

POLITECNICO DI MILANO

2016-2017

Software Engineering 2: PowerEnJoy

Design Document

Version 1.0

Peverelli Francesco

Reppucci Federico

**TABLE OF CONTENT**

|  |  |
| --- | --- |
| 1. INTRODUCTION  1.1. PURPOSE  1.2. SCOPE  1.3. GLOSSARY  1.4. REFERENCES  1.5. SUMMARY  2. BODY  2.1. OVERVIEW  2.1.1 REQUIREMENTS MAPPING APPROACH  2.1.2. DESIGN DECISIONS METHODOLOGY  2.2. COMPONENT VIEW  2.2.1. EXTERNAL COMPONENTS AND INTERFACES  2.2.2 SOFTWARE COMPONENTS VIEW  2.3. HIGH-LEVEL SYSTEM ARCHITECTURE  2.3.1. SOFTWARE ARCHITECTURAL PATTERN  2.3.2. SYSTEM ARCHITECTURE  2.4. COMPONENTS INTERACTION  2.4.1. COMPONENTS INTERFACES  2.4.2. RUNTIME SEQUENCE DIAGRAMS  2.5. COMPONENTS ARCHITECTURE AND PATTERNS  2.5.1. COMPONENTS ARCHITECTURAL VIEW (ALSO DB: ER DIAGRAM)  2.6. USER INTERFACE DESIGN  2.7. SELECTED TOOLS  2.8. DEPLOYMENT VIEW  2.8.1. RUNTIME DEPENDENCIES VIEW  2.8.2. SOFTWARE COMPONENTS MAPPING  2.8.3 TECHNICAL ENVIRONMENT REQUIREMENTS  2.9. ALGORITHM DESIGN  2.10. REQUIREMENTS TRACEABILITY  3. EFFORT SPENT |  |

***1. INTRODUCTION***

***1.1. PURPOSE***

The aim of this document is to provide a complete specification of the PowerEnJoy system's architecture which fulfills the requirements identified during the requirements specification phase. More specifically, this document is meant to serve both as a way to clarify to the stakeholders how the specified system will fulfill the requirements and how the requirements have motivated the design decisions presented, as well as a reference for the developers of the system that will guide the implementation phase.

***1.2. SCOPE***

The PowerEnJoy project aims to develop a car-sharing service run exclusively employing electric cars. The system will provide a mobile application by means of which the users, once registered, will be able to use the car sharing services. The main goals of the service are to provide a sustainable and environmentally-friendly car sharing service as well as to promote virtuous behaviors from its users.

***1.3. GLOSSARY***

***1.4. REFERENCES***

Francesco Peverelli and Federico Reppucci. “*RASD\_v1.1.pdf*”, 13 Nov. 2016.

IEEE Computer Society. “*IEEE Standard for Information Technology—Systems Design— Software Design Descriptions*.” Institute of Electrical and Electronics Engineers, Inc., 29 July 2009.

Fakhroutdinov, Kirill. “The Unified Modeling Language”. *Unified Modeling Language (UML) Description, UML Diagram Examples, Tutorials and Reference for All Types of UML Diagrams - Use Case Diagrams, Class, Package, Component, Composite Structure Diagrams, Deployments, Activities, Interactions, Profiles, Etc.*, [www.uml-diagrams.org/](http://www.uml-diagrams.org/).

***1.5. SUMMARY***

*1. OVERVIEW:* this section explains how the document is structured, and the reasoning behind the choice of this particular approach.

*2. COMPONENT VIEW:* this section shows the external software services that interact with the system and the structure of the system's main software components.

*3. HIGH-LEVEL SYSTEM ARCHITECTURE:* this section describes the Software Architectural Pattern of choice as well as the physical architecture of the system.

*4. COMPONENTS INTERACTION:* this section describes more precisely how the software components are connected, which interfaces they provide and how they interact with each other.

5. COMPONENTS ARCHITECTURE AND PATTERNS: this section specifies for each software component its architecture, as well as the main design patterns used to construct it and their specific purpose.

*6. USER INTERFACE DESIGN:* this section describes in detail the functionalities of the user interfaces.

*7. SELECTED TOOLS:* this section specifies all the main frameworks, languages, libraries and tools to use during the development of the system.

*8. DEPLOYMENT VIEW:* this section describes how the software components are mapped onto the system's hardware and how they interact with each other at runtime. Moreover, a set of requirements regarding the execution environment of the software is identified.

*9. ALGORITHM DESIGN:* this section describes the most critical algorithms to implement, providing a pseudo-code implementation.

*10. REQUIREMENTS TRACEABILITY:* this section explains how the requirements previously identified are met by the system described in the document.

***2. BODY***

***2.1. OVERVIEW***

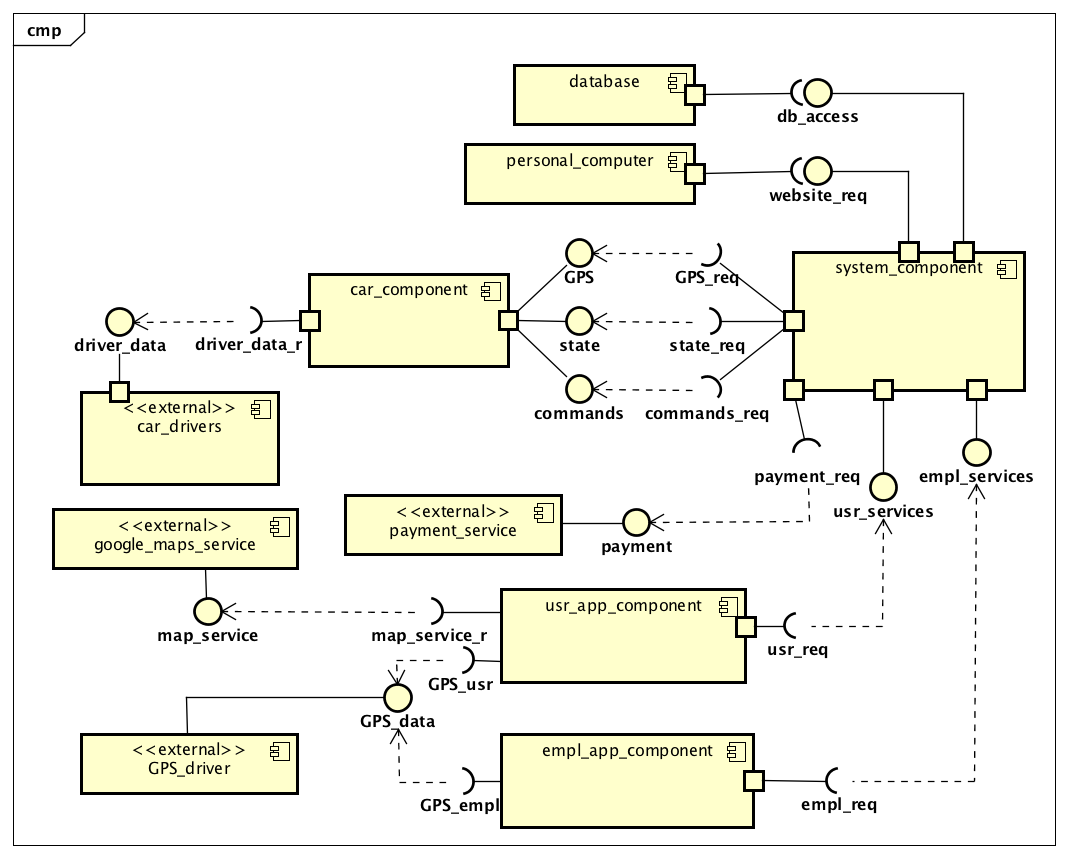
The structure of this document is meant to reflect the actual stages of the development of the architecture and design of the system, in order to illustrate not only the final result of the design phase, but also the intermediate steps of this process, and the rationale behind every major design decision. In order to achieve this, the document does not follow a strictly top-down structure, but starts from the design decisions that follow most directly from the requirements, namely the design of the system's main functional components. It is important to note that despite this difference in the rationale behind the structure of the document, for the most part the approach is still top-down, since such an approach is the most natural way of obtaining a cohesive design.

