**Selected architectural styles and patterns**

Use cases analysis in the RASD suggests that main actors of PowerEnJoy system will be Guests, Users, Drivers and Operators. These actors must not share anything with each other but they only need to send requests to the system and to retrieve responses. In addition, by definition they can access only a limited set of all system functionalities, each of them according to their actor types. Furthermore, actors are geographically distributed and they can even move during their communication with the system.

There will be the necessity to control some distributed system devices such as cars and special parking areas.

All these system properties lead us to use a Client/Server architectural style which basically rely on the MVC (Model View Controller) and Observer design patterns (even though it has not been explicitly stated before but they are inherently used in the Java EE environment).

The server side will manage all what concern part of the presentation layer, the business logic and the persistent data management. All these components are centralized in order to be available for every client that request it. So, most of the functionalities have not to be replicated onto the clients but just collected and made them central and accessible.

The client side is dived into two categories: PowerEnJoy clients and customer clients. PowerEnJoy clients are Operator Clients, Car Clients and Special Parking Area Clients. Customer Clients are Computer Clients and Smartphone Clients.

The operator management is integrated into the Client/Server infrastructure but do not follow that architectural style. Indeed, a Publisher/Subscriber pattern fits better in this case: once an Operator Client is available to serve the system, it requests to the Server Dispatcher to be added to the listeners list. In this manner, every time the Server will send a notification to operators, it will have the list of available operators ready to serve (listeners, exactly). The two used patterns cooperate to come up with a hybrid infrastructure solution for our system.

Nowadays, the Client/Server infrastructure is one of the most suitable for most of web oriented software applications. The fact that the core business is central and under a more restrictive control makes this infrastructure reliable and (at least in theory) more secure. Many of the execution parameters such as robustness and reliability can be easily monitored and further software improvements can be taken basing on feedbacks. Moreover, performance evaluations can be performed as the system starts to be used by users and in case it may be horizontally scaled. The horizontal scaling is possible thanks to the load balancer inside the Front End Server. A backup database is employed in order to recovery from disasters. The centrality of the core system makes it more maintainable by system and software engineers.

The main reason to choose Java Enterprise Edition is for portability, as the Java (Sun Microsystems) states: “write once, run everywhere”. Portability is fundamental to make it possible to reach as most as possible catchment area.

Below is presented a typical Java EE Client/Server infrastructure.

