

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)  2.5.2. OBJECT DIAGRAMS  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***

***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***

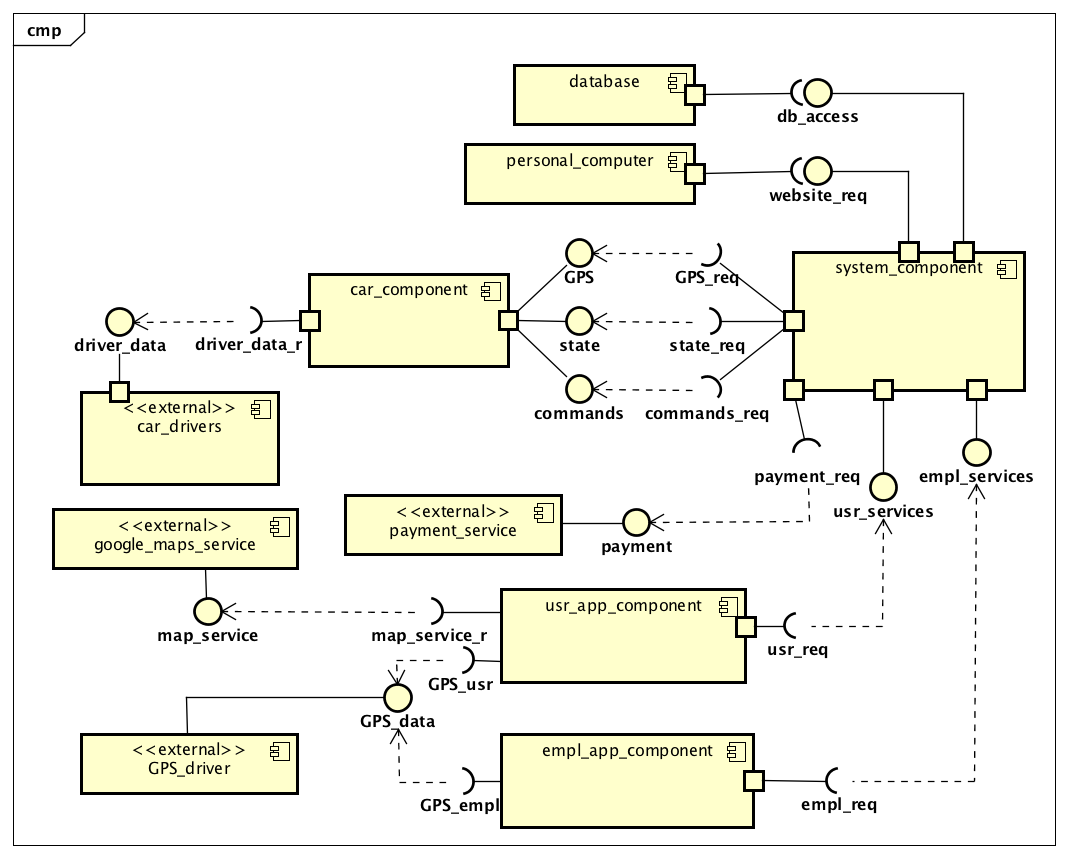
The structure of this document is meant to reflect the actual stages of the development of the architecture and design of the system, in order to illustrate not only the final result of the design phase, but also the intermediate steps of this process, and the rationale behind every major design decision. 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***

