POLITECNICO DI MILANO
Facoltà di Ingegneria
Corso di Laurea Specialistica in Ingegneria Informatica

GreenMove: a software infrastructure to support open vehicle sharing

Relatore: Prof. Gianpaolo CUGOLA

Correlatore: Ing. Matteo ROSSI

Tesi di Laurea di:
Sante Gennaro ROTONDI
Matr. n. 724821

Anno Accademico 2011–2012

Acronyms

A list of all of the acronyms which are used in the document.

DSL	*Domain Specific Language*
ECU	*Electronic Control Unit*
EPAC	*Electric Pedal Assisted Cycles*
GEB	*Green E-Box*
GMA	*Green Move Application*
GMC	*Green Move Center*
GMVSS	*Green Move Vehicle Sharing System*
GM	*Green Move*
GPS	*Global Positioning System*
HAL	*Hardware Abstraction Layer*
MVC	*Model View Controller*
NFC	*Near Field Communication*
RFID	*Radio Frequency Identification*
SAL	*Software Abstraction Layer*
TDD	*Test Driven Development*
ZELS	*Zero Emission, Light, Small*
ZEV	*Zero Emission Vehicle*

CONTENTS

LIST OF FIGURES

— CHAPTER 1 —

INTRODUCTION

This thesis describes the Green Move project, with particular focus the work of the author on the development on two components of the former (Green Move Center and Green Move Applications).

Green Move is an innovative vehicle sharing system designed and prototyped by Politecnico di Milano, with fundings by Regione Lombardia.

The aim of the Green Move project is to solve the mobility problem discussed in chapter 2, building a system which unifies and brings forward many existing solutions to this problem.

Western countries have a common personal mobility model: vehicles are often owned and used by individuals, both for work and for personal travels. The core of the mobility problem is that, given the number of people in the world - who wish for a personal mobility model similar to the one described below - the ending of the reservoirs of fossile combustibles and the extreme need to reduce the CO_2 emissions in the next twenty years, people won't be able to perpetrate or reach the aforementioned model of mobility.

Vehicle sharing allows people to limit the possession of an own vehicle, while preserving their ability to move without using only the public transportation system.

Chapter 2 presents multiple vehicle sharing initiatives which exist in the world, stressing their characteristics and limitations w.r.t to the problem they wish to solve.

The multi business-model of Green Move is described, along with another unique feature of this system, which is the multi ownership-model. Green Move does not infact require a single entity to possess all of the vehicles employed in the car sharing system. This allows B2C, B2B and even P2P business models, with the only constraint of adhereing to a set of standardized protocols which are developed and enforced by Green Move itself.

To comply with the requirement of reducing the CO_2 emissions, Green Move chooses to employ only electric vehicles in its fleet.

Chapter 3,4 and 5 focus on the overall architecture and core components of Green Move. A system with an ambitiuous goal such as solving the mobility problem requires an accurately developed infrastructure, both on the information technology (IT) and on the other sides.

Chapter 3 focuses on the IT components needed to operate the Green Move vehicle sharing service. These include both the hardware and the software ones, ranging from vehicles to web application frameworks.

Green Move does support multiple vehicle kinds, while the vast majority of existing vehicle sharing initiatives only feature one kind (e.g. only cars, only bikes).

Each vehicle kind has to be adapted to comply with the Green Move standards and protocols. This is achieved with the installation of a physical device on each vehicle, named the Green E-Box (GEB). GEB technology and design is described in chapter 3.

A GEB is a physical device with electronic circuits developed at Politecnico di Milano and an Android board, which is a computer running the Android operating system and custom software developed for the GEB operations.

The purpose of GEBs isn't limited to allow vehicles to be shared in the GM system: they also collect data from the vehicles with its builtin sensors.

The collected data are sent to a central repository and made available for use inside the GM system, both for the system itself and third parties which have a partnership with GM. This functionality is achieved with a custom software component named T-Rex. Chapter 3 describes this component and the functionalities it provides.

The scope of *T-Rex* is not limited to data collection, it is infact a Complex Event Processing middle-ware capable of reacting to given situations detected with the observation of the stream of collected data, which are treated as events.

