Selected Architectural Styles and patterns

The overall architecture we will adopt to develop the PowerEnJoy platform will be a three-tier architecture with distributed logic. Using this architecture, we will be able to decouple the business logic layer from the presentation and the data layers.

Our central system (where we can find the data layer and most of the business logic) needs to communicate both with the user and the vehicles. To achieve this communication, we will use:

-A client-server architectural style for the communication with the user. In this case, the client is the user device (pc or mobile device). The client side takes care of the presentation logic and will also contain some business logic (mainly the one used to communicate with the Google Maps service). We will however try to keep the client as thin as possible.

-An event based paradigm for the communication between the central system and the vehicles. Since the vehicles will need to notify state changes, the central system will register itself as a listener to all the vehicles and will act accordingly to these state changes. Moreover, the car must expose some methods to the central system (such as the one to unlock the car).

**Design Decisions:**

**Google Maps API**: the user device will interact directly with Google Maps services through Google Maps APIs. This will allow the device to display useful maps used by the user to access PowerEnjoy services.

**Java Persistence API:** we will use JPA for accessing, persisting, and managing our database, since JPA is already implemented in JEE.

**Design Patterns**:

Client-server: as aforementioned in the introduction of this chapter, we will make large use of the Client-Server design pattern, using it in the communication between the user device and the central system. To achieve this, the server will expose some methods (using a REST architecture) that will be used by the client to access the PowerEnJoy platform services.

Adapter: we will be using many interfaces between the components of our central system. In order to implement these interfaces, we will need to adopt the adapter design pattern.