SOEN 423

Assignment 2

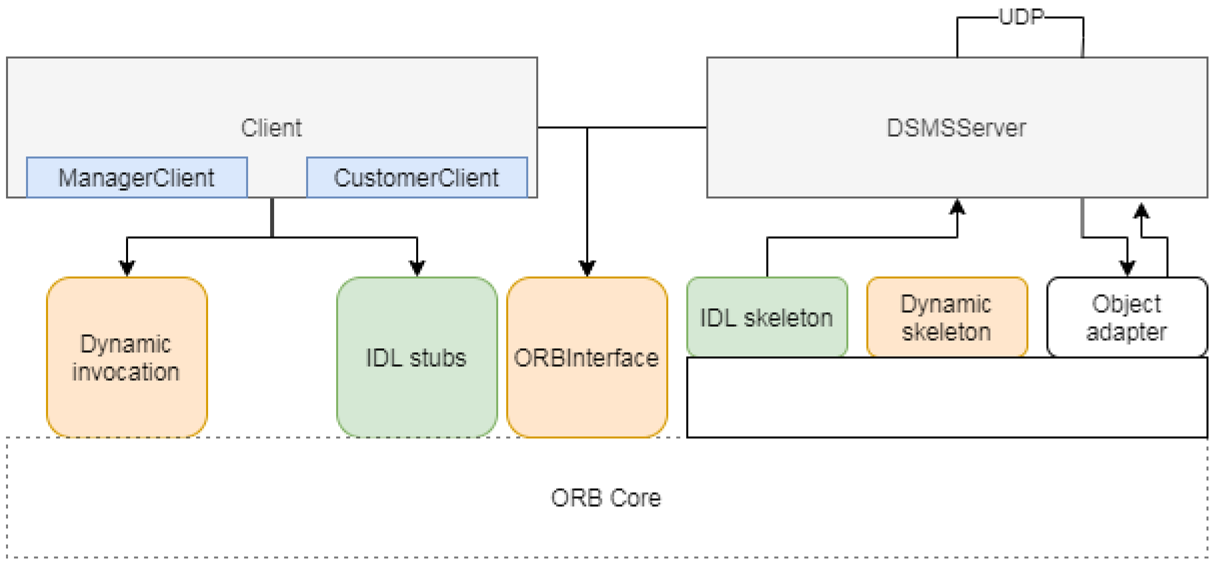
By: Saffia Niro

#40054733

October 23, 2020

1.0 Overview

This assignment was primarily an extension of the first assignment, and thus the methods, data structures, etc. are mostly the same. An `exchangeItem` method was added and instead of RMI, CORBA was used. Interserver communication was still done over UDP/IP.



2.0 CORBA

An interface definition language file (.idl) was created with the following methods:

Manager Operations:

```
boolean addItem(in string managerID, in string itemID, in string
itemName, in short quantity, in double price);
boolean removeItem(in string managerID, in string itemID, in short
quantity);
string listItemAvailability(in string managerID);
```

Customer Operations:

```
double purchaseItem(in string customerID, in string itemID, in Date
dateOfPurchase);
```

```
string findItem(in string customerID, in string itemName);
boolean returnItem(in string customerID, in string itemID, in Date
dateOfReturn);
boolean exchangeItem(in string customerID, in string newItemID, in
string oldItemID);
```

Other:

```
oneway void shutdown();
```

Further, I created a Date `struct` that contains 3 attributes, all of type `short`: day, month, and year.

The interface definition was then compiled to generate the other required files. In terms of client and server implementation, one class for the server was used, `DSMSServer`, and two classes for the client were used, `DSMSManagerClient` and `DSMSCustomerClient`.

3.0 Application

The same driver that was used for the first assignment was used for this assignment. Two additional options were added for customers, who can now exchange items atomically as long as the criteria for a valid purchase of a new item and a valid return of an old item are met.

Additionally, there is an option to test the concurrency and atomicity of the application. Selecting this option will start two threads for two users, who will attempt to simultaneously exchange the same item. To ensure that only one user can ever modify the status of an item, synchronization and mutual exclusion are used. These are used throughout the program to ensure that there are no race conditions, so as to ensure the integrity of the stores' inventories.

4.0 Example test scenario

Procedure:

1. Log in as a QC manager and list items
2. Log in as a user from BC
3. Purchase an item that is out of stock in QC and go on the waiting list
4. Log in as a user from ON
5. Purchase the same item that was out of stock in QC and go on the waiting list
6. Log in as a QC manager and add 1 instance of the out of stock item
7. Log in as the BC user who automatically bought the item and return it
8. Log in as an ON user and find said item
9. Log in as a manager and remove said item
10. Log in as a QC user and try to purchase an item that is too expensive from ON
11. Log in as a user from ON and purchase an available item from QC
12. Exchange that item for an item from BC
13. Log in as a user from BC and purchase the same item from BC
14. Perform the concurrency test with the former ON user and the BC user
15. Exit program

5.0 Final remarks

The most challenging part of this assignment was setting up CORBA, as I encountered some issues with switching to Java 8 in order to be able to use CORBA. However, once this was set up, I found the implementation fairly simple, as the general concept of distributed systems remained the same between RMI and CORBA, despite differences in the actual implementation.

However, I did experience some difficulties handling purchase and return dates, most notably with regards to the correct storing, passing, and verifying of dates.

As for improvements to my program, I would like to add more robust error-checking, particularly for edge-cases.