School of Computer Science, McGill University

COMP-512 Distributed Systems, Fall 2024

Programming Assignment 1: Distributing an Application*

Due date: Report is due September 30, Demo dates follow during the same week

Note that both team members are expected to work equally on this project and you should find a solution together (at least discuss and explain to each other how you implemented the individual parts). In the report that you have to provide (see Question 3 below), you will have to indicate clearly who did what for this project and you also have to indicate how you collaborated.

Your task in this programming assignment is to develop a component-based distributed information system experimenting with different communication paradigms.

You can achieve a total of 100 Points on this assignment. That is the number of points reflects the weight of each of the deliverables within this assignment.

Question 1: Implementation (75 Points)

This programming assignment is a Travel Reservation system, where customers can reserve flights, cars and rooms for their vacation. You can find more information about the client interface and explore the functionality in the `clientUserGuide.pdf`.

The functionality of the management system is very simple, but it will make the programming easier. We make the following simplifications:

1. There are only 3 types of reservable items (key denoted in *italics*):
   (a) **Flights:** *flight number*, price, and the number of seats
   (b) **Cars:** *location*, price, and the amount available
   (c) **Rooms:** *location*, price, and the amount available

   Note that since location is used as a key, there is only one type of car/room (and only one price) for a particular location.
2. Adding an duplicate item updates the price (if greater than zero) for all already existing items, and increases the count.
3. Each customer maintains a list of their reserved items.

*This assignment is an adaption of the project of CSE 593 of the University of Washington.

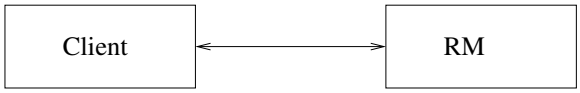
4. Querying a customer returns the list of reserved items, as well as the total cost to the customer.

5. Deleting a customer also cancels all reservations they performed.

Beginning with the starter code provided on myCourses, this programming assignment requires you to distribute the simple application in two ways: RMI and sockets.

RMI Distribution

The provided sample code already implements a single-client, single-server system using RMI for communication. On the client side, an interactive console takes text-based commands (see `clientUserGuide.pdf`), parses the input and sends a message over RMI to the `ResourceManager`. The `ResourceManager` receives the command and modifies its local state accordingly. Note that in this implementation, a single `ResourceManager` is responsible for resources of all types.



The first task in this milestone involves distributing the system *without* modifying the client.

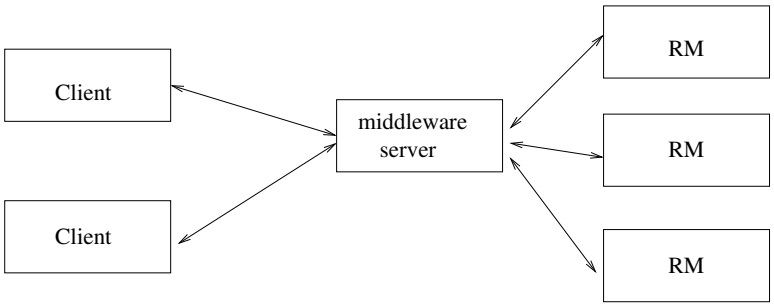
1. Introduce a new intermediary server called the `Middleware` that sits between the client and the `ResourceManagers`. On startup it should be given the list of `ResourceManagers` and implements the same interface as the `ResourceManager`.

2. Create `ResourceManagers` for each type of resource (one for each of flights, cars, and rooms). For simplicity, you can keep only one implementation providing the entire interface – the `Middleware` will only call the flight-methods on the flight-`ResourceManager`.

3. Decide how to handle customers – either with an additional server, through replication at the `ResourceManagers`, or at the `Middleware` (which then becomes a type of `ResourceManager`).

4. Add an additional function `bundle` to the `Middleware` which reserves a set of flights, and possibly a car and/or room at the final destination.

All clients send requests to the `Middleware` server which are then distributed appropriately.



TCP Sockets

Note: You might be able to experiment with possible solutions for this sub-task in parallel with the RMI distribution but at the end the final TCP implementation should also have a distributed architecture with a middleware.

Note also: The below description is short, but expect it to be more time consuming than RMI.

Re-implement your system using TCP sockets for communication between all layers instead of RMI. The client can remain blocking (i.e. it sends a request and waits for the reply before sending the next request). However, the `Middleware` must not block when it is waiting for the `ResourceManagers` to execute a request. That is, when the `Middleware` receives a request from the client, it forwards it to the corresponding `ResourceManager` and continues accepting client requests. When it receives a response from the `ResourceManager`, it messages the appropriate client. Similarly, the `ResourceManagers` should be able to handle several requests concurrently.

Question 2: Report (15 Points)

Project Description (10 Points)

Write a short (3-4 pages) report detailing your architecture and design for both RMI and TCP. The TCP section will likely dominate the report, and should explain your strategy for message passing as well as concurrency. Briefly mention your implementation choices for customers and bundling.

One page of the report should list the tests you conducted to test whether your implementations are correct. Think of specific sequences of updates and queries that might cause trouble and show that your system handles them correctly.

Collaboration (5 Points)

As last section, your report must contain a detailed description of the contributions of each of the group members. That is, for each of the tasks listed above (RMI distribution, TCP sockets, custom functionality and report), indicate how much and what each of the team members has contributed. Furthermore, describe your collaboration efforts. For that, you

can, e.g., include the dates and durations of each of the meetings, and/or the number of emails / Slack discussion entries / messages that you had in order to come up with the overall solution. Each team member is expected to contribute significantly to the project work. Please note that that things like 'report writing', 'manual testing', 'discussed ideas' by themselves are not considered a significant contribution. If we notice a pattern of certain members not contributing consistently, they will receive a lower grade letter at the end of the course.

Demos (10 Points)

To grade your implementation (i.e., the 75 points above), we will use live demonstrations with the TA where you show your running system.

Furthermore, the way you present and demonstrate your system will be evaluated by a further 10 Points. It is recommended that your demonstration include a very short (no more than a couple of slides) presentation with highlights of your system and your implementation strategies. The presentation should be distinct slides (or a well-thought narrative) and not "scrolling through the report". You might also think of how to show particular test cases.

The demonstrations of all groups will take place over two or three days within week 6. A sign-up sheet will be put in place so that you can reserve a time-slot. Show up early and have your system running, using at least 5 different machines for your client and servers. The TA will ask you to perform certain tasks that show the working of the system. Expect also questions about your architecture, code design, implementation details, etc. Ensure the person who is doing the demo is efficient with Unix command prompt. Practice a bit on your own before showing up for the demo. Questions can be directed at anyone and not just the person who is executing the commands. **ALL team members must be present for the demo. Absentees will not receive the grade.**

All code is due on MyCourses by the time of your demo.