***2.1.1 REQUIREMENTS MAPPING APPROACH***

***2.1.2. DESIGN DECISIONS METHODOLOGY***

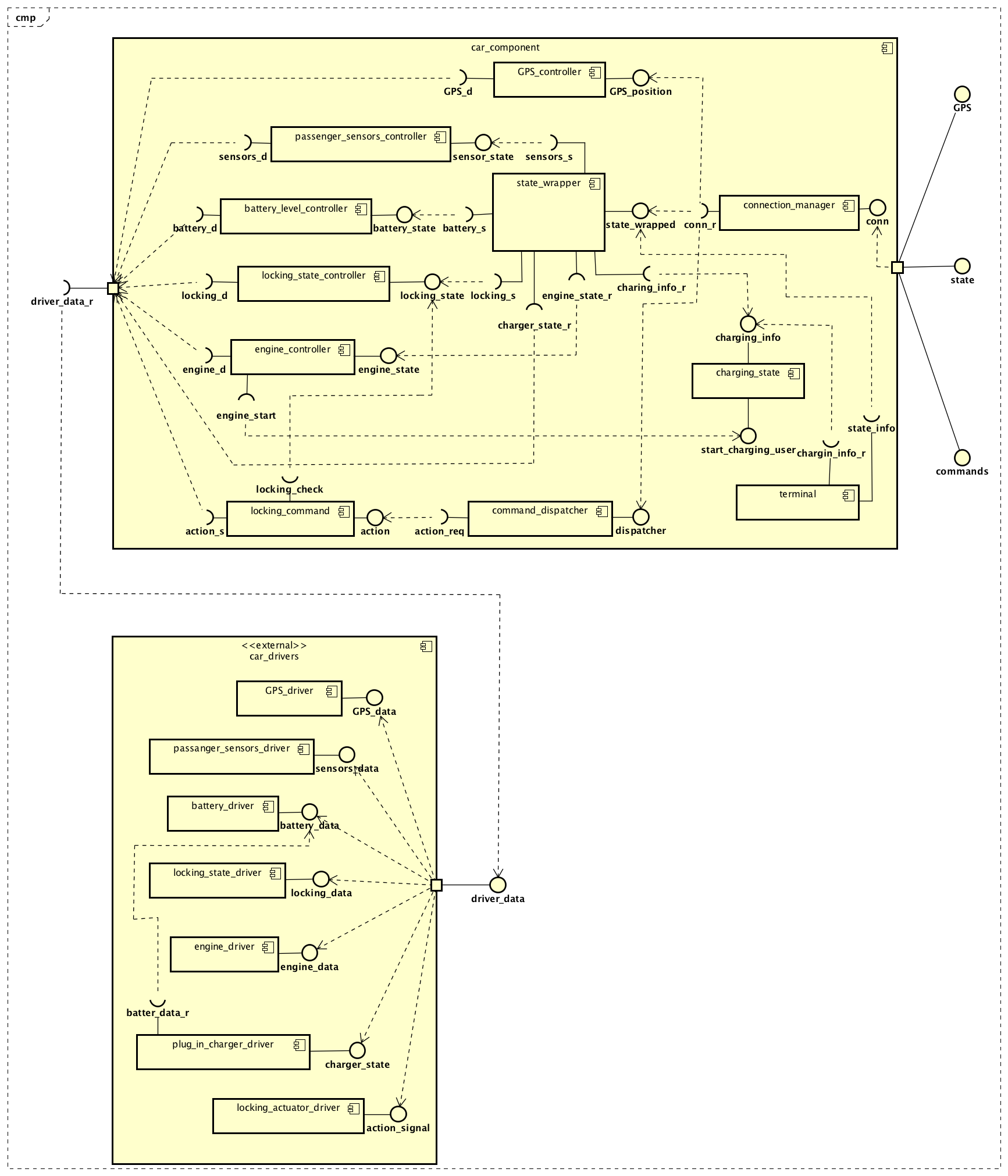
***2.2. COMPONENT VIEW***

***2.2.1. EXTERNAL COMPONENTS AND INTERFACES***

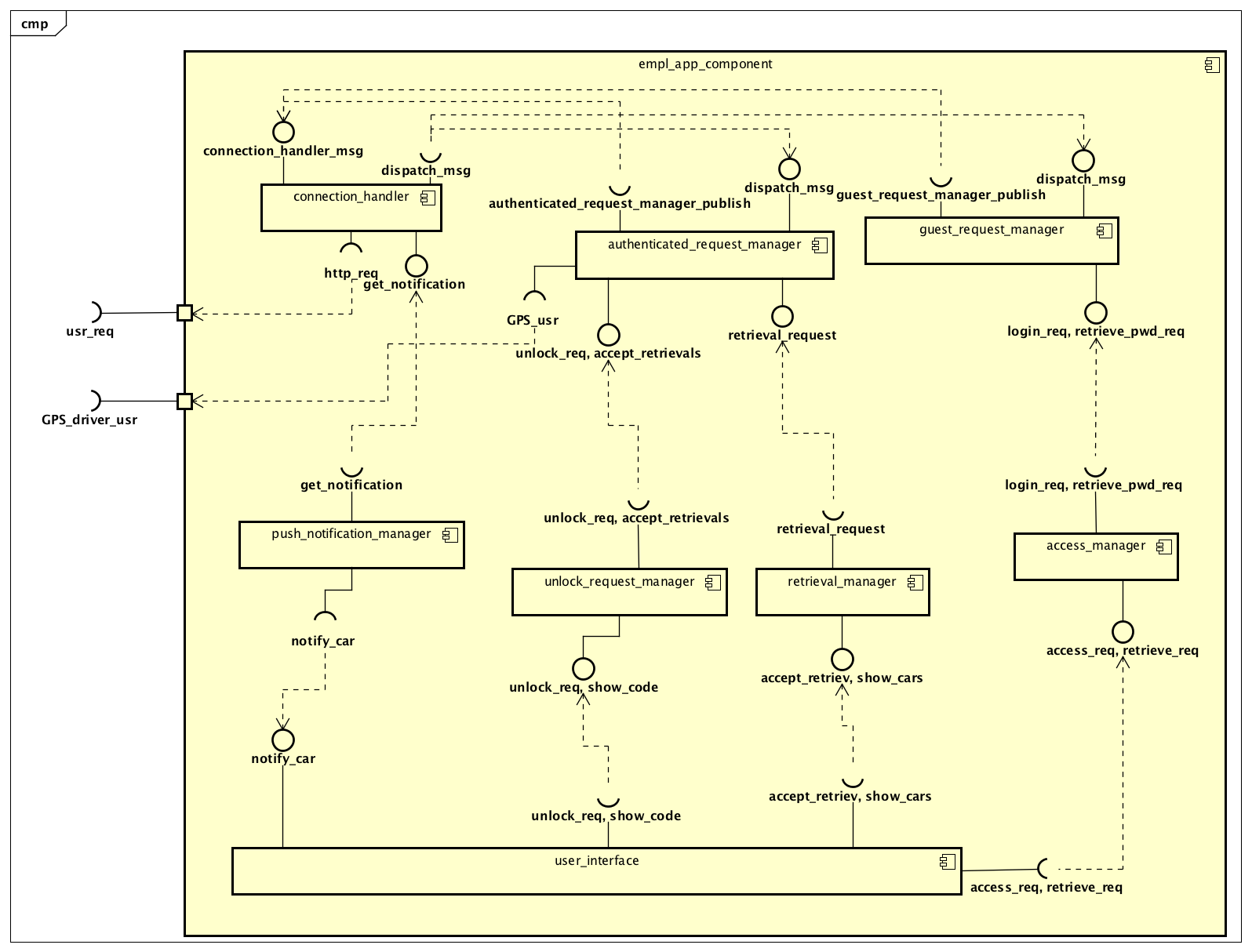
******

***2.2.2 SOFTWARE COMPONENTS VIEW***

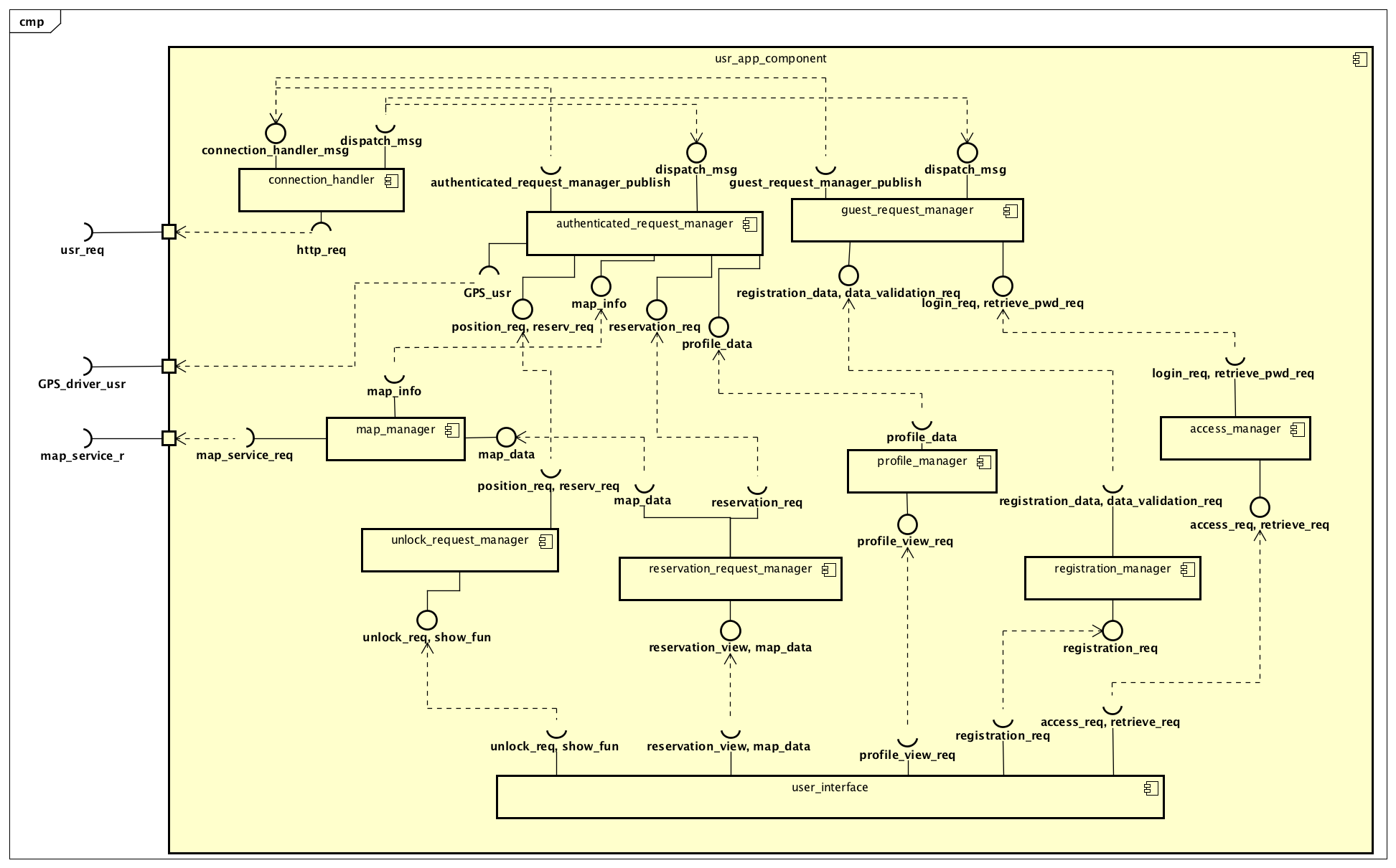
*Car*

******

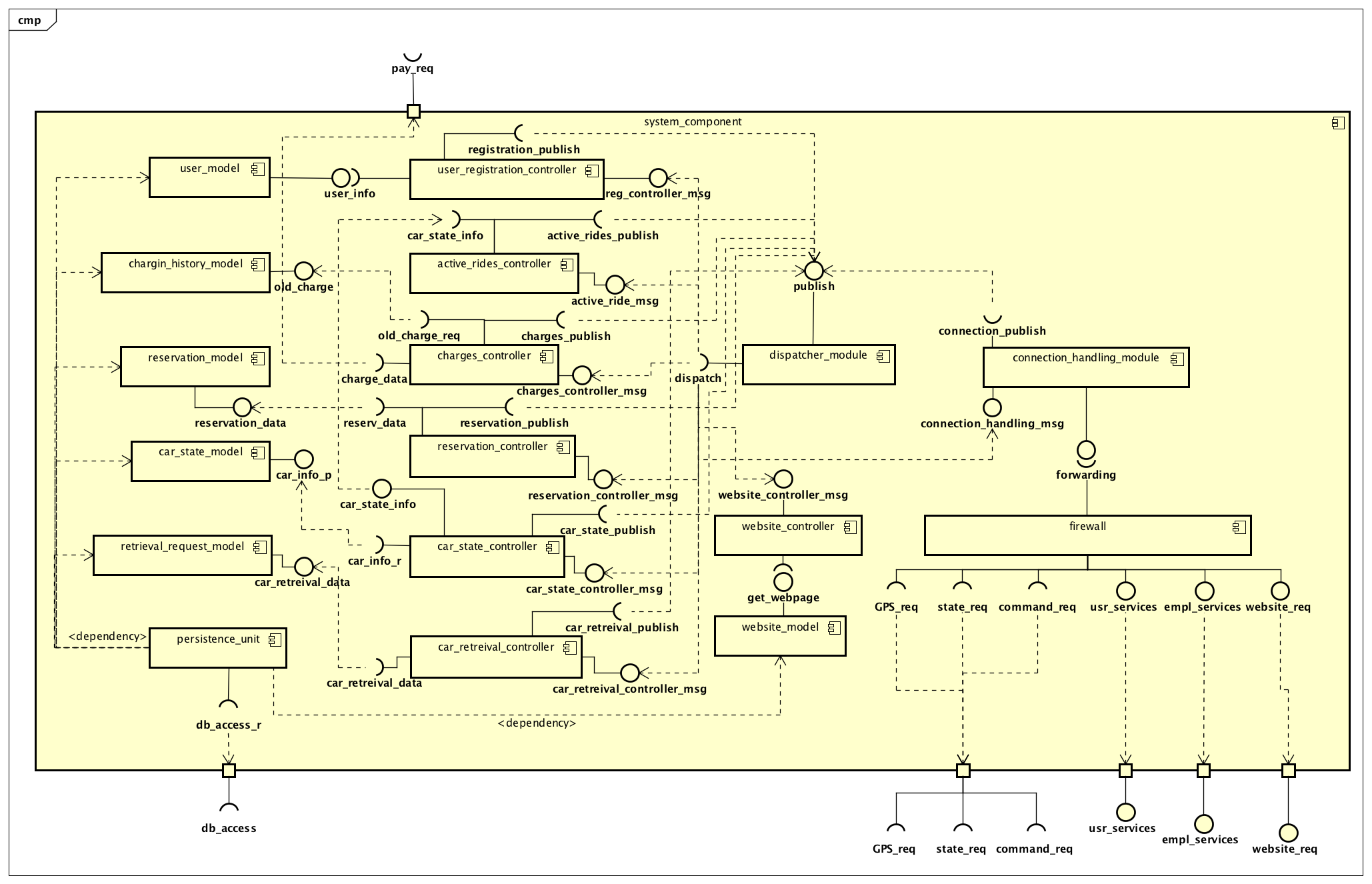
*Employee application*

******

*User application*

******

*System*

******

***2.3. HIGH-LEVEL SYSTEM ARCHITECTURE***

***2.3.1. SOFTWARE ARCHITECTURAL PATTERN***

Based on the components individuated to carry out all the tasks required of the system, the architectural patter most