Green Move aims to minimize the cost of operating a vehicle sharing system, and does this by automating everything possible and removing the need of any expensive structure which may be not needed or replaced with technology.

Users need to own a smart-phone to rent vehicles of the GM fleet, since the reservation process can be made online and on mobile and no physical keys are needed to access the vehicles.

This allow GM to automatize the administrative functionalities of billing and to avoid having a pervasive human presence to operate the key delivery and return functions.

The smart-phones run an application which enables them to connect to the GM system, place reservations and manage their billing preferences.

Chapter 3 also presents the basic elements of a core component of Green Move, which is the Green Move Center (GMC) and is described accurately in chapter 4.

GMC is composed of a web application, which acts as the website which users visit to browse offers and place reservations, and the T-Rex based communication system. The web application is a software component developed with the Ruby programming language, plus minor parts in other languages required for the website (JavaScript, CSS, HTML).

The purpose of the GMC is to provide the functionality of users' registration, vehicle's reservations and the administration of the whole GM system.

GMC provides this functionalities both as a website and as web service, since it exposes a public application programming interface (API) which is used by the users' smart-phone application to communicate with the GMC.

Chapter 5 focuses on another important innovative functionality of Green Move, which is the possibility both for GM itself and its partners to push additional logic on the GEBs on demand. The cost of the software mantainance of the whole system is lowered since it has a mean of extending the code on the GEBs remotely. The extensions of functionalities on the GEBs are named *Green Move Applications* in a sort of analogy with the existing App based systems (e.g. iOS, Android).

Such a system has also a side benefit, since it allows some functionalities to be present and used only where needed, based on the context of the current user and vehicle. This is a unique feature for a vehicle sharing initiative.

Chapter 6 describes some of the technologies and techniques used to aid the development of GM.

Such a complex system requires many skills for its development, along with proper tools for testing and debugging of its components and the interactions between the latter.

Chapter 7 presents the conclusions of this work along with the proposals for the future work on the GM project.

— CHAPTER 2 —

MOBILITY SYSTEMS

THE MOBILITY PROBLEM

In western countries, personal mobility models based on an internal combustion engine are consolidated. People expect an evolution toward more comfortable, clean, technologically advanced and economic vehicles. There are two billions of people in developing countries which aim to a fast reach of the same level of mobility.

This is unlikely to happen though, since traditional fossil based combustible stocks are rapidly decreasing and there is the need to diminish the CO2 emissions in the atmosphere.

Unless a strong technological singularity happens, the solution to the mobility problem will lay upon three evolutionary lines.

- Reduction of weight and size of vehicles
- Massive reduction of internal combustion powered vehicles and the subsequent rise of electric vehicles
- Change in the mobility model, moving from the traditional single-owned model toward a new one, based on vehicle-sharing

It is also important to note that electrical grids are evolving, becoming a collector of energy from various sources (wind, solar, geothermal, carbon). Mobility should look at electrical grids as an access point to power, since it is anachronistic to look at sustainable mobility without considering the energy consumed by vehicles.

In the next years, new sustainable mobility models will arise around the world. We believe that most of them will have in common the *small, electric, shared* vehicle paradigm.

VEHICLE SHARING SYSTEMS

In the last few years many car sharing initiatives have been promoted around the world, as car sharing (and more generally vehicle sharing) has become a viable solution for the future of urban transportation [Katzev, 2003]. In this section we briefly report on some of these experiences, focusing on the underlying technological issues.

Zipcar [Zipcar, n.d.] is a popular car sharing system available in the United States and some cities of Canada, Spain and United Kingdom. Zipcar started in the early 2000's in the US. This service offers different kinds of ICE or hybrid cars to registered users, who reserve vehicles via web or via phone. Users are provided with RFID cards that are used to open cars, and the vehicle keys are physically chained to the car, together with a fuel card; users can also locally lock/unlock the car using a smartphone app, though the interaction does not occur directly with the car, which is opened remotely by the control center when an SMS is received.

