Comp 512: Deliverable 1

Jacob Errington (260636023)
Alexandre Laporte (260635979)

October 6th, 2015

Part 1: WebServices

General Architecture:

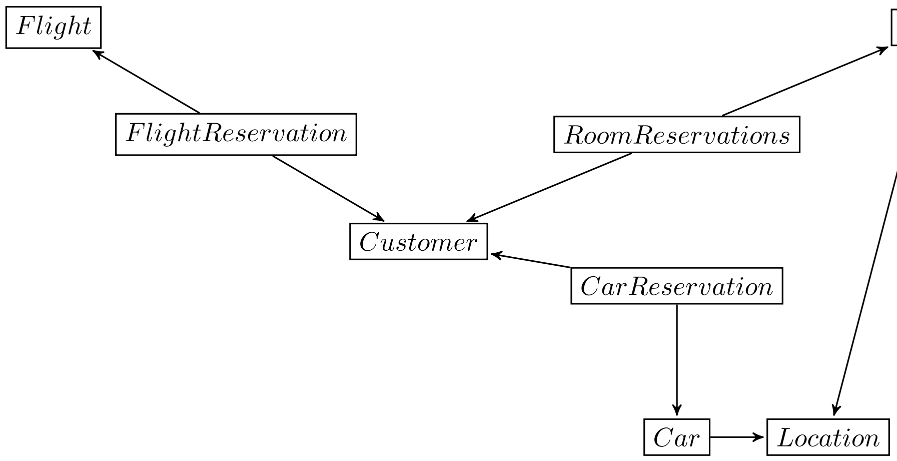
The design of the webservies component of the deliverable was centered around creating a the middleware manager. In order to acheive this we created a middleware server which implements the server's Resource Manager interface.
This middleware program is implemented as follows, it manages three Resource Managers which each point to servers, namely the flight Manager, car Manager and room Manager. Its task is then to process requests from a client and forward these to the appropriate server for handling, returning value provided by the server back to the client.
A notable exception to this general process is the handling of adding clients which is handled by the middleware in conjunction with the database server (see below).

The server code also had to be changed in order to account for concurrent access to data and synchronization between servers. To this end we added an additional server, our database, which hosts a POSTGRESQL database. All the server code was then changed to rather than use distributed hash tables, query and write to the database. This gives us a guarantee that upon concurrent access of data no problems arize. Additionally connections to the database are handled using DBCP connection pools ensuring that concurrent connections are handled without issue.
Additionally in the case of customer queries from the client the middleware interfaces directly with the database, likewise with the reserveItinerary function.
This means that our system has the following structure:

$$\begin{tikzcd}
& & Server2(carManager) \arrow[dr] & \\
ClientServer \arrow[r] & MiddlewareLayer \arrow[r] \arrow[ur] \arrow[dr] \arrow[rr, bend left=70] & Server1(flightManager) \arrow[r] & DatabaseServer \\
& & Server3(roomManager) \arrow[ur] &
\end{tikzcd}$$

(1)

Database Architecture: For clarity the database architecture is included here



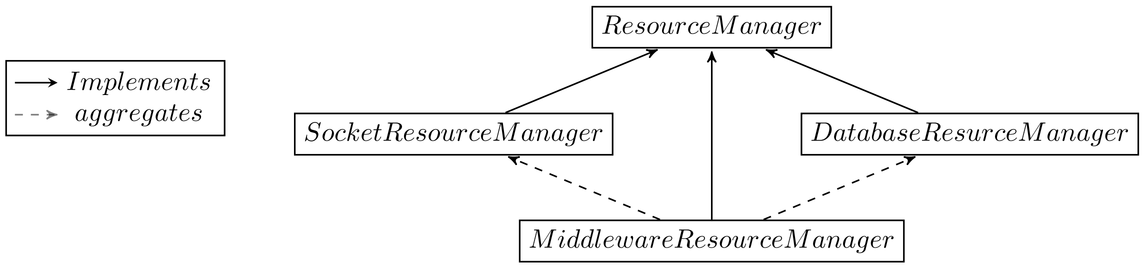
(2)

Part 2: Socket Code

The overaching architecture of the websockets code is identical to the structure shown in (1). As such the description will be focused on the implementations of webservices on top of websockets.

We begin by creating an executor service and oppening a websocket. The server then listens on the socket of all incomming connections, once a connection is opened it is accpted and packaged into a request context which contains a RequestManager and the socket connection. The request manager implements runnable and is passed to the executor service which assigns the connection a thread and invokes Request Manager's run. The executor service guarantees that connections are run asynchronously and act as a threadpool.

The RequestManager's run fetches the socket from the request context, upon receiving an input stream we use Java builtin serialization to pass our own request and response objects, this allows us to gracefully handle exceptions in a network transparent manner. The Resquest Handler of the recipient then uses Java's build in reflection toolset to identify which method should be invoked providing us with a method of remote method invocation through sockets. As per the webservices code database access is handled through DBCP connection pools.



(3)

The above diagram can be read as follows, SocketResourceManagers represent client Resource managers, their purpose being to communicate with the middleware and forward method calls. The databaseResourceManager represents the server Resource managers where every call is either a database read or write operation. The MiddlewareResourceManager aggregates both these managers as the middleware can either act as a client to the databaseResourceManager or perform database operations, specifically in the event of customer operations.