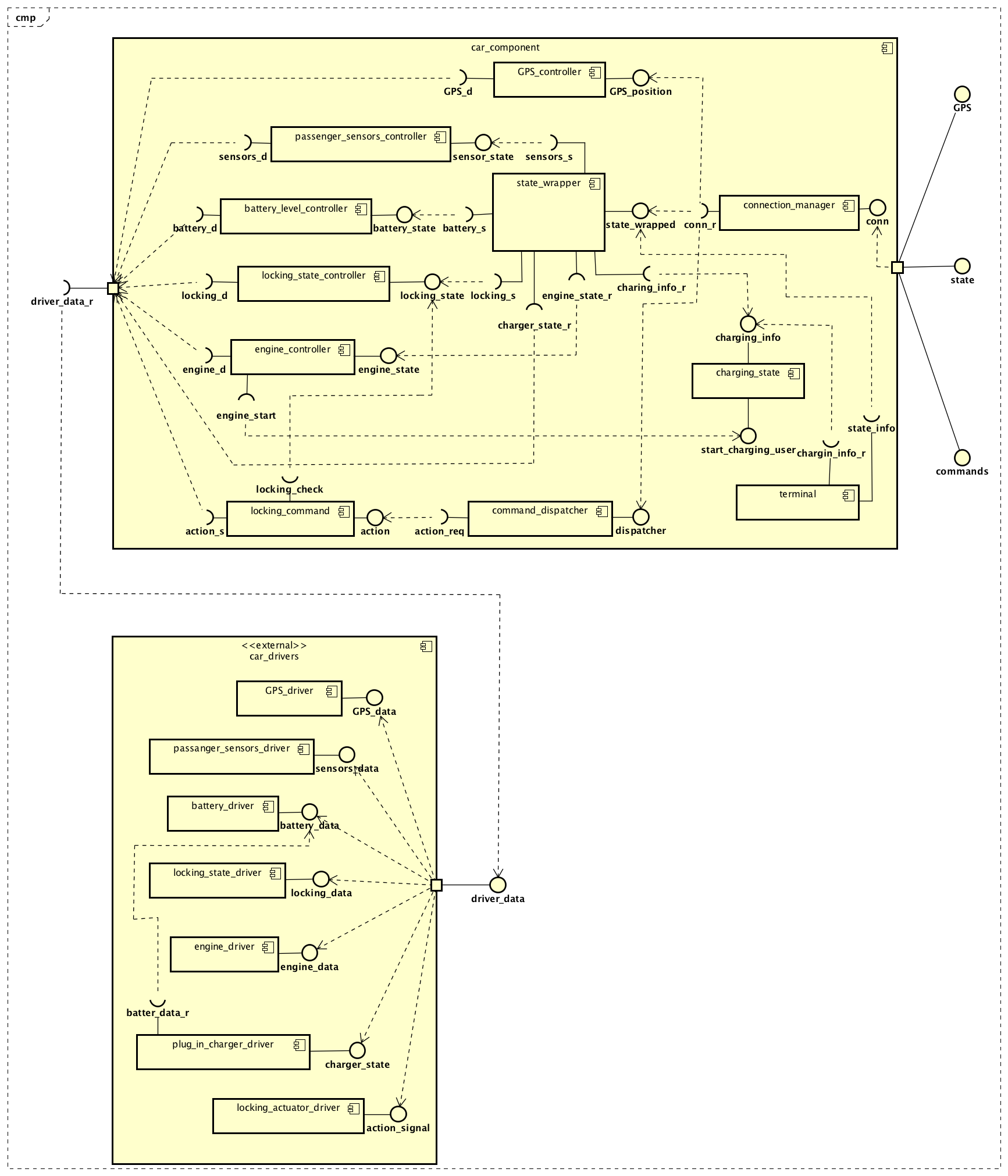
***2.2. COMPONENT VIEW***

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

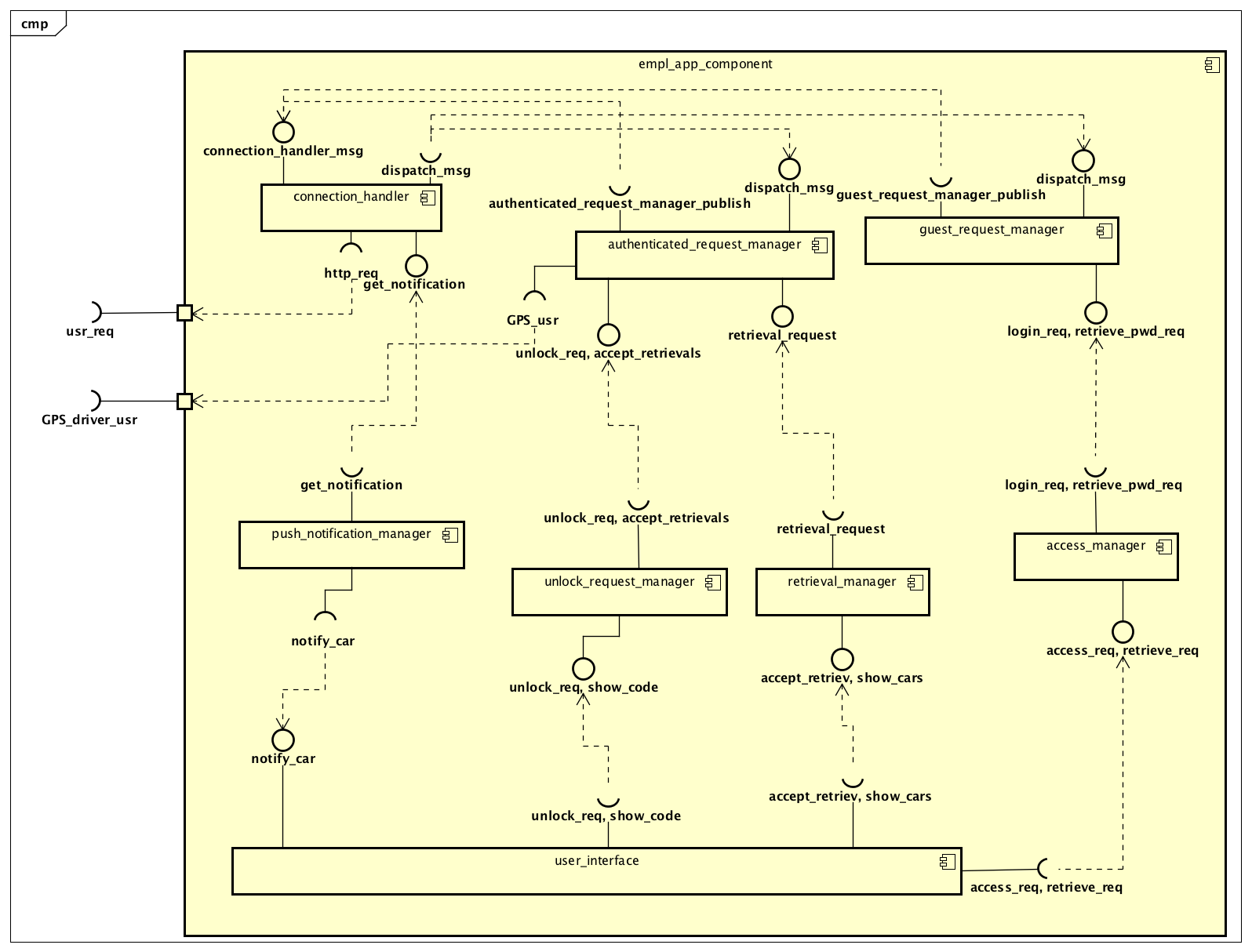
******

***2.2.2 SOFTWARE COMPONENTS VIEW***