suited for the system is the event-based (or pub-sub ) model. The main reason for this choice is the need for the system to communicate with different software components, such as cars, the users and the employees' applications, and receive and send messages related to different topics. One main objection to the adoption of this pattern may be that there are not so many one-to-many event-driven communications, but we can for example point to the notification system for car retrieval and the system requesting the car's position to allow a user to reserve a car as the two main examples.

***2.3.2. SYSTEM ARCHITECTURE***

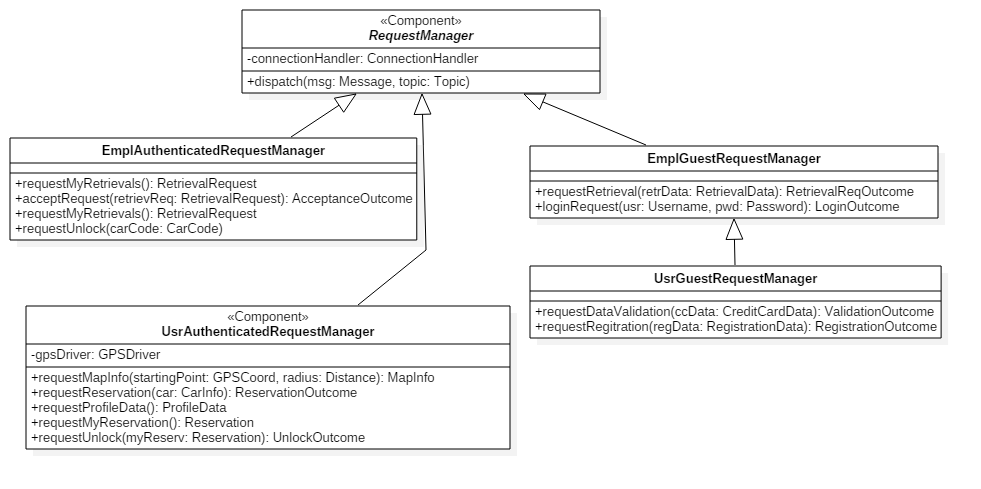
