Design and Implementation of an
Electronic Commerce System using Java RMI

By
Min Liang (mliang@vt.edu)
Zhenping Liu (zhliu@vt.edu)

ABSTRACT

We implement an Electronic Commerce System. The primary objectives of this project are to understand deeply many concepts of the distributed operating system. Our Electronic Commerce System (ECS) extends the basic required function by the default project. The system allows transaction among multiple clients, multiple stores, and multiple banks. Any client can view and purchase items from any store. When a client places a purchase order to a store, the store will check whether the order is valid, then it will connect the corresponding bank to verify the customer’s account. The bank site, before authorizing the expenditure, will seek confirmation from the client directly. If everything is valid, bank will charge money and inform the store, the store will commit the order and tell the client the result. The interaction between a client and a store is asynchronous, so ECS allows a client to conduct business with several stores simultaneously. Friendly Graphic User Interfaces (GUIs) are used to facilitate the usage of the system.

Some algorithms and technologies are employed in building the system. (1) Java Remote Method Invocation (RMI) mechanism. (2) Monitor that enforces reader priority policy is used to synchronize the access to the database in store sites. Same monitor policy is also used for the access synchronization of the bank account database. On client sites, we use a simplified monitor policy for the access synchronization of the checkout history log file. (3) When several clients queries all item information of a store site, to permit concurrent and consistent reading of all items and at the same time to avoid deadlock, we use an algorithm similar to a solution for problem set 1 of HW2 (4) We use two phases (reserve phase and commit phase) when handling the orders at store sites. (5) Java multi-thread technology is used to ensure asynchronous interaction between a client and a store. (6) We use a private key cryptography method (Conversation key) to ensure security of important data.(7) On a client site we use unique sequence number as timestamp to record the transaction with store. It is used to avoid replay of message and ensure ID authentication. (8) We use a class that implements serializable to contain the important data (various authentication packages), so the safety of data is strengthened.

PROJECT DESCRIPTION

BRIEF INTRODUCTION:

Our Electronic Commerce System (ECS) greatly extends the basic requirement of the default project. The system consists of multiple clients, multiple stores, and multiple banks.

The system architecture is shown as Fig.1. There’re three kinds of servers in the system: Client Server, Bank Server and Store Server.
A client can query all items of a store and request a store to checkout an order. Query transaction is simple. So here we only give the checkout transaction steps:

1. The client assigns a unique sequence number (customerSeqno) to the current transaction, then constructs an authentication package, which we named as customer_package_1 (refer to Fig.2). The package includes customerSeqno, storeName, currentClientURL, userAccountID, encryAuthenID1 (encrypted by some method). We use an object of class “AuthenPackage” that implements serializable to contain this authentication package, so the safety of data is strengthened. Note all various authentication packages below are all realized using class “AuthenPackage”.

2. The client sends the checkout request to the store with orderList, userAccountID, BankLocation and customer_package_1.

3. The store checks if all ordered items are available. If not, it informs the client the result. If yes, it reserves them for that customer, then continue (4).

4. The store constructs its authentication package named store_package (refer to Fig.4). The package includes userAccountID, storeName, storePasswd, totalAmount. It then sends store_package and customer_package_1 to the corresponding bank for a transaction request.

5. Bank will decrypt store_package and customer_package_1. Before authorizing the expenditure, the bank first checks the validity of the storePasswd, totalAmount, storeName, userAccountID. Then it will check the account balance. If all these are OK, it will seek confirmation directly from the client.

6. At the confirmation request from the bank, the client will construct a different authentication package named customer_package_2 (refer to Fig.3) which includes customerSeqno, storeName,
userAccountID, userAccountPasswd, totalAmount, encryAuthenID2 (encrypted by a method different from encryAuthenID1), and send it directly to the bank.

(7) When the bank receives the package customer_package_2, it will decrypt the package, then compares it with information in customer_package_1 to check the validness of customerSeqno, storeName, userAccountID, userAccountPasswd, totalAmount. It will also decrypt the encryAuthenID2 and check whether it is same as decrypted encryAuthenID1. If everything is valid, bank will charge money.

(8) The bank returns the execution result of the transaction to the store.

(9) The store will commit or rollback the order according to the result from the bank.

(10) The store returns the transaction result back to the client.

**DESIGN OF CLIENT SITES**

The client, store and bank sites all act as both client and server during a checkout. **Java RMI mechanism** is employed to establish interactions among client site, store site, and bank site. A client site can view all items of a store and send an order to any store for checking out acting as a client. It can also answer the confirmation request from a bank acting as a server.

Friendly **Graphic user interfaces (GUI)** are used to help the customer buy things and facilitate the interaction between clients and stores, such as submitting query request, displaying query result, adding or deleting or emptying items in shopping cart, placing orders, inputting credit card number and password, and what’s more, prompting messages are used to display the transaction result and failure causes if necessary.

**Asynchronous interaction**, query and order, between a client and a store are supported. That is, the client should not block waiting for the store to reply to its requests to view or purchase items. So ECS allows a client to conduct business with several stores simultaneously. **Java multi-thread** is used to implement the asynchronous transactions. Whenever the client submit a request to store, it generates a new thread to process the transaction in order that the main window can continue to respond to user’s other requests.

