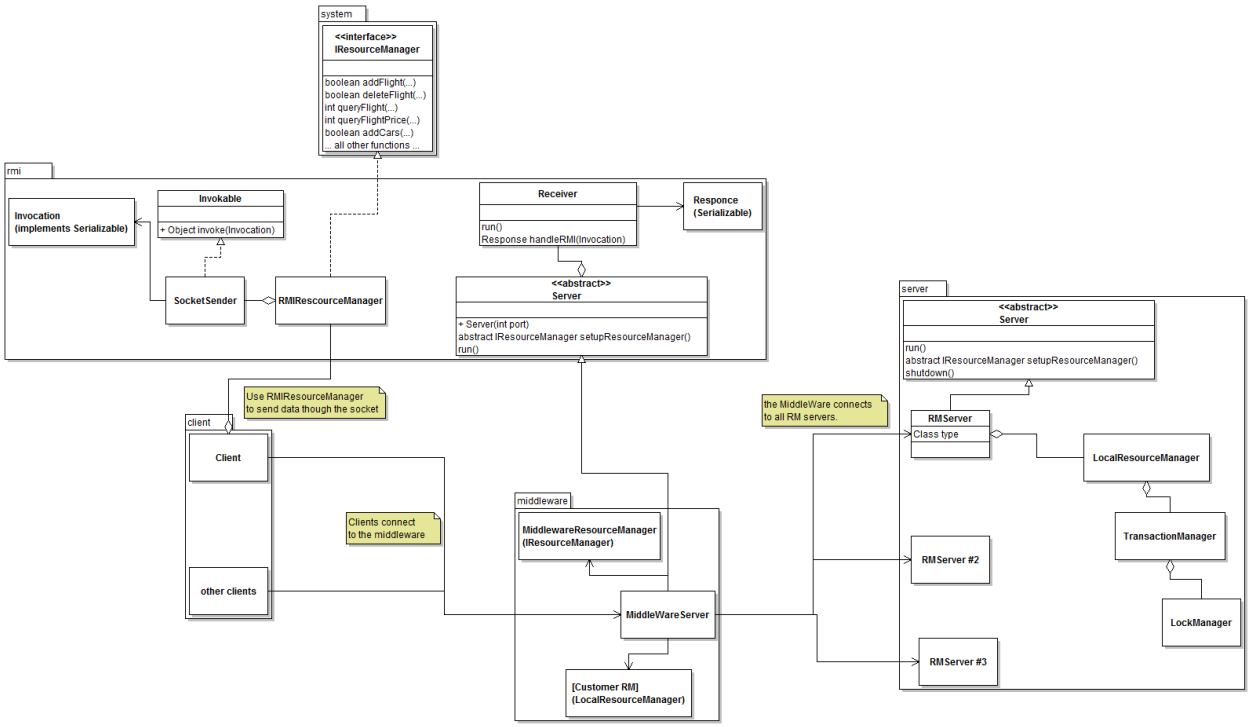
Guillaume Labranche

Serguei Nevarko

Comp512-Project2

Functionality Overview:

This project was written using sockets. The middleware connects to 3 RM servers and clients connect to the middleware. To illustrate this process in details , here is a functional view of the process (The picture is small in the document so please look at the file functionalView.png):



Client:

The client class is the starting point of the client program. It does all the setup, provides the program loop and reads the users input. The client contains an RMIResourceManager object that it uses to do all the functionality.

RMIResourceManager (rmi package):

This class it implements all the functions that are supported by the client. It uses the SocketSender class to send requests and knows what objects it has to send as parameters and what type of object comes back from the MiddleWare.

SocketSender:

This class contains the socket connected to the MiddleWare and sends objects of type Invocation through the network. It receives an object back from the MiddleWare and passes it to the RMIResourceManager.

Invocation:

This object holds the function name and the parameters needed for the function that we want to call on the MiddleWare (that will then be called in the proper RM). This object gets serialized and then sent through the network.

MiddleWareServer:

This class sets up the MiddleWareResourceManager, connects to the chosen RM servers and listens to incoming connections. When a client connects it creates a Receiver class that is a thread and keeps looking for connections.

The MiddleWareServer also deals with the received deadlock signals or transaction type requests from the client.

Receiver:

This class is a thread that holds the connection with a single client. When it sends data back to the client it creates and returns a Response object.

MiddleWareResourceManager (implements IResourceManager):

This class implements all the functionality supported by the client (and RM server). It also holds all the transactions of the clients using the TransactionManager object.

RMServer:

The RM server contains a “type” that defines if it is going to work with flights, cars or room. It implements the Server class (from the server package) so it listens to all incoming connections from the MiddleWare.

It also contains an RMIResourceManager object that is used to execute the commands.

LocalResourceManager:

This is what provides the functionality of the RM server by actually executing the commands passed by the MiddleWare. The server then sends back to the MiddleWare what this class returns (in the form of Response object).

LocalResourceManager also uses the TransactionManager to keep track of the transactions. The transaction manager holds all the locks and returns a deadlock exception when a deadlock occurs.

Customer RM:

We call it a Customer RM, but it’s an object inside the middleware that is used to create customers. Every time we need to create or change a customer, we first call the Customer RM and then we forward everything to the 3 other RMServers. For example, if we want to create a new customer we first create it in Customer RM, it send back a customer id, and we then create the customer in each RMServer using that id.