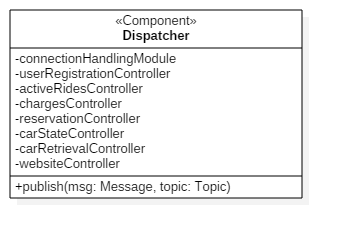
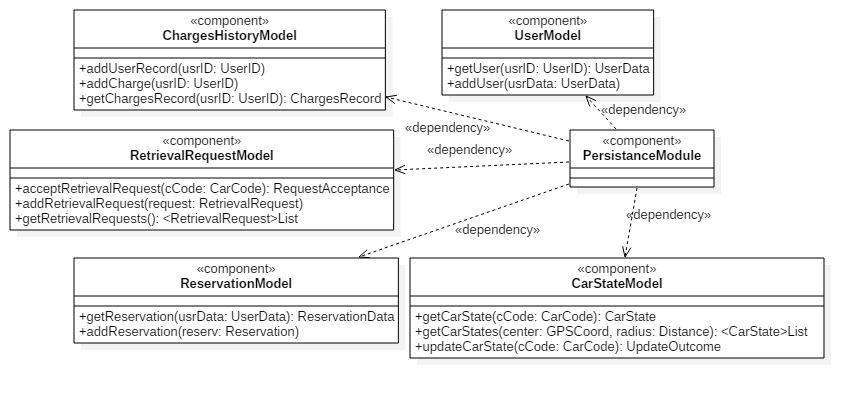
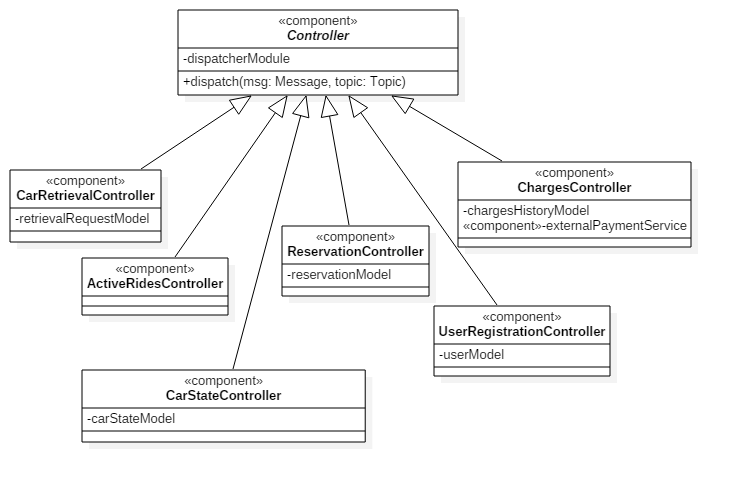
The system architecture naturally follows from the functionality of the components previously described. The result is a three-tier architecture, where the presentation layer is located on the mobile applications and the car, the business logic is almost entirely on the server's side (although both the mobile apps and the software systems on the car contain some control logic, it is mostly used to formulate requests for the server to evaluate, or to pass on messages to act upon), and the persistency layer comes in the form of a database component.

