purpose of improving the economic well being of individuals by cultivating new trading relationships. We present a currency exchange protocol for use in alternative currency markets, such as those based on the Lightweight Currency Protocol, Local Exchange Transaction Systems, and Time Dollar Systems. In order to further enable the benefits derived from the use of alternative currencies, we propose the establishment of an exchange protocol that enables users to swap currencies. In this paper, we define the protocol we developed for this purpose, and describe the systems we prototyped to demonstrate the practicability of the currency exchange protocol.

Keywords: micro-currency, alternative currency, community currency, e-commerce.

1. Introduction

We refer to currencies based on the Lightweight Currency Protocol (LCP) as LCP currencies [1]. LCP is an account-based micro-payment system that operates over the Internet. Any person or organization is free to issue or publish an LCP currency. Issuers of LCP currencies imbue value into their currencies by backing them with a commodity, so that holders may redeem units of an issuer's currency for these commodities. Some of the commodities that have been considered as backing for LCP currencies are storage/bandwidth [2], spam reduced email [10, 11, 12], pdf document stores, video streaming, and content distribution.

The motivation for this research is to increase the value of using alternative currencies such as Local Exchange Time Systems (LETS), Time Dollars, and LCP currencies [3, 4, 5, 16, 17, 18]. To accomplish this purpose we have developed an Internet currency exchange protocol based on secure SOAP messaging. Our approach was to develop a prototype currency exchange server and prototype currency exchange client to test and refine our protocol. Throughout this paper we refer to the currency exchange as the X-Server. To test the server, we developed a currency exchange agent for use by managers of banks holding LCP currencies. The idea was to better understand how a bank would manage large reserves of multiple alternative currencies.

Currently there are more than 600 alternative currencies being used in Japan [3]. Although Japan has the largest number of alternative currencies in use other areas such as the United States, Europe, South America, and many other areas [16, 17, 18] have alternative currencies of their own. The use of these alternative currencies has been and continues to be strong because of their ability to cultivate new trading relationships. To further the reach of these community building currencies we propose the adoption of a standard currency exchange protocol that can be implemented by competing currency exchange service providers. The benefit of having an open and non-proprietary currency exchange protocol is that the market mechanism will help keep exchange fees to a minimum.

![Figure 1: Communications Diagram for Currency Exchange Protocol](image)

We used SOAP based web services in the development of our protocol because the availability of web service tools and frameworks should facilitate the adoption of the protocol among alternative currencies.

This paper is organized as follows. In section 2, we describe the exchange protocol and its implementation in the X-Server. In section 3, we present a typical example of the exchange protocol being used, and we explain our
experiments with our prototype X-Server and prototype LCP bank system. Section 4 describes future work and extensions to accommodate other currencies.

2. X Server Operations

Figure 1 illustrates the main components in this prototype. Bank managers log into the bank's system from an administrator interface. They can view the bank's currency reserves, and make decisions regarding which currencies to sell and which currencies to buy. In the bank's interface, they specify the exchange operations, which the bank system translates into SOAP messages and delivers to the exchange service. The exchange service processes the requests from banks and sends back responses to the banks to indicate whether the requests are successfully processed. The manager of the exchange service uses a browser to login to the administration interface of the exchange system. The exchange service manager can study customer transactions, manage the exchange's currency reserves, and set service fees. Whenever the operation is to transfer currencies between banks and the exchange service, the currency issuers will participate in the transaction.

The Internet Currency Exchange Protocol (ICEP) is a secure request/response protocol. All messages are delivered over HTTPS. The exchange server authenticates to the client using a certificate signed by a trusted authority. The client authenticates to the exchange certificate by including a user name and password within each request message except for the get commission operation.

For all operations invoked on it, the X server returns an XML response message containing the following attributes: a success code, an error message when the operation fails, a success message when the operation succeeds, and transaction fee details. The success code is a boolean value indicating the success or failure of an operation. The error message is a string indicating the reason if any for failure of an operation. The success message is an array of strings containing information related to the successful completion of the operation. The transaction fee details indicate the currency issuer and amount.

As can be seen from the use case diagram in Figure 2, the X server has nine operations. The two operations available to a user who wishes to make an exchange offer are: post offer, and cancel offer. The two operations available to a user who wishes to find and accept an offer are: search offer, and accept offer. Before users can invoke any of the above operations they need to establish an account and deposit funds. To do this there are two operations available: open account, and deposit. When a user wishes to access funds in her account there is one operation available: withdraw. Finally there are two operations available to view a user's balance and the X-Server's schedule of fees: get balance, and get commission. In the following subsections, we detail each operation in the currency exchange protocol.