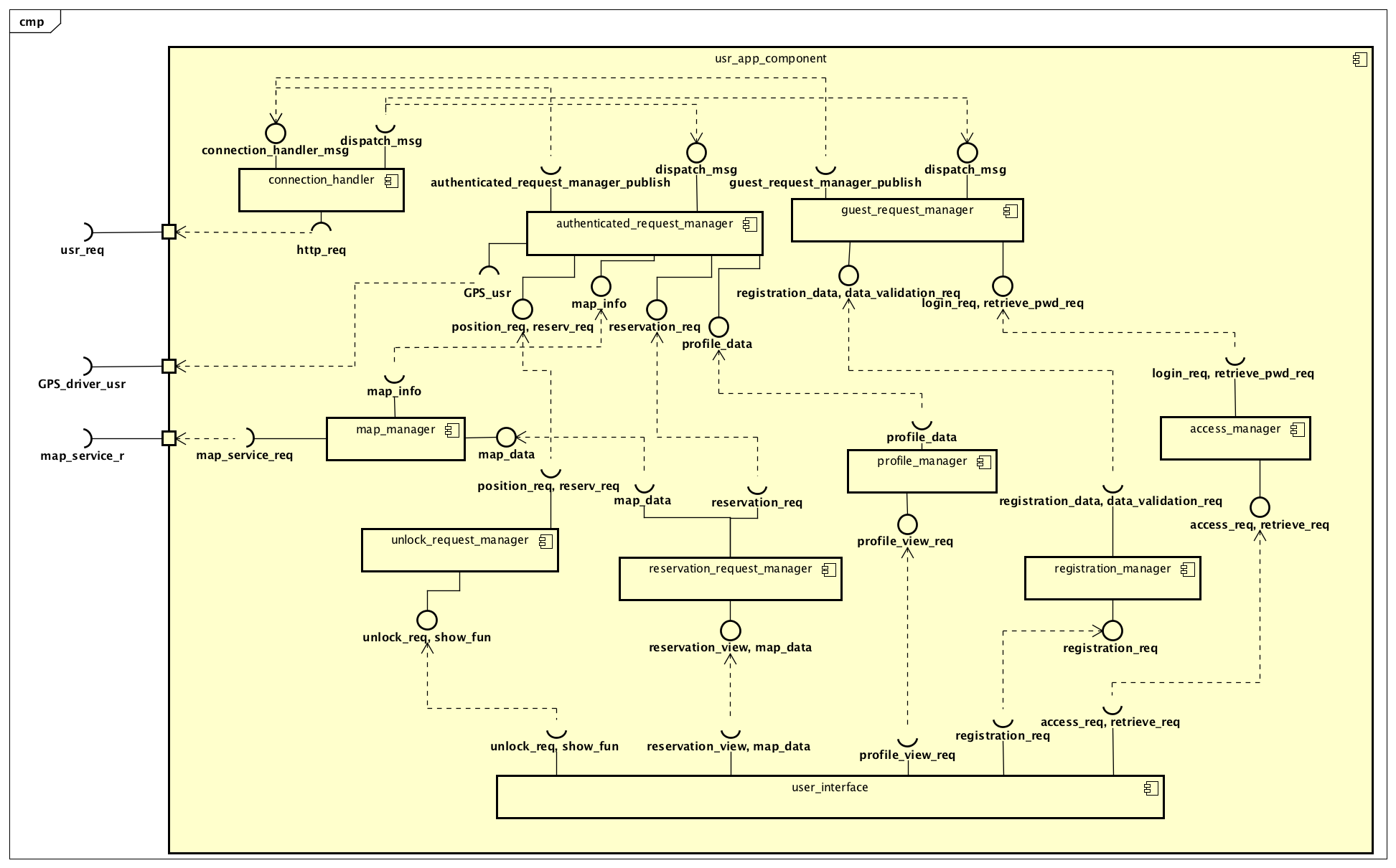
*Car*

******

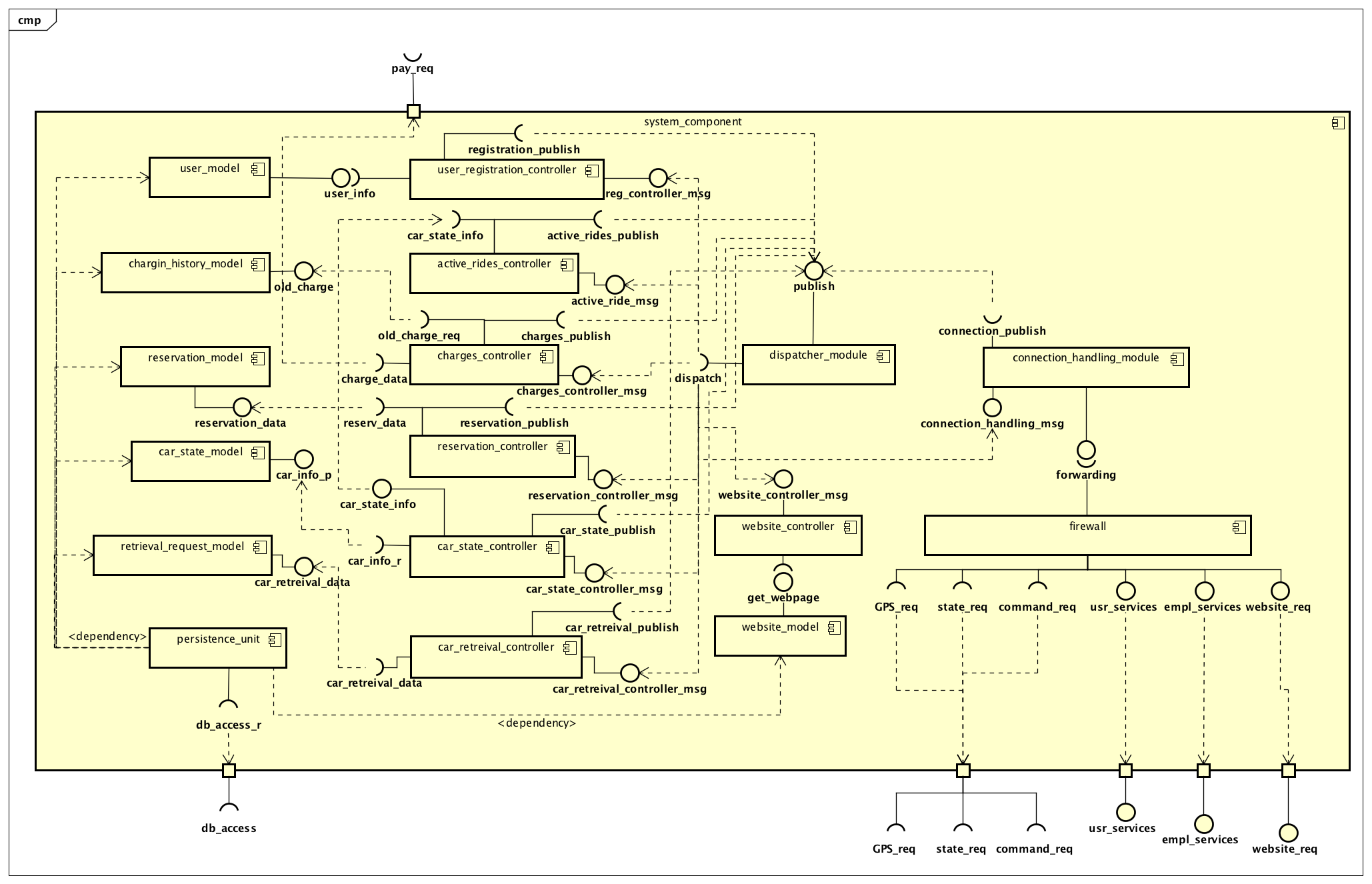
*Employee application*

******

*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.

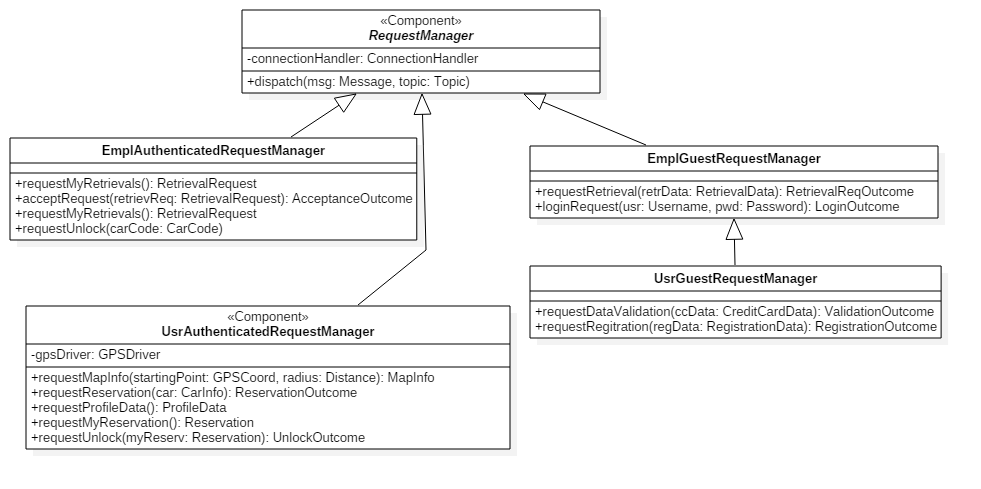
