Problem 2: Distributed Order Management Service

Due: 11:59PM, Sunday, March 16, 2025 (same for both on- and off-campus students)

I. Overview

This course project is intended to familiarize you with the tools and with the process of building and deploying a simple distributed CORBA-based application. The objective of this machine problem is to use CORBA to build a Distributed Order Management System for restaurants where a restaurant manager can view current orders, and customers can view menus and place orders. Its output information provides data-driven insights that help in operating and managing a restaurant efficiently.
Note: We expect each student will design and implement this programming assignment individually. Please schedule a time with the TA to show the project demo, between March 17 and March 21, 2025.
Detailed requirements for this project are outlined below.

II. Project Requirements

For this project, you are to develop a simple distributed order management server for restaurants. It can list menus and receive customer orders and can display current orders to restaurant managers. The order management server will keep track of orders placed by each customer and accordingly update the newly generated orders. At any point, the customer may decide to join the system, view menus, and place an order. In which case, the manager can refresh the page and receive all placed orders. More specifically, two Java applications are required, a server and a client. Any number of clients may connect to the server. The system must meet the following general requirements:

- There are two user roles: Customer and Manager. A separate User Interface is required for each role. Only one active role at a time needs to be supported. At any given time, there may be at most one manager and any number of customers active for a single restaurant. *However, the server must not lose information even if clients are repeatedly stopped and restarted.*
- The Client may be built as a single application that supports both Customer and Manager interfaces, or two separate applications may be built, one for a Customer and another for a Manager.
- Errors, server exceptions, and invalid user input shall produce reasonably informative error displays to the user.
- Note that for this project, a single IDL interface is sufficient. This interface should contain the exception declarations, attribute declarations, and all of the operations interfaces that specify the functionality described for the Customer and the Manager.
- It is acceptable for the client to be replicated on additional machines. Client and Server shall not operate out of the same directory. The Client environment shall not contain unnecessary Server components.
- The Customer and Manager roles should provide the functionality specified by the following use case scenarios (illustrated in Figure 1):
  - **view_menu**: When a customer joins the system, he can initiate a request to get the restaurant's menu. The menu contains the dishes provided by the restaurant and the corresponding prices. In order to simplify the system, we assume that the restaurant only provides two kinds of food: fried chicken and cola, the prices are 5 dollars and 1 dollar, respectively.
  - **place_order**: Any customer, other than the manager, may place an order on the system by providing a user name and quantity of each type of food. If the quantity is valid (e.g., not smaller than zero), the system should receive the order and return a confirmation to the user. The confirmation could be a simple notification string (e.g.,"The order is received") to let the customer knows the order status.
  - **check_order_status**: After the customer receives the system's confirmation, he can check his balance by supplying his user name. For the system simplicity, we assume each user can only place one order. So, the system can use the user name as the key to identify the customer's order and return it to the customer. The returned status should contain the customer's name, the quantity of each type of food ordered by the customer, and the total price of this order. If the customer provides a user name that has not placed an order on the system, the returned information should be an error notification.
  - **view_current_orders**: For the manager, by calling this function, he should know all the restaurant orders. The output of this function should include all orders placed. For each order, the output information should include the user name, the quantity of each type of food ordered by the customer, and the order's total price. To simplify the system, after a new order is submitted, we do not require this function to update all order information automatically. But it should be ensured that after the new order is submitted, the newly generated order should be involved if the function is called again.

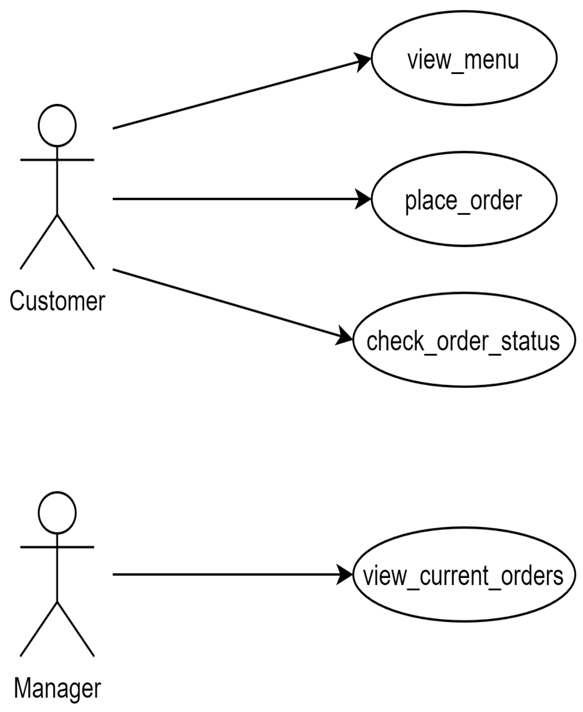

Fig. 1. Case Scenarios for Customer and Manager's Functionality

III. What to Hand In

A. Design

Before you start hacking away, plot down a design document. The result should be a system-level design document, which you hand in along with the source code. Do not get carried away with it, but make sure it convinces the reader that you know how to attack the problem. The source code and reports should be submitted through Canvas.

You will have to show the demo of this project virtually in Webex between March 17 - 21, 2025. TA Md Shafiqul Islam (shafiqul@iastate.edu) will allocate the time slot and discuss in detail in Canvas. Note that the demo requirement is for both on-campus and off-campus students. For students who do not show a demo, the project will be graded manually by the TA on Canvas.

B. Measurements

In order to compare the efficiency of your implementation, you will perform some measurements. I will leave it to you to decide what measurements should be done.