<table>
<thead>
<tr>
<th>Bank Manager</th>
<th>Bank Server</th>
<th>X-Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>post</td>
<td>post</td>
<td>post</td>
</tr>
<tr>
<td>username</td>
<td>user name</td>
<td>public</td>
</tr>
<tr>
<td>password</td>
<td>password</td>
<td>password</td>
</tr>
<tr>
<td>commission</td>
<td>commission</td>
<td>commission</td>
</tr>
<tr>
<td>withdrawal</td>
<td>withdraw</td>
<td>withdraw</td>
</tr>
<tr>
<td>success</td>
<td>success</td>
<td>success</td>
</tr>
<tr>
<td>message</td>
<td>message</td>
<td>message</td>
</tr>
<tr>
<td>fee</td>
<td>fee</td>
<td>fee</td>
</tr>
<tr>
<td>amount</td>
<td>amount</td>
<td>amount</td>
</tr>
</tbody>
</table>

Figure 3: The Post Offer Operation Sequence Diagram

The Post Offer Operation

As illustrated in Figure 3, a bank manager offers to exchange some amount of a currency by sending a post offer request message to the X-Server. The post offer request message contains an attribute indicating the number of units in a currency that the seller is offering to sell. The buy attribute specifies the number of units in a
currency that the seller will accept to make the deal. The commission attribute specifies the number of units in a currency that the seller agrees to pay the exchange for posting her offer. The reason for this attribute is to ensure that the exchange client is aware of the current schedule of fees. If the fee schedule has changed then the X-Server will reject the operation with an appropriate error message, which allows the client's agent to either update its record of exchange fees and/or inform the user of the change.

When the X-Server processes and accepts the post offer operation, it subtracts the transaction fee from the user's balance and makes the offer publicly available. The X-Server will return a response message indicating the success or failure of the post offer operation.

Search Offers Operation

In the search offers operation, the exchange client includes attributes within the request message similar to the post offer operation: sell, buy, and commission. The X-Server performs conditional select operations on its offer database to generate a list of offers that match the buyer search criteria. The X-server then returns this list of offers to the buyer in the response message.

Accept Offer Operation

The accept offer operation takes a list of ids for those offers the buyer wishes to accept. The X-Server performs the operation by adjusting its database to reflect the transfer of funds between the accounts of the buyer and sellers. There is no commission charged for accepting offers. The X-server then returns a response message to the buyer.

Cancel Offer Operation

The cancel offer operation specifies an offer to cancel by supplying the offer id. After authentication, the X-server cancels the offer, if it has not yet been accepted by some buyer. The X-server will then return a response message. Note that accepted offers are no longer pending, and thus the cancel offer operation will return an error response if invoked on an already accepted offer.

Operations that Interact with Currency Issuers

The deposit, withdraw, and open account operations require a two-step interaction with a currency issuer. The X-Server prototype currently supports LCP currency issuers. LCP currency issuers also use SOAP over HTTPS in a request/response protocol. When an LCP currency user invokes a transfer funds request on a currency issuer the issuer returns a payment token if the operation succeeds. The sender of funds then gives the payment id to the recipient of the funds. The recipient of the funds then provides the payment id to the issuer in a verify operation in order to verify the payment. All three operations of deposit, withdraw, and open account require this two-step process with the currency issuer.

![Figure 4 Deposit Operation Sequence Diagram](image)

Deposit Operation

As shown in Figure 4, the exchange client deposits currency units in its exchange account by transferring those units to the X-server's account with the currency issuer. The X-server then verifies the deposit by invoking a verify operation on the currency issuer. The X-Server then deducts a commission, if any, and adds the balance to the exchange client's account.

Withdraw Operation

The exchange client withdraws funds for its use outside of the exchange by invoking the withdraw operation in which it specifies the number of units in a currency that it wishes to withdraw. When the X-Server receives a withdraw request, it checks that the client's balance in the given currency is sufficient. If the balance is sufficient, the X-Server then invokes a transfer funds operation on the currency issuer transferring funds from its account with the issuer to the client's account with the issuer. When this operation completes the X-Server receives a payment id from the issuer, which it returns to the client. The client then verifies the operation with the currency issuer.
Open Account Operation

Figure 5 shows the sequence of interactions for an open account operation. To open an account the exchange client first transfers funds to the X-Server, which results in a payment token. The client then invokes an open account operation in which it specifies the payment token for the funds it just transferred to the X-Server. When the X-Server receives the open account request, it verifies the client’s payment using the payment token. If the verification operation is successful, the X-Server establishes a balance for the client in the currency it has just received. The client’s initial balance will equal the amount she transferred minus the commission charged by the X-Server for opening the account.

Figure 5: Open Account Operation

View Account Operation

In order to view information about its account or the schedule of fees, the exchange client uses two operations: get balance and get commission. These operations return the balance and the schedule of fees, respectively. The get commission operation is the only operation that does not require user authentication, and is the only operation that can be invoked over a non-secure channel. This is to allow prospective clients to view the schedule of fees before opening an account.

3. The X-Server Exchange Market

In this section, we present a typical usage of the X-Server in an LCP currency exchange. Suppose Alice issues a currency that is backed by a storage/bandwidth service. That means holders of Alice’s currency may redeem their currency for storage/bandwidth services provided by Alice. Suppose also that Bob issues a currency that is redeemable for viewing video content.

Suppose that Claire uses payment based email as described in [11], and has received units of Bob’s currency by collecting email delivery fees.