../2.%20BODY/3.%20HIGH-LEVEL%20SYSTEM%20ARCHITECTURE/ArchitectureDiagram.png

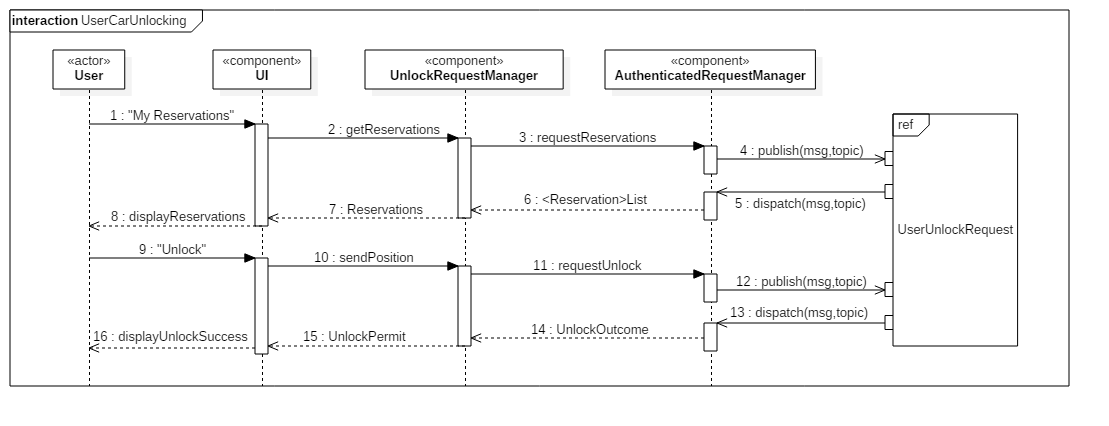
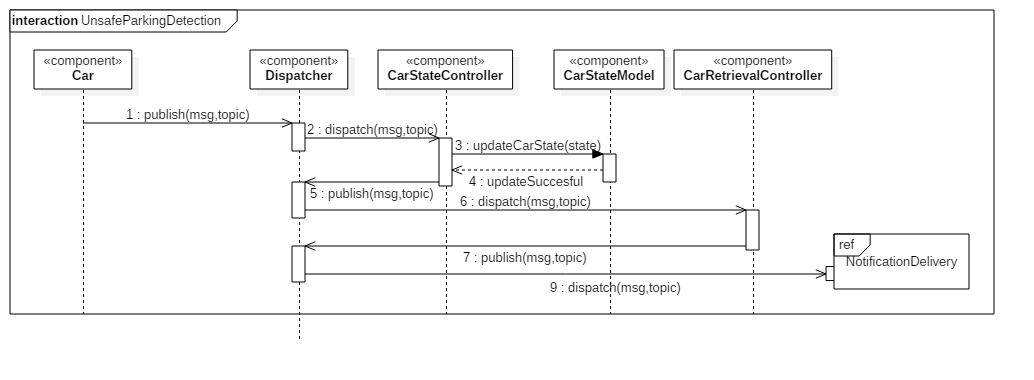
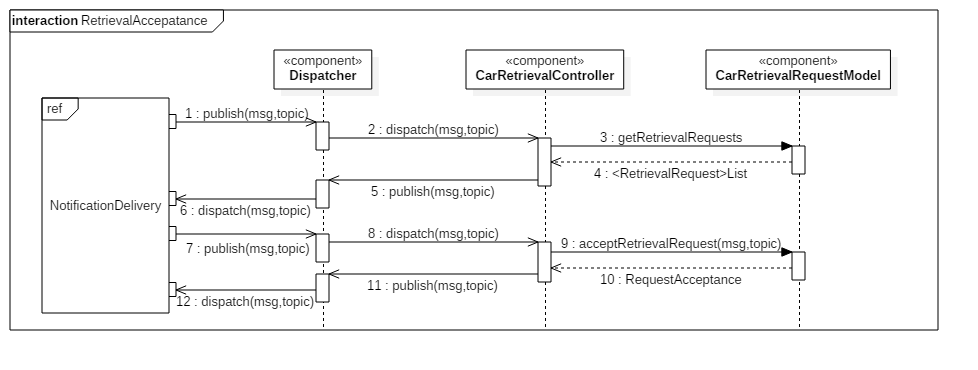
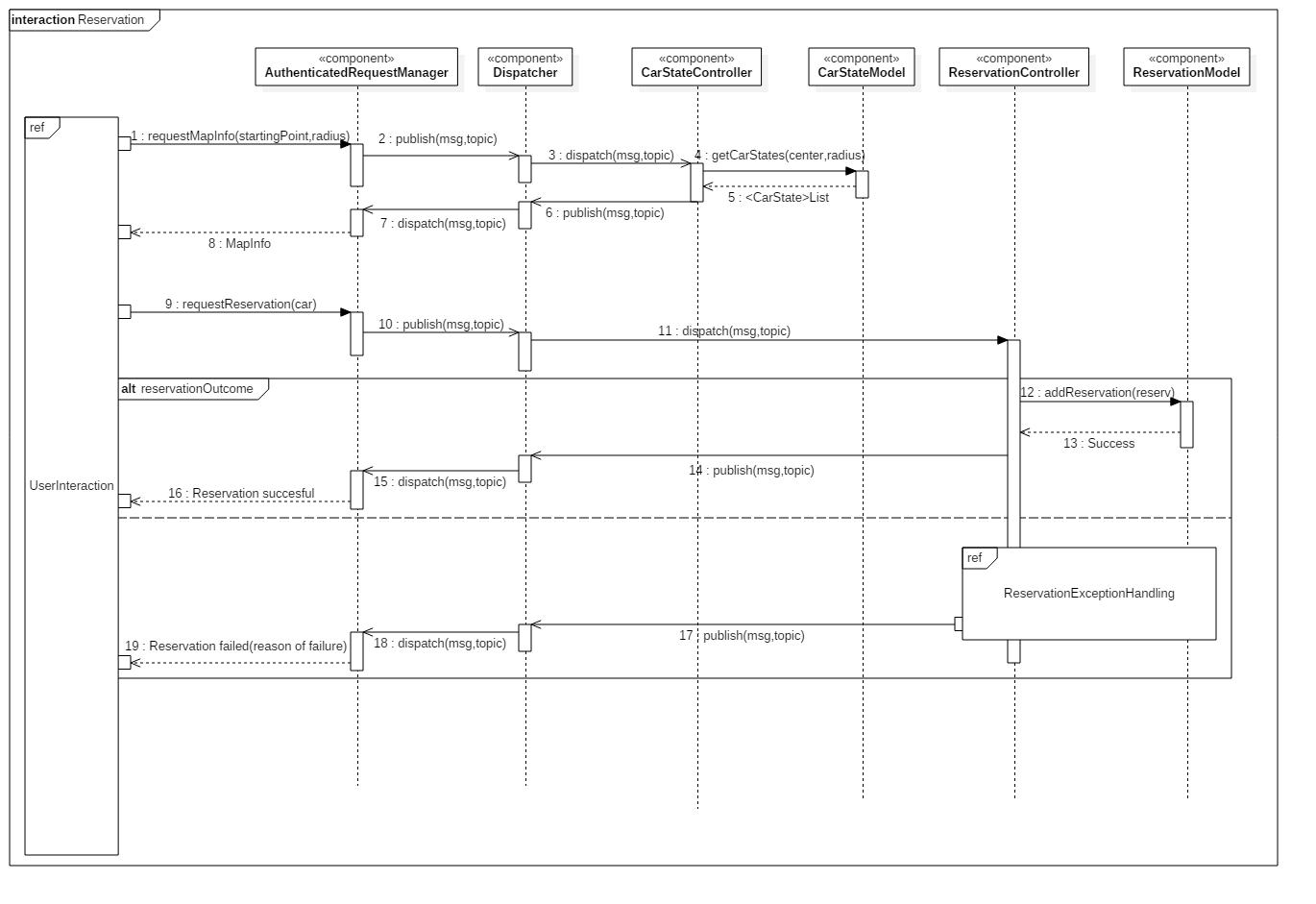
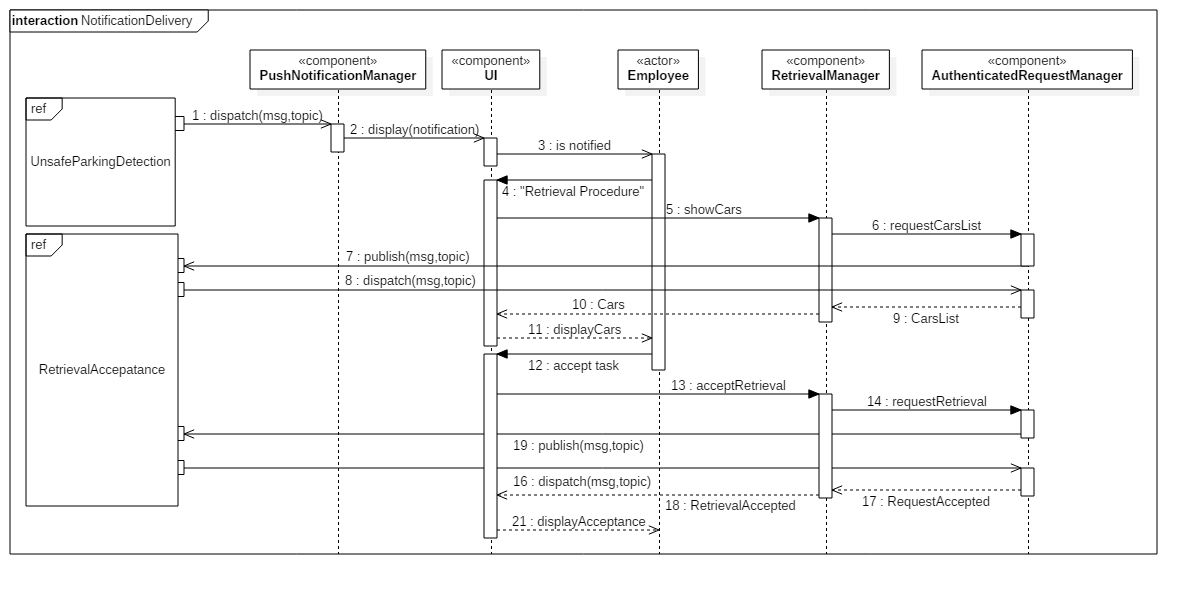
***2.4. COMPONENTS INTERACTION***