To ensure **security** of important data and to avoid replay of message, we take the following technologies:

(1) We use a class “AuthenPackage” that implements **serializable** to contain the important data (such as userAccountID, userAccountPasswd, totalAmount, customerSeqno, storeName, encryAuthenID1, encryAuthenID2). To serialize an object means to convert its state to a byte stream for transmitting on network and revert it back into a copy of the object at the destination site. So the safety of data transmitting is strengthened. Only the destination site can have a complete copy of the object and understand it.

(2) The client send two AuthenPackage packages “customer_package_1” and “customer_package_2” (refer to “BRIEF INTRODUCTION”) by different way to the bank. “customer_package_1” is to bank by way of store. Store can’t understand it and just forwards it to the bank. “customer_package_2” is direct to bank. So the bank will decrypt the packages and check the validness of the client, user account, total amount, as well as the validness of the store. These can be used to avoid the replay of message and ensure security of important data.
(3) The client randomly generates an integer number, then encrypt it into “encryAuthenID1” by using cryptography method 1 and “encryAuthenID2” by using cryptography method 2. We conclude these two encryAuthenID in AuthenPackage packages “customer_package_1” and “customer_package_2” respectively. So bank will decrypt and check them. These can be used to avoid the replay of message ensure security of important data.

(4) On client site we use unique sequence number as timestamp to record each transaction with store. This sequence number will be contained in AuthenPackage packages as timestamp to bank. It is used to avoid the replay of message and ensure ID authentication.

We also need solve the concurrent writing problem on the client site. When a client check out an order to a store, it will add a record about the current transaction information in its “checkout history log” file; When bank request direct confirmation from the client, the client will search the log file for corresponding transaction information. If find, it will delete corresponding record from the log. We use a simplified (only have synchronized writing access) monitor policy for the access synchronization of the checkout history log. We only allow one write operation at a time.

**DESIGN OF STORE SITES**

A store site also acts as a client and a server. It can perform queries, processing orders using a two-phase commit protocol.

Reader-priority Monitor policy is used to synchronize the access to the same data item of database in store sites. The whole database is represented as a vector “ItemList” which consists of many data items. Each data item is an object of class “OneItem”. “OneItem” is the class to present one item of the whole storehouse. When several processes need to read or write on data items, we use Reader-priority Monitor policy to solve the problem. We allow multiple reading or one writing at the same time.

When several clients queries all item information of a store site, at the same time some processes are reading or writing on data items, to permit concurrent and consistent reading of all items and to avoid deadlock, we use an algorithm similar to a solution for problem set 1 of HW2. Let a process query all data items by the order of item(1), item(2), …, item(n-1), item(n). When it read item(i), it firsts need to acquire the lock on the item(i) based on reader-priority monitor policy, then read it, then begin to read item(i+1), …. When it read all items, it then free all locks also by the order of item(1), item(2), …, item(n).

To avoid the inconsistency of the system, we divide the whole commit process into two phases.

Phase 1: On receiving an order request from a client, the store first checks the availability of all items. If any one of order items isn’t available, the store aborts the transaction and informs the
client. If all items are all available currently, the store reserves them and contacts the bank to request money transfer.

Phase 2: The bank checks the validness of the client, the store, customer account and account balance. Then it seeks direct confirmation from the client. If all are valid, the bank site charge the money and informs the store site to commit the order. Otherwise, the bank site informs the store site to abort the order. If the bank returns success, the store will commit transaction. Otherwise the store will rollback and put the reserved items back to the store.

This method also has some disadvantage that the reserved items are not available to other customers until rollback.

To ensure security of important data and to avoid replay of message, we take the similar technologies as the client site:

?? The store use object of class “AuthenPackage” that implements serializable to contain the important data. The store sends its authentication package named store_package (refer to fig. 4) along with customer_package_1 to the corresponding bank for a transaction request.

<table>
<thead>
<tr>
<th>userAccountID</th>
<th>storeName</th>
<th>storePasswd</th>
<th>totalAmount</th>
</tr>
</thead>
</table>

Fig. 4 AuthenPackage package “store_package”

DESIGN OF BANK SITE

When the bank accepts the money transfer request from a store, it is as a server; when it sends confirmation request to the client, it is as client. Since we've described many design factors about bank in the above paragraphs, such as checking validness of package, seeking direct confirmation from client and decryption. We only give the design factors that not included above.

Reader-priority Monitor policy Similar to the store site is used to synchronize the access to the same account item of database in bank sites.

SUMMARY

Through this project, we profound our knowledge of synchronization, Java monitor, deadlock, cryptography and security. We also gain some practical experiences of using Java's RMI mechanism, two-phase commit protocol, and Java’s GUI programming techniques. This project also gave us an insight to some challenges involved in designing and building a distributed system.

However, Some improvement can be added into the system to enhance its functions:

1. Some rollback and recovery work can be done in case that the Banks or the Stores are crashed suddenly. Log files and consistent checkpoints can be used to do this.

2. All databases are local now. Future work can be done to the distributed database though this is complex job.