Alice desires to view one of Bob’s videos, and therefore she needs Bob dollars. Claire desires to use Alice’s storage service, and therefore she needs Alice dollars.

Alice opens an account at an X-Server by depositing 1000 units of her own currency, and posts an offer to exchange 100 Alice dollars for 200 Bob dollars. Claire opens an account at the same X-Server by depositing 1000 Bob dollars. Claire locates Alice’s offer through a search operation, which she accepts by invoking an accept offer operation. The X-Server then reduces Alice’s balance of Alice dollars by 101 units (assuming a 1 unit commission), and increases Alice’s balance of Bob dollars by 200. Similarly, the X-Server reduces Claire’s balance of Bob dollars by 202 units (assuming a 2 unit commission in Bob dollars), and increases Claire’s balance of Alice dollars by 100 units. Alice and Claire are now able to withdraw the funds that they need to purchase their desired commodities.

In the above described scenario, the X-Server facilitated a trading relationship that would have been otherwise difficult or impossible. A large amount of such potential trading relationships exist across alternative currencies. Such exchange services are needed for alternative currencies to gain wider acceptance.

Another scenario that illustrates the use of our exchange protocol involves a community of Local Exchange Transaction System (LETS) users. In this scenario, Alice works at her local public library and gets paid in LETS dollars. Alice uses the LETS dollars she earned to pay Bob to mow her lawn. Bob in turn uses the X-Server to exchange the LETS dollars that he earned from Alice with Claire for Claire LCP dollars. Bob uses his Claire dollars to buy network resources and Claire uses her LETS dollars to pay her library fines.

4. Conclusion

The main goal of the X-Server project is to strengthen the viability of alternative currencies by providing a means for users of alternative currencies to reach wider markets by exchange their currencies. This is a need cited by Kytojoki in [9]. Currently, the X-Server fulfills this
role for LCP currencies. By extending
the X-Server to operate with other
currency systems, these exchanges of
Currencies could happen anywhere in the
emerging world of alternative
currencies.

References

A Lightweight Currency Paradigm for the
PPP Resource Market. Seventh
International Conference on Electronic
Commerce Research, Dallas, TX, Jun
2004.
[2] David Turner, Daniel Havey and John
Ewart. Allocating Resources in Storage
Cooperatives with Pseudo Currencies. In
proceedings of the International
Conference on Computer Science and its
Currencies in Japan Today: History,
Originality and Relevance.
International Journal of Community
Currency Research. Vol.8, pp.1-23
[4] Seyfang, Gill. Tackling social
exclusion with community currencies:
learning from LETS to Time Banks.
International Journal of Community
Currency Research. 2002: Volume 6
Currencies for Fair and Stable
International Exchange Rate.
International Journal of Community
Currency Research. 2001: Volume 5
Payword and micromint: Two simple Micropayment
schemes. In Security Protocols
Workshop, pages 69-87, 1996.
[7] B. Yang and H. Garcia-Mollina. PPAY:
Micropayments for Peer-to-Peer Systems.
In Proceedings of the 10th ACM
Conference on Computer and
Communications Security (CCS), oct 2003
[8] C. Schmidt & R. Muller, "undated".
"A Framework for Micropayment
Evaluation," Sonderforschungsbereich
373 1998-66, Humboldt Universitaet
Berlin.
Micropayments - Requirements and
Solutions. Tik-110.501 Seminar on
[10] David Turner and NI Deng. Payment-
Based Email. 6th International
Conference on Software Engineering,
Artificial Intelligence, Networking,
and Parallel/Distributed Computing
(SNPD2004), Beijing, China, Jun/Jul
2004.
Controlling Spam through Lightweight
Currency. In proceedings of the Hawaii
International Conference on Computer
Sciences, Honolulu, HI, Jan 2004.
[12] Cashette, inc. Payment based email
User Attitudes towards Electronic
Payment Systems. Proceedings of Joint
AFIHM-BCS Conference on Human-Computer
[14] Bakos, Y. and Brynjolfsson, E.
Aggregation and Disaggregation of
Information Goods: Implications for
Bundling, Site Licensing and
Micropayment Systems, in Proceedings of
Internet Publishing and Beyond: The
Economics of Digital Information and
Intellectual Property. D. Hurley, B.
Kahn, and H. Varian, eds. MIT Press,
Cambridge, Massachusetts (August 2000).
Markets: The Emerging Role of
Electronic Marketplaces on the
Internet, Communications of the ACM,
35-42.
Tauschring Hannover (TTH): Experiences
of a German LETS and the relevance of
theoretical reflections. International
Journal of Community Currency Research,
[17] Jeffrey Jacob, Merlin Brinkerhoff,
Emily Jovic and Gerald Wheatley. HOUR
Town - Paul Glover and the Genesis and
Evolution of ithaca HOURS.
International Journal of Community
[18] Stephen DeMeulenaere. Reinventing
the Market: Alternative Currencies and
Community Development in Argentina.
International Journal of Community
REFERENCES


6. Java API. http://java.sun.com/j2se/1.5.0/docs/api

