Architectural Styles and Patterns

This application is structured as a classical client-server system, and more specifically this is the case of three-tier architecture (client, server and database) composed by a thin client (the group of pages of the website, they only need to send informations, no operations are executed on the client side) and a fat server (the whole management system and every operation are executed inside the server). Following the basic principles of software engineering, the ideas behind the project are the “divide et impera” concept and the code reuse. Every macro component exchange informations with others using RPCs (Remote Procedure Calls) following the Single Responsability Principle: every component is in charge of the operations related to a single task, asking other components for data not related to his specific role. Every macro component is then divided in many sub-components, in order to create atomic blocks of code ready to be used for different purposes, reducing work and complexity; every sub-component is linked to others by internal sockets.

During the application design, it was necessary to decide some specific architecture costraints in order to make everything work as planned. In this process design patterns are very useful, because they are structures that represent ready solutions to common problems.

In this document the choice is to present every solution adopted grouped by pattern type:

* Thread Pool

In every service in which is possible to make multiple concurrent request it is necessary to insert a Thread Pool pattern in order to manage the requests, and to maximize the occupation of every service “worker” to obtain the best performance.

In this particular project this kind of pattern is necessary to manage the multiple taxi requests or reservations from users, the procedure of taxi assignement preformed by the queue manager, and data access and modification by multiple users.

* Façade

The Façade pattern is used to hide the details of complex blocks of components with different interfaces in order to simplify the communication between these blocks and the rest of the architecture.

In the project this pattern is used to allow the communication between the application and the database, solving possible compatibility problems, to simplify the taxi assignement hiding the division of queues, and to allow the use of localization system without knowing implementation details.

* Observer

This pattern is the best way to trace the changes of a particular component; it is composed by an object called “subject” related to an object called “observer”: every change in the subject state is recorded by the observer, often triggering actions.

In this case the Observer is used by the notification component that, as an observer, is ready to listen to every event generated by requests, and send proper notifications to the involved users.

* Read-Write Lock

Another useful pattern to manage concurrency: it is used in the database to control multiple access on data and preventing errors due to data corruption or simultaneous operations on the same record.