COMP-512 Report: Project 1

Group 14: Jin Dong, Shiquan Zhang

1: RMI Distribution

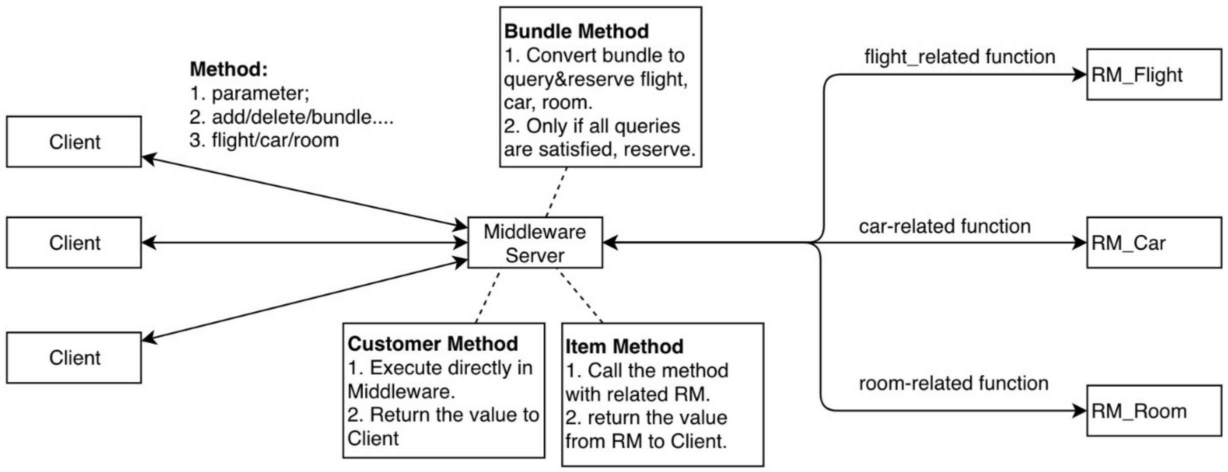
Method:
1. parameter;
2. add/delete/bundle....
3. flight/car/room
Client
Client
Client
Bundle Method
1. Convert bundle to query&reserve flight, car, room.
2. Only if all queries are satisfied, reserve.
Middleware Server
Customer Method
1. Execute directly in Middleware.
2. Return the value to Client
Item Method
1. Call the method with related RM.
2. return the value from RM to Client.
flight_related function
RM_Flight
car-related function
RM_Car
room-related function
RM_Room

Figure 1

Figure 1 shows the overall architecture of the system.

For clients, they only communicate with middleware as with server before. For RMs, they also only communicate with middleware as with clients before. For the **Middleware** server, we regard it as a client of three RM servers and a server of clients. Thus, middleware first gets references of three RM servers and then wait callings from clients. Middleware is also the Customer-RM server, so it will handle customer-related functions. All other functional methods of middleware are just re-calling corresponding methods of related-RMs, and returning what RMs return to clients.