A car sharing service similar to Zipcar is the Swiss Mobility Carsharing [Sharing, n.d.], started in 1997. In 2010 Mobility operated about 1200 stations in 450 locations in Switzerland. As for Zipcar, upon registration users receive RFID cards that are used to open the cars, that are reserved online or via phone. The keys are in the vehicle, but not chained to it, so doors can be locked/unlocked using the vehicle key itself. A simple interface available in the car allows users to modify on-the-fly the trip by extending or stopping the reservation. A fuel card is onboard to refill the car, if needed.

A more flexible car sharing service is the car2go project [Car2Go, n.d.], which is available in cities such as Ulm, Germany, and Austin, Texas. As Daimler AG is the promoter of the initiative, cars available to Car2go users are Smart ForTwo, equipped with an RFID card reader to unlock the car. When a user registers to the service, an RFID chip is applied to her driving license. The car has an advanced user interface, and the position of every vehicle is tracked via GPS. This allows users to leave/take the car in every free parking of the city. Moreover, the reservation is optional. Employees of the car2go service refill the cars when the fuel level (which is remotely monitored) is low.

The last decade has seen the emergence of a different kind of car sharing services, the so-called "peer-to-peer" ones. With peer-to-peer car sharing any private car owner can share her own vehicle with other users. To guarantee that private cars are not damaged/stolen when lent out, the vehicles' position must be known in real-time. Moreover, a control center keeps track of which user is using which vehicle. Therefore, shared vehicles must have features such as GPS positioning, GPRS communication with the control center, and mechanisms to identify the user. An example of a peer-to-peer car sharing service is Google's Relay Rides [RelayRides, n.d.]. To join this service each would- be lender must register her vehicle on a website. Then, a technician takes care of installing on the car a "box" containing

- a GPRS/UMTS communication module
- a GPS device
- a RFID card reader
- a fuel card

Users must register online, after which an RFID card is delivered to them. Reservation is also done trough a website. Relay Rides allows lenders to configure both the schedule and the charge for the use of their vehicles. Moreover, comments on drivers' behaviour are collected in a forum; this allows a lender to avoid the most "dangerous" users.

None of the car sharing services analysed so far focuses on electric vehicles. yélomobile [Yelòmobile, n.d.] is an example of such service, which is available in La Rochelle, France. This is a traditional car sharing system, but it operates only with ZEVs, and in particular with electric cars. As in other services users must register online, after which they are given RFID cards to unlock the cars, which are taken from recharging stations. The keys are chained to the vehicle.

More recently, the e-Vai car sharing service, which is also based on electric vehicles, has been set up in Milano. This initiative complements the railway service provided by the Ferrovie Nord group, thus it can be seen as an example of "last-mile" car sharing. Upon registration, a user is sent a card which is used to access both the railway and the car sharing services. Then, she accesses the electric vehicles as in a standard car sharing service.

Finally, the autolìb service, which is available in Paris, is substituting its ICE vehicles with ZEV ones based on the Bluecar Bollorè. These vehicles have an RFID card reader, GPS localization, GPRS/UMTS connectivity and a user interface which allows users to access value-added services. Moreover, recharging stations are "smart": they interact with the user to check her identity and to lock/unlock the cars, and also send information to the control center.

GREEN MOVE

Green Move started in 2010 as a project of Politecnico di Milano, funded by Regione Lombardia. The Green Move service combines together the most interesting characteristics (use of electric vehicles, peer-to-peer sharing, etc.) that are found separately in existing car sharing initiatives. However, it goes beyond them on many aspects, such as the mechanisms through which users interact with system and vehicles, which are now entirely key-less and smartphone-based.

Green Move tackles the mobility problem on both the technological and business model sides, aiming to put the foundation to a new urban and metropolitan mobility model. Many research groups of Politecnico di Milano are involved in Green Move, in a joint effort to provide a sound and complete solution.

Existing vehicle sharing systems can be divided in two main groups:

- First generation systems: vehicles are rented in the classical manner, directly managed by personnel of the company operating the service with respect to reservation / payment. These systems usually employ normal internal combustion vehicles and usually are found in personal mobility systems from/to airports. There are also many small examples of micro-systems which employ bicycles or scooters, usually managed by tourist villages, municipalities. An example oh these system is *Car2Go*.