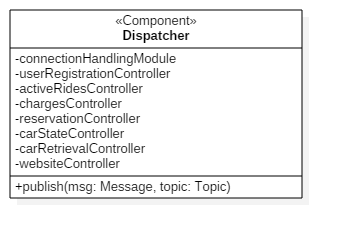
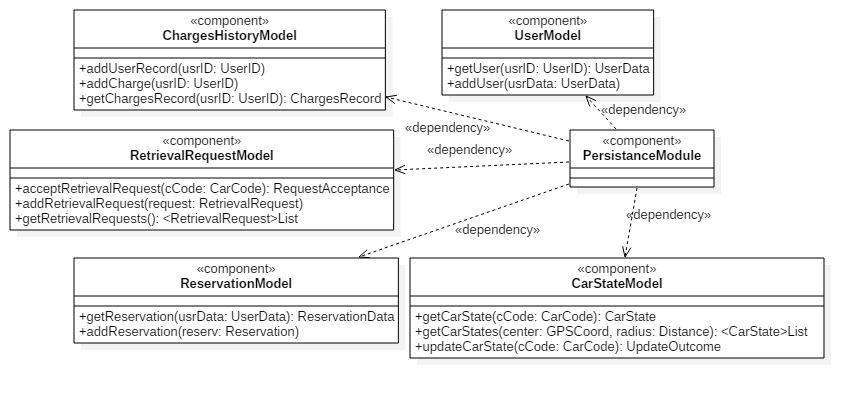
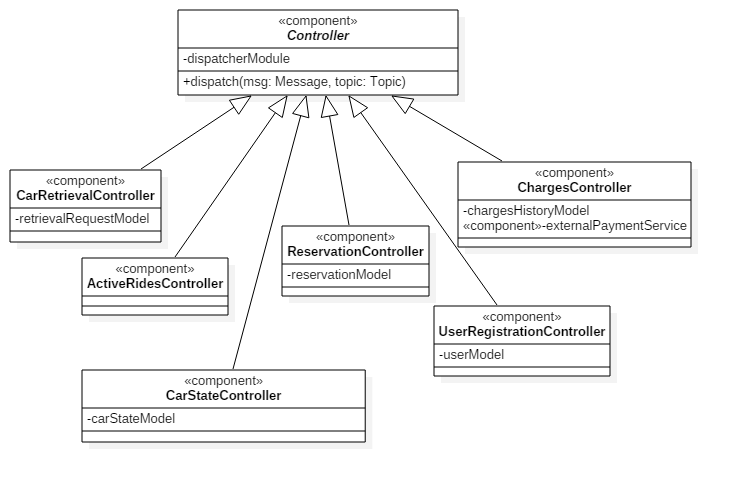
***2.3.2. SYSTEM ARCHITECTURE***

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.

