Simple E-Commerce System
Implementation

Course Project
CS 5204 – Advanced Operating Systems
Fall 2000

By
Joy Mukherjee  (jmukherj@vt.edu, Student ID# 990-22-5624)
Tejas Patel  (tepatel@vt.edu, Student ID # 900-13-7540)
Abstract

Objective

The purpose of the project was to build a simple E-Commerce application. The system was designed and built taking into consideration Remote Method Invocation Fundamentals. The technologies used in the development of the system were JDK 1.3 and operating system used was Windows 2000 Professional. The objective of the exercise was to get a feel of how processes are executed in a distributed environment and what synchronizations measures one should take for achieving distributed processing. Also other objective was to learn how to use sockets, explicitly to connect to remote machines.

Problem

The project aims at developing a distributed system where four different machines communicate with one another to perform simple e-commerce transaction. Out of the four machines, one acts as a client who selects the store to transact and then orders the items. The store machine then asks for the account number and password of the client and sends it to the bank, running on the other machine for verification. The Bank verifies the client account and balance and asks the client about the validity of transaction. If client confirms then bank debits the account and store updates the inventory. However the requirement was, when client placed the order, his behavior should be asynchronous, meaning that he should not be blocked and should be able to transact with other store it he wishes.

Architecture of the System

In order to implement the above-mentioned system, we have a three level client server architecture as explained below.

The basic architecture consists of customer acting as a client and selects one of the stores to transact. Store in this case, running on other machines acts as Server for the client. A customer can at a time have multiple transactions with the same store or can transact with other store. Since we could have multiple transactions running at a time, we spawn a new thread for each transaction of the client. Once the customer places his order, he needs to provide his account number and password to the store.

The store on receiving the account information, contacts the bank, running on the other machine for the verification of the customer and also sends the total cost incurred during the transaction. This forms the second level of client server architecture, where bank acts as a server and store is the client. Bank then verifies account information as well as the balance in the bank. If everything seems OK then bank contacts the client again to seek his confirmation.
This then forms the third level of Client-Server Architecture, where bank acts as a client and customer acts as a Server. When bank shows customer the summary of purchases he made from the store, client either agrees or disagrees. Depending on the customer reply, bank then informs the store to complete the transaction or cancel it.

**NOTE:** We have assumed that a valid customer interacting with the store already has the account with the bank. There is no facility where a new customer can create his account with the bank through our system.

Database has being implemented using simple text files. Bank contains text files named on the account number of the client and contains information about password and balance. While store maintains a database about items in its inventory. Each file is named on the item name and contains information about quantity, buying price and selling price.
Control Flow Diagram of Client Transacting with Store1

Customer

Thread 1

Thread 2

Store 1

Thread

Bank

Store 2

(1)

(2)

(3)

(4)

(5)

(6)
Steps:

1) Client wants to transact with Store1 so spawns a thread Thread1 and places order.
2) Store in turn spawns a thread, which communicates with both bank and the client.
3) Bank in turn again spans a thread, which asks the client about the authenticity of the order placed and Store.
4) Client the spawns another thread to communicate with the bank (Thread 2).
5) Based on the reply by client, bank responds to the store.
6) Based on the reply received by the bank, store completes or terminates the transaction with the client.

Three Way Client Server Architecture:

<table>
<thead>
<tr>
<th>Step No</th>
<th>Client:</th>
<th>Server:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Customer</td>
<td>Store</td>
</tr>
<tr>
<td>2</td>
<td>Store</td>
<td>Bank</td>
</tr>
<tr>
<td>3</td>
<td>Bank</td>
<td>Customer</td>
</tr>
<tr>
<td>4</td>
<td>Customer</td>
<td>Bank</td>
</tr>
<tr>
<td>5</td>
<td>Bank</td>
<td>Store</td>
</tr>
<tr>
<td>6</td>
<td>Store</td>
<td>Customer</td>
</tr>
</tbody>
</table>

Features:

1) Remote Method Invocation:

We have not explicitly used the RMI interface provided by JAVA, but have implemented the RMI at the basic socket level. Each client contacts the server by opening a socket connection and then invoking the method at the server by passing some token. Server on his behalf would execute appropriate method based on the token value. For example Bank would pass a token “COMMIT” on its output stream to the store acting as a server during step 5. Based on this token value store would then update the inventory. Thus RMI has been implicitly implemented.

2) Synchronization:

Since it is a distributed system, there must be the element of synchronization when same client or different clients try to access the same store database. Our system provides synchronized access to database. Also bank accounts are synchronized meaning that if the same client is communicating with different
stores at the same time, whichever transactions gets completed second gets the updated view of the clients account.

3) Presence of Two Phase Commit protocol:

In our system the database updating is done using two -phase commit protocol. There are two phases

1) First phase is a checking phase where clients account number and password are being passed to the Bank for validation.
2) Second phase is that of confirmation. If a bank validates the account number and password and confirms with the client, it asks store to update the database. If validation phase fails then, transaction is terminated there itself.

Also during the second phase, along with updating, confirmation is also asked from the client. If client refuses to accept the transaction then, the transaction is terminated there itself.

4) Support for Multiple Clients:

Our systems support the existence of multiple clients. That means multiple clients can be present and they can transact with any number of stores (here 2) simultaneously and multiple clients can transact with the same store at the same time.

5) Easy to use User Interface:

Our system has a pretty nice client side user interface. This interface allows client to select his items from a list of items dynamically generated by consulting the database of the selected store. It also allows client to edit his order before placing the order. This is something easy compared to normal command prompt interface.

Things that are desirable in the existing system:

1) Presence of locking at item level rather than at database level.

At present only one process can access the database for updates at a time. Thus even if two updates are to be made on two different items, they cannot be executed simultaneously. This is however somewhat inefficient. We would desire synchronization at the item level so that two processes trying to access different items in the inventory should be able to execute simultaneously.
2) **Synchronization at the account level in the bank.**

This is the same as in the previous case – the store. Here also two accesses to the same account need to be mutually excluded rather than two accesses to the bank as a whole which is the present situation.

3) **Presence of some form of cryptography, for transmitting secure information such as password on the communication channel.**

**Screen shot of client side user interface**
Summary:

It was a rich experience to work on this project. It certainly helped to give form and structure to many of the theoretical concepts leaned from the study books. Also it helped us in getting a better grasp on the Java language and helped us to find out the power of Java as distributed operating system. It also gave us a good insight into ways and means of an e-commerce system and made us familiar with the problems normally encountered in distributed environment.