***2.4.1. COMPONENTS INTERFACES***

The following diagrams specify the interfaces of some of the main business-logic components of the system. The components are represented as classes of a UML class diagram, where each component has public methods representing the interfaces it offers to other components, and private attributes representing all the components which can be accessed by that specific component.

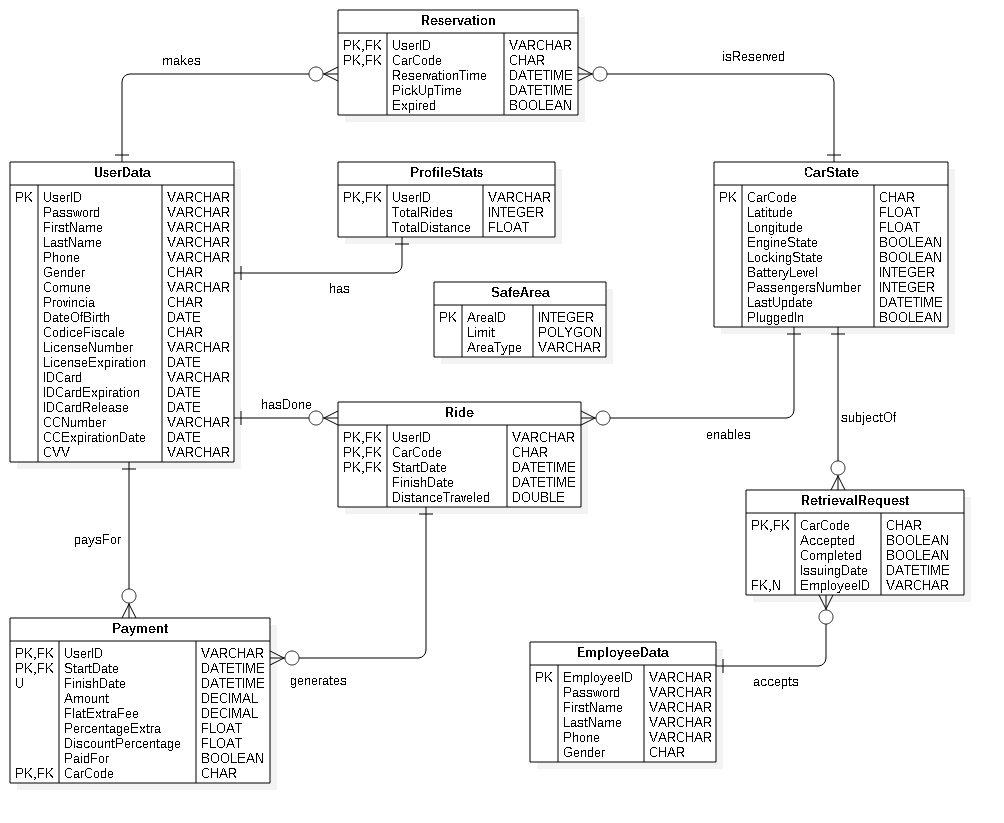


***2.4.2. RUNTIME SEQUENCE DIAGRAMS***

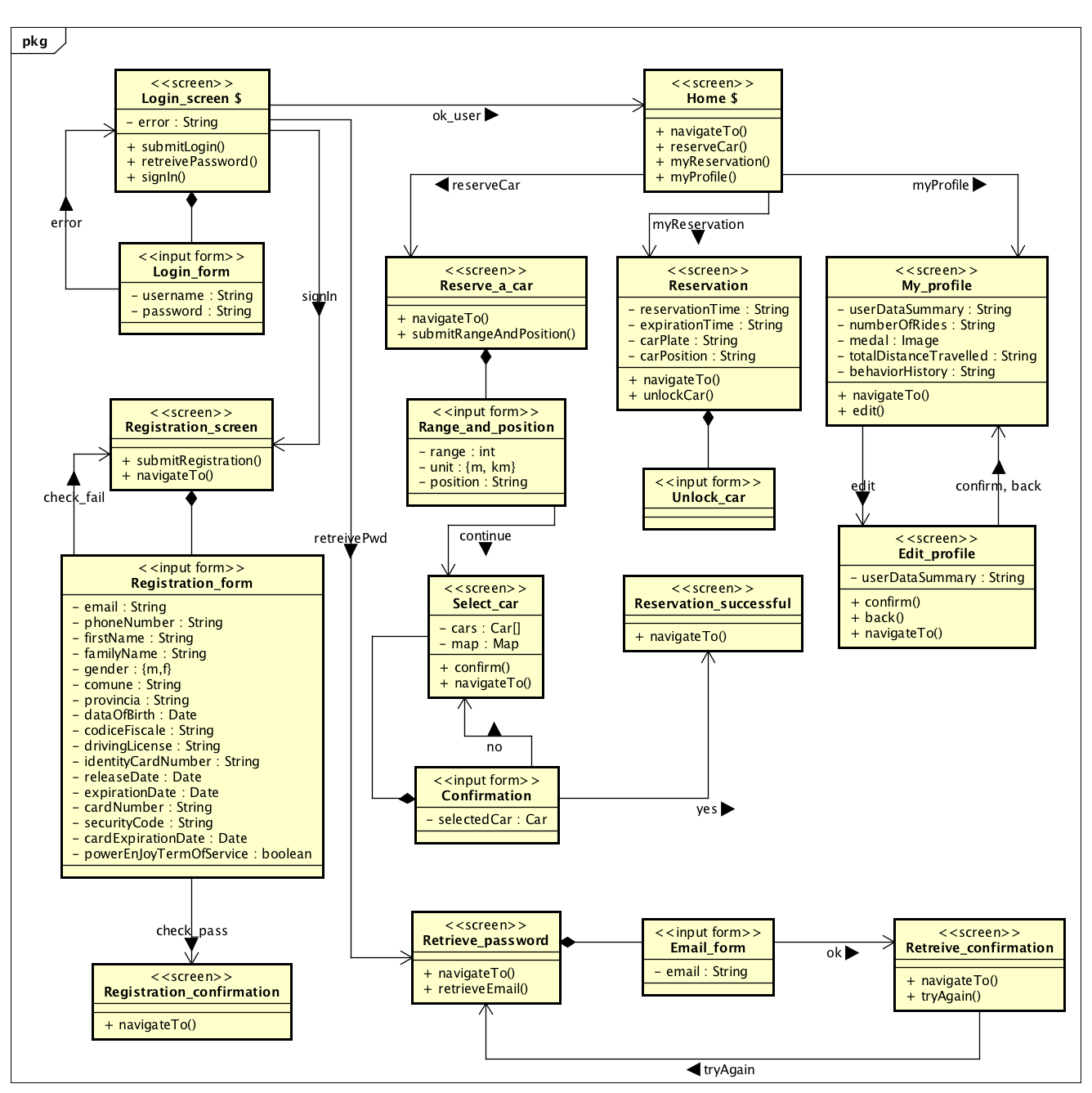


***2.5. COMPONENTS ARCHITECTURE AND PATTERNS***

***2.5.1. COMPONENTS ARCHITECTURAL VIEW (ALSO DB: ER DIAGRAM)***



***2.6. USER INTERFACE DESIGN***



***2.7. SELECTED TOOLS***

*Operating systems*

SUSE Linux Enterprise 10: is the operating system running on the server machines

Microsoft Windows Embedded Automotive: is the operating system that the provided cars come with

Windows, iOS and Android are identified as the operating systems for which the mobile application will be developed

*Application server*

Payara: is an open-source application server derived from Glassfish, and although it's not Java EE certified it is effectively Java EE 7 compliant and offers far more stable patch releases, security fixes, production support and developer support. It also has a very responsive community.

*Database*

MySQL: among all the available DBMSs, MySQL stands out for its scalability and flexibility, which also comes along with good performances and availability. In addition, it is open source, just like Payara, and can run on multiple platforms. All these reasons contribute to make MySQL our DBMS of choice.

*Frameworks*

J2EE: is a solid framework which will be used to ease the development of the application logic and the presentation layer for both the website and the apps on the server side.

Windows Automotive Application Framework: is chosen as a native framework that our windows developers are familiar with.

*Communication*

On the server side JMS is used as the messaging API, and the messages are exchanged in a text-based XML format via SOAP Transport Protocol.

On the client side Kaazing WebSocket Gateway APIs are used to support JMS messaging

*Ides*

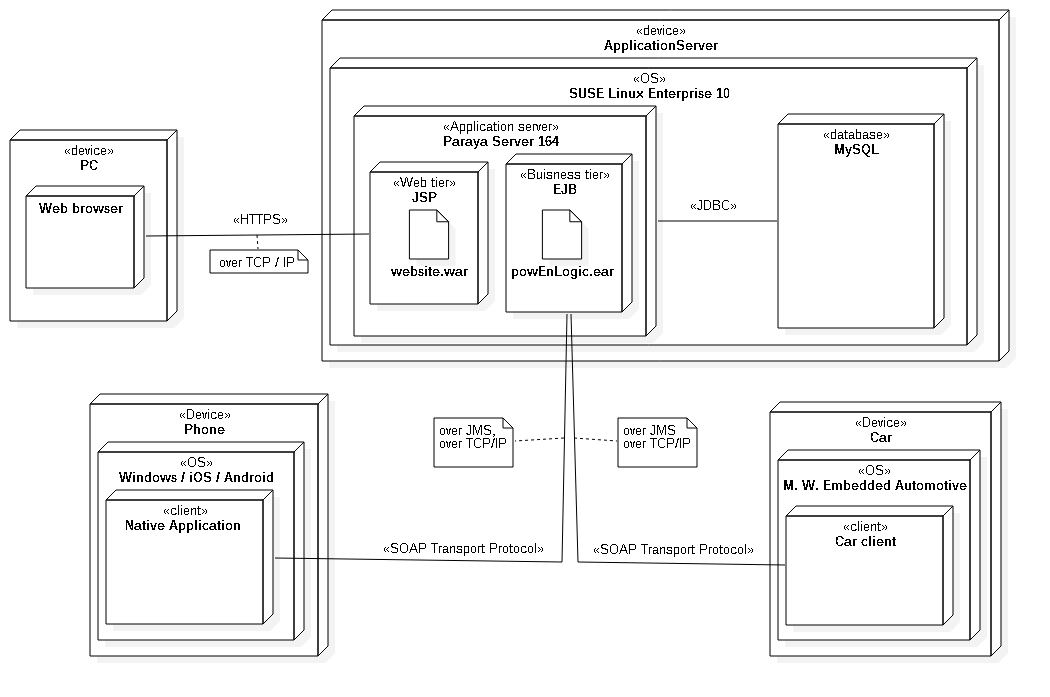
The selected IDEs are NetBeans for the development of the server-side application, VisualStudio for the development of the software running on the cars, and AndroidStudio, XCode and VisualStudio are the integrated environments selected to develop for Android, iOS and Windows respectively.

***2.8. DEPLOYMENT VIEW***

***2.8.1. RUNTIME DEPENDENCIES VIEW***

***2.8.2. SOFTWARE COMPONENTS MAPPING***

The deployment of the software components follows naturally from the previous design decisions: the native application clients will be installed on the clients and employee's phones, according to the phone's OS. For the application server a device of the IBM Power System series has been selected, which provides good flexibility and scalability potential, as well as reliability and is in general a solid choice for medium-sized enterprise applications. The application server software modules are organized in an. ear archive to guarantee a coherent deployment. Another .war file has been created to separate the software components which realize the website service, in case this module needs to be moved to another machine in the future.