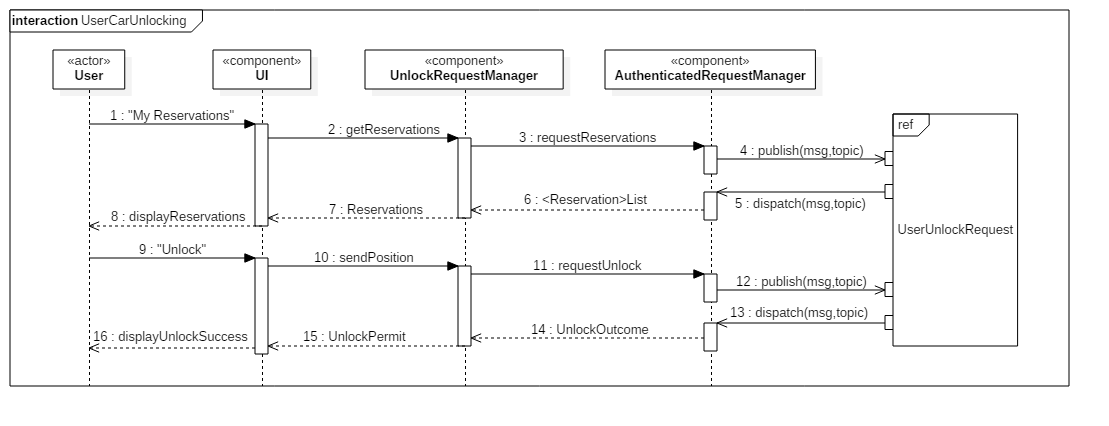
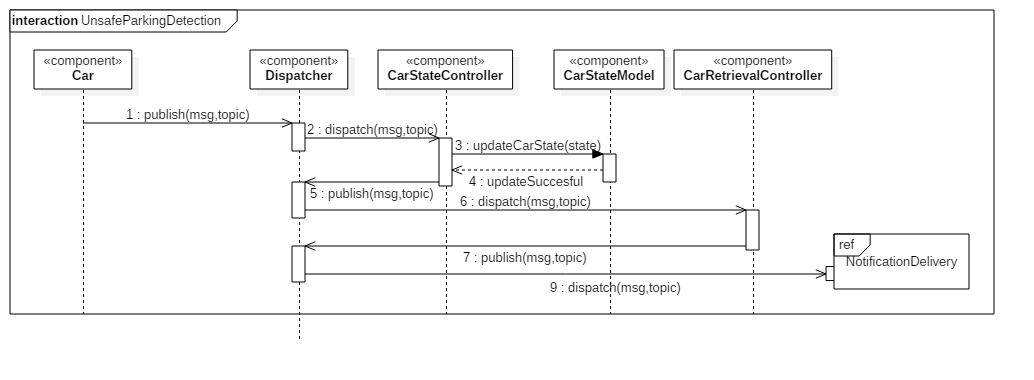
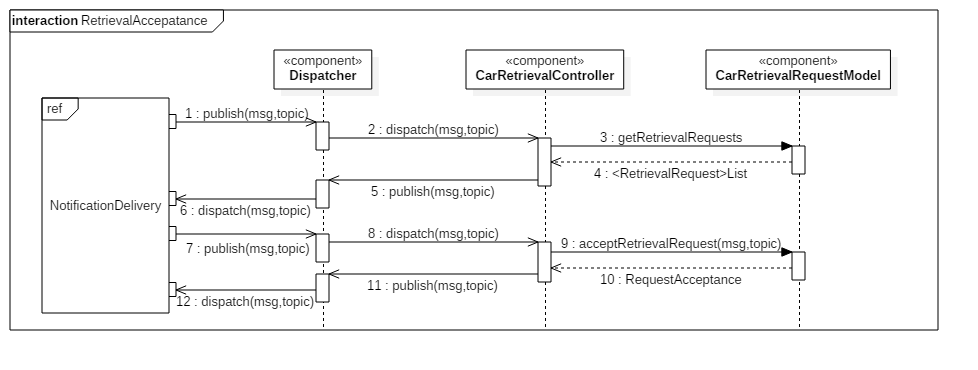