2. TCP Sockets

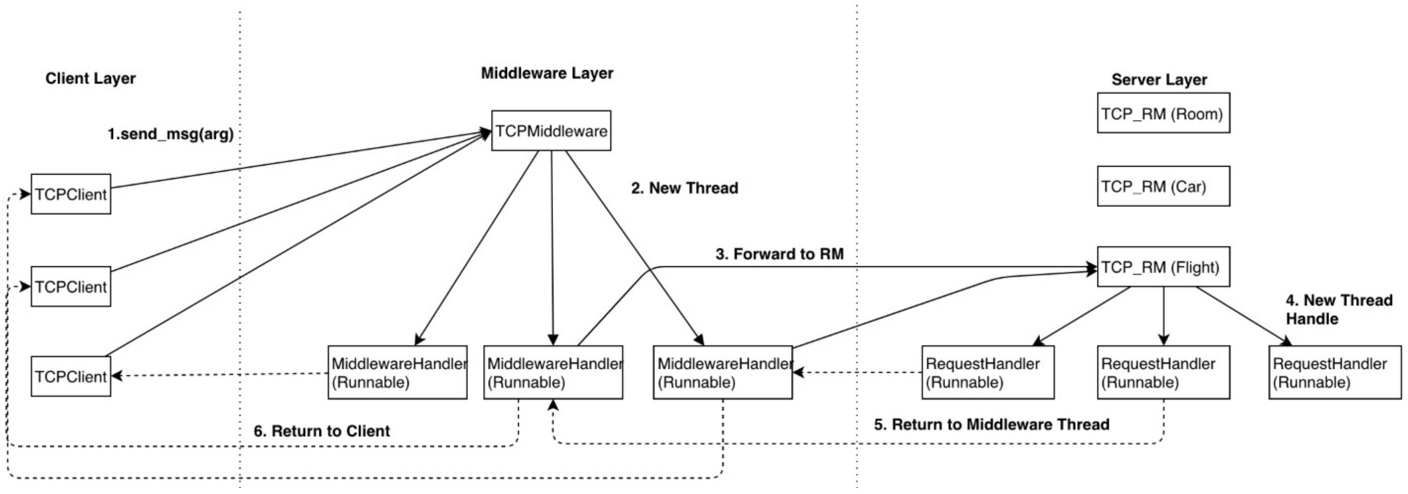

Figure 2

Figure 2 shows the multithreading strategy of three parts in the system. Switching from RMI to TCP sockets, we use multithreading in Middleware and RM servers to listen and handle the requests concurrently.

At the beginning, the Middleware and the RM servers start and keep listening on the assigned port. Then, when a client receives a command, it creates a socket connecting to the Middleware. When the Middleware receives a connection from a client, it creates a new thread to handle the connection between this client. After parsing and partitioning the requests from the client, the Middleware distributes the requests to three RM servers. It builds a new socket to connect to one of the RM servers and send its requests to the RM server. Then it stands by, waiting for the result of its requests. Next, in the RM server, when it receives a connection from the Middleware, it also creates a new thread to handle this socket. After receiving and handling the requests, it writes the result to the buffer of the socket. And then this thread ends. Receiving the result from RM servers, the Middleware then relays the result to the socket which connecting to the client and closes the socket connecting to the RM server. The handling thread in of the Middleware also ends after finishing sending result to client. Finally, the client receives the result and closes the socket connection with the Middleware.

The multithreading design allows the server to handle multiple requests concurrently. The Middleware can receive many connections from different clients, as one RM server can receive multiple socket connections from the Middleware. Therefore, they can handle different requests in different threads, which achieves the concurrency of the servers.

For the message passing, we define the message passing in TCP socket as a vector of string *Vector<String>*, just the same as the parameters reading from command line in the client. Besides, we use the enumerate class *Client.Command* to differentiate different types of requests. More specifically, both the Middleware and the RM servers parse the request command. The Middleware relays the original request command to the corresponding RM server after parsing the request and making some operation on the *customer* RM. As for the result returned from the servers, both RM servers and the Middleware send a *String* of *"True"* or *"False"* for the Boolean results or a *String* of number for the number querying results.

3. Implementation of customers and bundling
We keep the same design style for the two functions in RMI and TCP.

For customer handling, we use Middleware to handle customer-related method. So, when the middleware receives a customer-related requests, including *Add/Delete/QueryCustomer, ReserveItem* and *bundle*, it will call the corresponding functions of itself if communicating using RMI. For other types of requests, the Middleware will directly call the corresponding functions in three RM servers. In the TCP mode, the RMI calls are replaced by TCP socket connections. Thus, the function calls related to the customers are changed to TCP socket message sending from client to the Middleware and handled in the Middleware. Besides, the function calls related to *reservable* items in the customer functions are also changed to the TCP socket message sending from the Middleware to the RM servers.

For bundling function, we split it into several query functions and reserve functions in the Middleware. Then Middleware will send RMs query callings first. If all the items in the bundle are satisfied, Middleware then will send RM servers reserve callings and return the result to clients.

4. Custom Functionality

We implement the *analytics* custom functionality. To use this functionality, user input the command "*Analyitcs,<item>,<quantity>*". The first parameter is Flight, Car or Room and the second parameter is the threshold number of the quantity. It will return all the items of type "*<item>*" which quantity is lower than the threshold. The Middleware parses the command and forwards it to the corresponding RM server. Then the RM server searches the hashmap for all items that the amount is lower than the quantity threshold. At last the result is returned as a *Vector<String>* to the client.