***2.8.3 TECHNICAL ENVIRONMENT REQUIREMENTS***

No particular conditions need to be met to allow the system to function properly, barring some basic needs such as a suitable physical space to collocate the application server and a good enterprise Ethernet connection.

***2.9. ALGORITHM DESIGN***

***2.10. REQUIREMENTS TRACEABILITY***

In this section for each functional requirement are listed the components which achieve its fulfillment. The connection handling components, the firewall and the dispatcher are always omitted, since they do not serve any specific functionality, despite being involved in all the communications between the server and the different clients and devices.

[R1.1]: The app is available for any person to download and run on his/her phone

This requirement will be achieved at a later stage.

[R1.2]: From the home page of the app any person can carry out the registration procedure

[R1.3]: The registration procedure requires a person's credentials and payment info to be carried out

[R1.4]: The registration procedure uses the external payment service to verify the validity of the provided payment info

[R1.5]: At the end of the registration procedure the person whose credentials were used is registered in the system

[R1.6]: At the end of the registration procedure the person receives an e-mail containing a password which he/she can use to access the system

[R1.7]: At the end of the registration procedure the person can ask the system to send another mail

*COMPONENTS*: UserInterface, RegistrationManager, GuestRequestManager, UserRegistrationController, UserModel

[RA1.1]: The app allows any person to log in by providing a valid e-mail and password

[RA1.2]: The app does not allow any person who does not provide a valid e-mail and password to log in

[RA2.1]: From the home page of the app the password retrieval procedure can be initiated by any person

[RA2.2]: If a person provides a valid e-mail address during the password retrieval procedure the system sends an e-mail to that address containing the associated password

*COMPONENTS:* UserInterface, AccessManager, GuestRequestManager, UserRegistrationController, UserModel

[RA3]: Access to the PowerEnJoy's website (a static page) is granted upon request by the system (no login required)

*COMPONENTS:* WebsiteController, WebsiteModel

[R2.1]: The "Reserve a car" function can be accessed by the user from the home page of the app

[R2.2]: The "Reserve a car" function allows the user to select a range (distance)

[R2.3]: The system acquires the user's current position through the GPS coordinates of the user's phone

[R2.4]: The system tracks all available cars' current position through their GPS coordinates

[R2.4.1]: The cars must possess a device which can be tracked via GPS

[R2.5]: The "Reserve a car" function allows the user to select a starting position for the search, which can be either their current location or a given address

[R2.6]: When the user confirms the inserted parameters the search is carried out and the "Reserve a car" function displays to the user the data of the search acquired from the system in a Google provided map

[R3.1]: The app allows the user to tap on any available car on the map displayed as the result of a search conducted through the "Reserve a car" function.