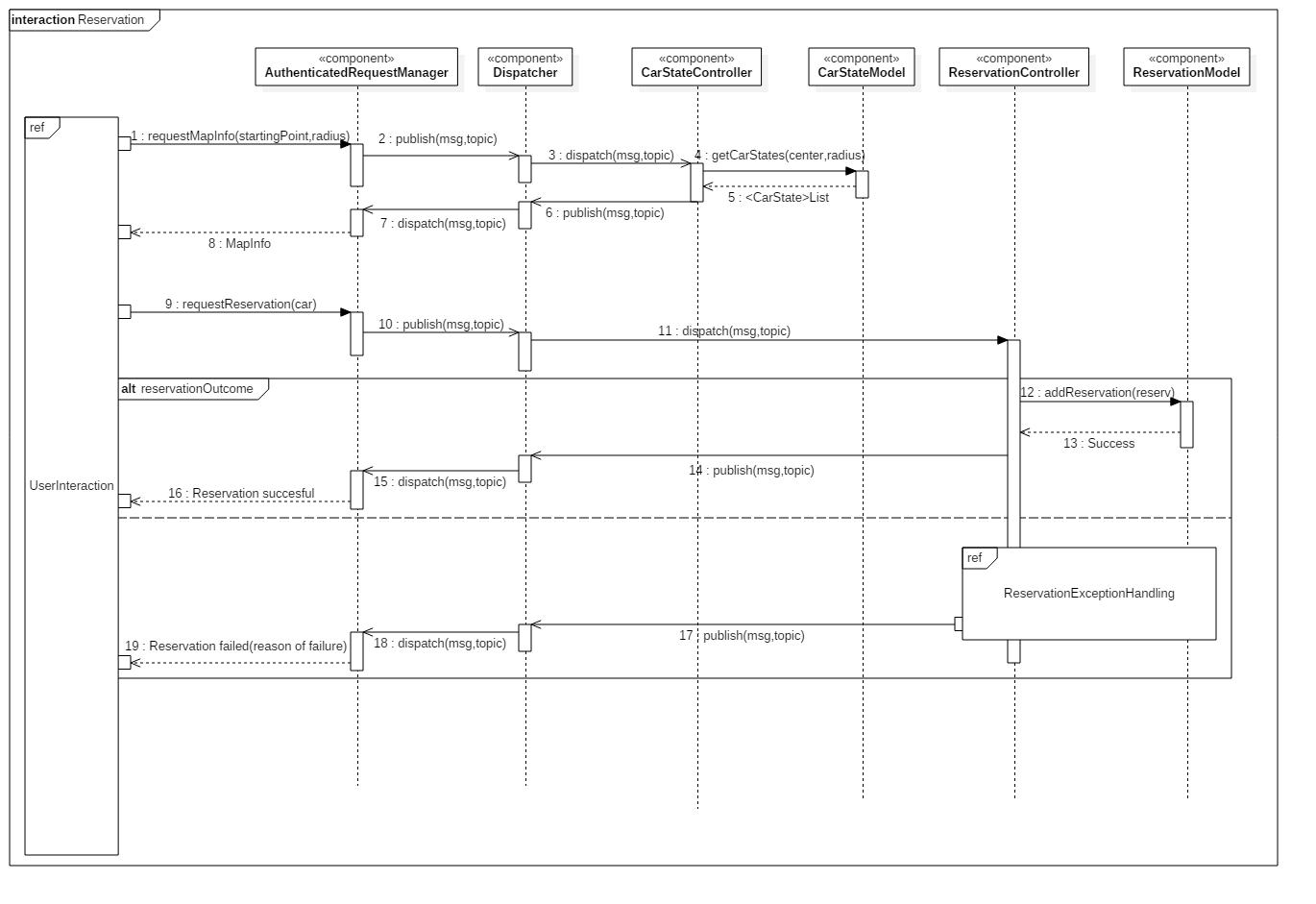
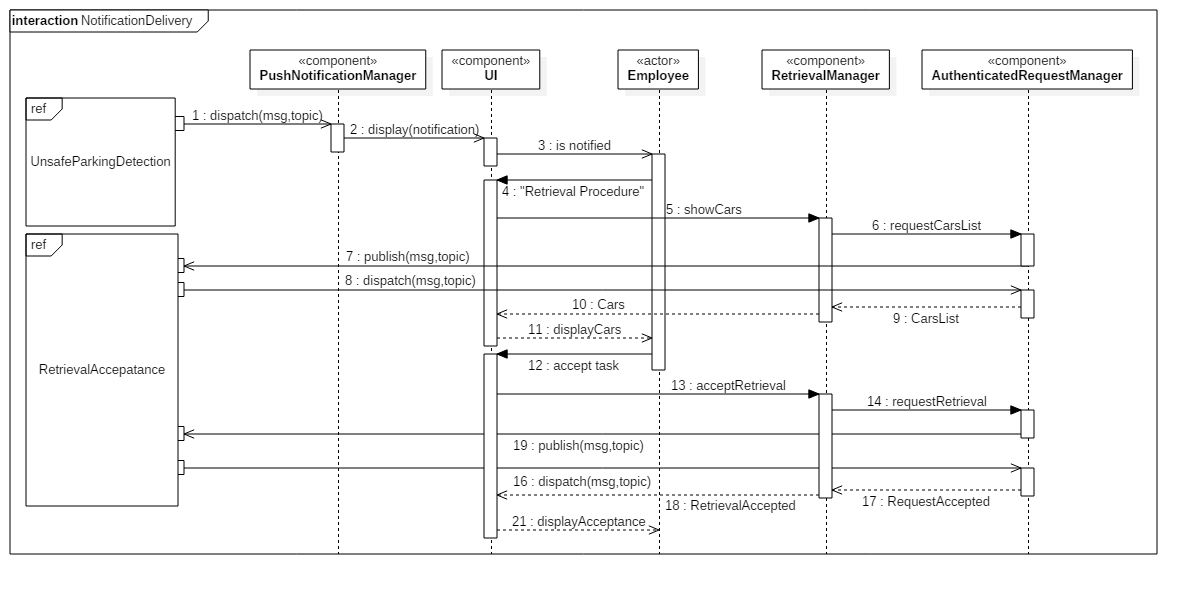
../2.%20BODY/3.%20HIGH-LEVEL%20SYSTEM%20ARCHITECTURE/ArchitectureDiagram.png