[R3.2]: When a user taps on a car the app generates a pop-up asking the user if he/she wants to confirm the reservation.

[R3.3]: As long as the car was not reserved by another user in the meantime, when the user confirms the car is marked as reserved by the system and the user can see the "Reservation successful!" message on the app.

[R3.4]: When the system marks a car as reserved any reservation request from any user is rejected by the system while the car is in the reserved state.

[R3.5]: A car is in reserved state for one hour from the moment it was marked as reserved.

[R3.6]: A car in reserved state is not signaled by the system during the "Reserve a car" procedure.

[R3.7]: After one hour from its reservation a car is no longer in reserved state.

[R3.8]: A car not in reserved state is considered available by the system only if it is parked in a safe area less than 3 km away from a power grid station and has more than 20% of its battery.

*COMPONENTS:* UserInterface, ReservationRequestManager, MapManager, AuthenticatedRequestManager, ReservationController, ReservationModel, CarStateController, CarStateModel,

+ StateWrapper and the controllers on the car providing status updates.

[R4.1]: One hour after a car has been reserved if it was never ignited the system charges for 1 EUR the user who reserved it

*COMPONENTS:* ReservationModel, ReservationController, ChargesController

[R5.1]: From the home page of the app the user can access the "My reservations" section

[R5.2]: In the "My reservations" section if the user has reserved a car less than an hour ago an active reservation is displayed with an "Unlock" button

[R5.3]: If the user is less than 10 meters away from the car and presses the unlock button the car unlocks

*COMPONENTS:* UserInterface, UnlockRequestManager, AuthenticatedRequestManager, ReservationController, ReservationManager, CarStateController, CarStateModel, CommandDispatcher, LockingCommand

[R6.1]: When a car is ignited the system starts charging the last user who reserved the car

[R6.2]: When the charging starts, the display on the car shows a "Current charge" field with a number representing the current total charge, which starts from 0

[R6.3]: Once a minute the "Current charge" value is incremented by a set amount

*COMPONENTS:* EngineController, Terminal, StateWrapper, ChargesController, ChargesHistoryModel, ChargingState

[R7.1]: When a car is stopped and the sensors in the car detect no one inside, if a user was being charged for the car the system stops charging him/her.

[R7.2]: One minute after a car has been stopped and the sensors in the car detect no one inside, the system locks the car.

*COMPONENTS:* PassengerSensorsController, LockingStateController, StateWrapper, ChargesController, ChargesHistoryModel

[R8]: The moment the car is stopped, if the sensor in the car detected two passengers the system records it as a possible discount of 10%

[R9]: The moment the car is stopped and the sensors in the car detect no one inside, if the car has more than 50% of its maximum battery the system records it as a possible discount of 20%.

[R10]: If before 2 minutes since the moment the car has been stopped and its sensors detected no one was inside the car is plugged in a power grid and its position is within a special parking

[R11.0]: The moment the car is stopped and the sensors in the car detect no one inside, if the safe area nearest to the car is more than 3km away from it, the system records an extra fee of 30%

[R11.1]: The moment the car is stopped and the sensors in the car detect no one inside, if the car has less than 20% of its maximum battery, the system records an extra fee of 30%.

[R12]: After two minutes since the car has been stopped and its sensors detected no one was inside, the system applies all the extra fees and if there are none it applies the highest discount among the possible ones to the cost of the ride.

*COMPONENTS:* GPS\_Controller, PassengerSensorsController, BatteryLevelController, EngineController, StateWrapper, ChargesController, ChargingHistoryModel

[RA4]: If a user with a pending payment procedure tries to reserve a car, a pop-up lets the user know that he/she needs to pay for his/her last ride to be able to reserve a car and the app does not allow the user complete the reservation procedure.

*COMPONENTS:* ChargesController, ChargesHistoryModel + components for a reservation

[RA5]: If in the user's profile either the credit card or identity card expiration date has already passed, when the user tries to reserve a car a pop-up lets him/her know that the data in the user's profile need to be updated and the app prevents the reservation procedure from being completed

*COMPONENTS:* UserRegistrationController, UserModel + components for a reservation

[RA6.1]: From the home page of the app the user can access the "My profile" section

[RA6.2]: From the "My profile" section the user can use the "Edit profile" button to modify his/her credential and payment info

[RA6.3]: The system can check via the external payment service whether the payment info inserted are valid

[RA6.4]: If the inserted payment info is valid the user can save the changes by tapping the "Confirm" button.

*COMPONENTS:* UserInterface, ProfileManager, AuthenticatedRequestManager, UserRegistrationController, UserModel

[RA7]: If the user has already a reservation which is not expired yet when he/she tries to reserve a car, a pop-up lets the user know that he/she cannot reserve a car and the app does not allow the user complete the reservation procedure

*COMPONENTS:* ReservationController, ReservationModel + other components for a reservation

[RA8]: When a car is locked the system checks its GPS coordinates, and if they correspond to those of a non-safe area the last user who reserved the car is charged for a set extra fee.

*COMPONENTS:* GPSController, CarStateController, CarStateModel,

[R13.1]: Each employee has access to an application, AdminPowerEnJoy, on their phone

[R13.2]: When a car is locked the system checks its GPS coordinates, and if they correspond to those of a non-safe area all employees are notified through AdminPowerEnJoy that the car needs to be retrieved

[R13.3]: AdminPowerEnJoy allows an employee to accept a retrieval request through the "Retrieval procedure" function

[R13.4]: If an employee has already accepted a retrieval request, the retrieval request can no longer be accepted

[R13.5]: After 12 hours, if an employee has accepted a retrieval request but has not retrieved the car, the request is issued again by the system and another employee can accept it

[R13.6]: When an employee is notified of a car to retrieve, the notification contains the information necessary to set up the navigator of the company's cars to find the position of the car to retrieve

[R13.7]: AdminPowerEnJoy allows an employee to unlock any car for which he/she has accepted a retrieval request

*COMPONENTS:* UserInterface, RetrievalManager, PushMotificationManager, UnlockrequestManager, AuthenticatedRequestManager, CarStateController, CarStateModel, CarRetrievalController, RetrievalRequestModel

[R14]: When the employee ignites a car which was opened through AdminPowerEnJoy the system does not initiate any charging procedure

*COMPONENTS:* UnlockRequestManager, AuthenticatedRequestManager, CarStateController, CarStateModel, CommandDispatcher, LockingCommand

***3. EFFORT SPENT***

Together: 2h + 2.5h + 3.5h +3h +3h +4h [18h]

Reppucci: 2h + 2h + 2,5h + 3h + 2h + 2h [13,5]

Peverelli: 2h +2h +5h +2h + 2h +3h +1h [16h]

**= [47,5h]**