***2.4. COMPONENTS INTERACTION***

***2.4.1. COMPONENTS INTERFACES***

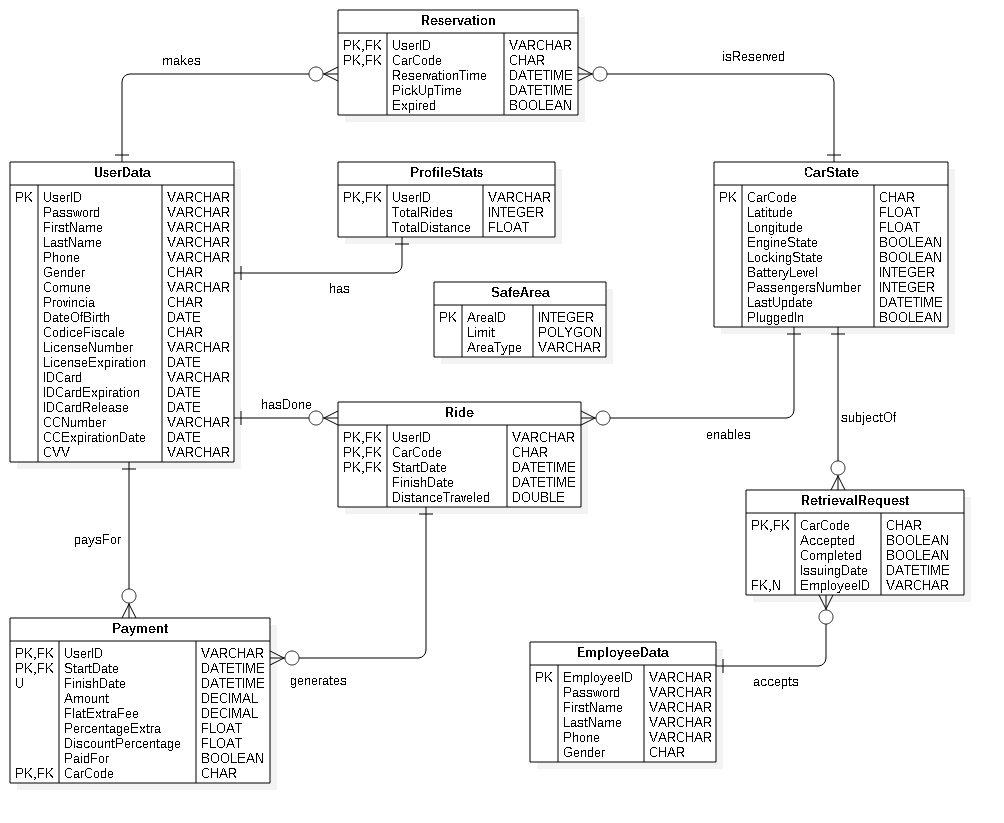


***2.4.2. RUNTIME SEQUENCE DIAGRAMS***



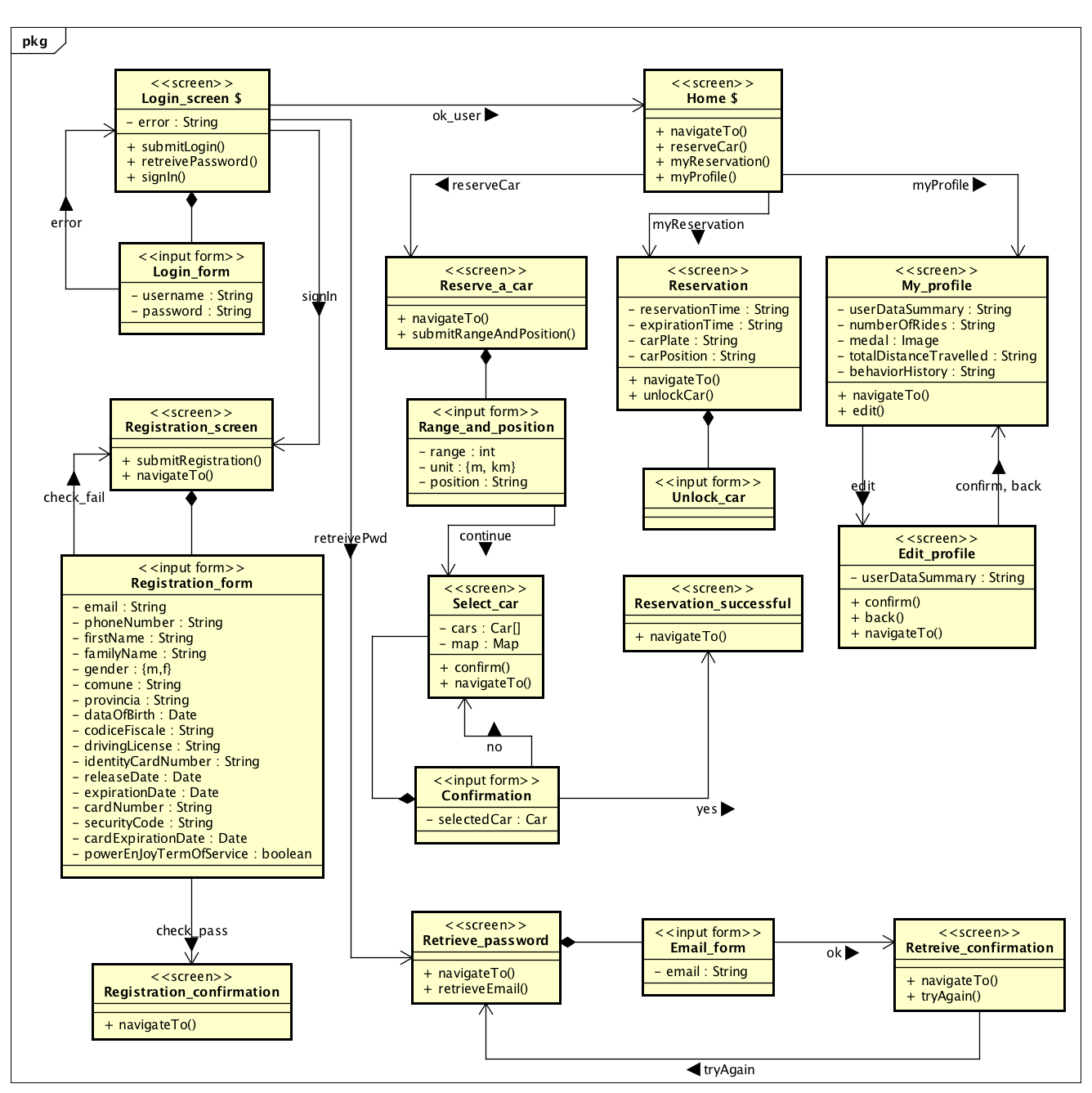
***2.5. COMPONENTS ARCHITECTURE AND PATTERNS***

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



***2.5.2. OBJECT DIAGRAMS***

***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

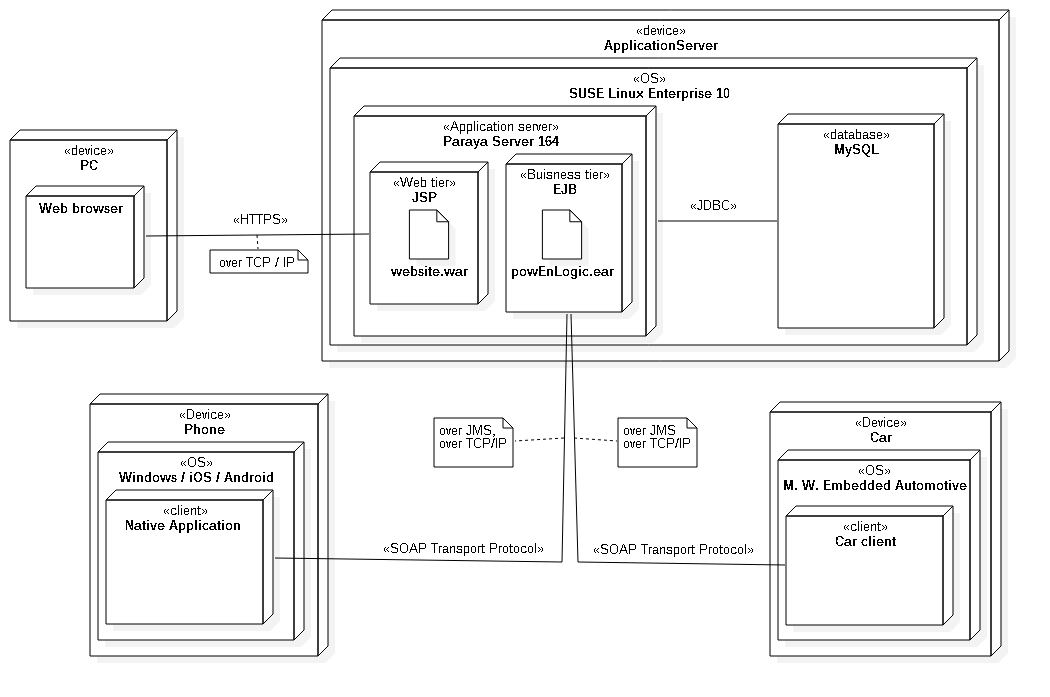
*Ides*

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***



***2.8.3 TECHNICAL ENVIRONMENT REQUIREMENTS***

***2.9. ALGORITHM DESIGN***

In this section is analyzed the procedure of reservation of a car. The procedure is written in pseudo objective oriented programming code.

/\*\*

\* The ReservationControl class controls if a reservation asked from the user is well

\* formed.

\* A reservation is well formed if:

\* 1) the user hasn't other active reservations, and

\* 2) the user hasn't pending payment, and

\* 3) the user's credit card and identity card are not expired, and

\* 4) the car is available to be reserved

\*/

**class** ReservationController implements Publish, Subscriber{

**attribute** ReservationModel reservationModel

/\*\*

\* Controls the reservation. Is the reservation is well formed, the reservation is

\* added into the reservation model.

\*/

**method** void checkAndAddReservation(Reservation reservation) {

try {

doubleReservationCheck(reservation);

lastPaymentCheck(reservation);

validityUserDataCheck(reservation);

carAvailabilityCheck(reservation);

reservationModel.addReservation(reservation);

} catch (DoubleCheckReservation or PendingPaymentException or

InvalidUserDataException or NotAvailableCarException){

throw ReservationException

}

}

/\*\*

\* Controls whether the user has not already an active reservation.

\*/

**method** void doubleReservationCheck(Reservation reservation) {

user = reservation.getUser();

reservationToCheck =

reservationModel.getNotExpiredReservation(user);

if reservationToCheck != null

throw DoubleCheckReservation

}

/\*\*

\* Controls whether the user hasn't pending payment.

\*/

**method** void lastPaymentCheck(Reservation reservation){

user = reservation.getUser();

payment = user.getLastPayment();

if payment.isPending()

throw PendingPaymentException

}

/\*\*

\* Controls whether the user's credit card and identity card are not expired.

\*/

**method** void validityUserDataCheck(Reservation reservation){

userData = reservation.getUser().getUserData();

if not userData.areValid()

throw InvalidUserDataException

}

/\*\*

\* Controls whether the selected car is actually available

\*/

**method** void carAvailabilityCheck(Reservation reservation){

carState = reservation.getCar().getCarState();

if not carState.isAvailable()

throw NotAvailableCarException

}

}

***2.10. REQUIREMENTS TRACEABILITY***

***3. EFFORT SPENT***