- Second generation systems: these difference from the first generation systems mainly because the reservation/take/release/payment phases are completely automatized. Systems of this kind have been recently implemented on large scale and feature small and cheap vehicles (bikes). The most famous and important example is the bike-sharing system of Paris *Velib*.

These systems show that, when correctly funded and implemented, they can respond to the personal mobility problem. They are, though, too simple and limited to solve the larger problem of the urban personal mobility. Limiting characteristics are mainly these:

- Vehicle mix: both first and second generation systems usually feature only a kind of vehicle (eg. only bikes, only cars). Cars are usually internal combustion vehicles which are not thought for a specifical short range urban use. This *mono-vehicle* approach contrasts with the *pay-what-you-need* principle, resulting often sub or super sized with respect to the user needs.
- Interoperability: these systems are intrinsecally closed, in the sense that no inter-operation between different vendors exists.
- Ownership of the system: these systems are single owned by a vendor. Each system thus requires a completely dedicated funding, which lowers the chance to reach a *critical mass* both on the size of the vehicle fleet and on the points of access to the service.
- Business model: first and second generation systems have a single business model. Each system (car rental, bike sharing, etc) has a specific business model and charging mechanism (free for some users, free with a caution, *a forfeit* without reservation).

Green Move proposes an innovative system, which can be defined of *third generation*. Green Move embraces the open, multi-business, multi-owned principles of the digital networks such as the Internet, which is based on open, standardized and inter-operable protocols. The presence of a *system manager* guarantees the consistency of the project. The principle of the open protocol is declined at various levels of the system, both at the hardware and software and at the business model level.

- Vehichle mix: Green Move is a multi-vehicle system, because it features different macro classes of vehicles (cars, scooters, bikes) and different vehicle kinds in each macro class. For instance, two, three and four-wheelie are featured in the same category. The only restriction enforced by Green Move is that any vehicle must use a ZELS engine (Zero Emission, Light, Small). Vehicles of this kind can be electric bikes (EPAC: Electric Pedal Assisted Cycles), electric scooters (2,3,4-wheelie, with or without cover and thus with or without the need to wear an helmet), electric quadricycles with 2 or 4 seats. Use of ZELS vehicles helps the evolution of the concept of vehicle sharing: it lowers the pollution with respect to traditional vehicles and enables power savings, with an accurate monitoring and profiling of consumptions. This allows to define more and targeted fares for the single user, considering the effective consumption and not only the distance covered.
- Interoperability: it is the key feature of Green Move. Each vehicle can be different from another, but must adhere to a common shared protocol

which enables it to join the network. Green Move aims to enable every vendor to adapt its vehicles to the standard protocol by installing a small electronic device (the Green e-Box) and a standard docking system in the parking stations.

- Ownership of the system: Green Move is a distributed system, managed by an entity which takes care of maintenance, coordination and standardization. The system includes a set of sub-systems (vehicles and docking stations) which are owned and managed by different vendors. Those sub-systems are standardized in their interfaces so that any vehicle can be connected to any docking station. The automatized management and billing of Green Move enables a flexible way of dealing with special subsets of users (eg. a multi located company may own only a small number of docking stations near its facilities, and a few vehicles. these can be made available only for the employees of the company, but be connected to every docking station in the urban area. Conversely, the docking stations owned by the company can be used with any other vehicle of Green Move).
- Business Model: various business model can coexist in Green Move. Fixed annual pricing for all the users of a municipality, free for employees of a company, pay as you go for vehicles provided by a vehicle-rental company.. Each actor can choose his business model, contributing to Green Move with his vehicles and the installation of new docking stations. The only requirement will be the adopting of the Green Move standard protocol. This will help the system to quickly reach the critical mass.

The main goal of Green Move is to bring together the benefits of zero emissions electric vehicles and car sharing, by developing a new and innovative vehicle sharing system based on electrical, small vehicles suitable for urban use. Green Move is an electric vehicle car sharing system. Any electric vehicle can be part of the system, regardless of the ownership and the kind of vehicle (two and four wheelie vehicles are already part of the system, with three wheelie planned in the upcoming months).

Another important key feature of Green Move is the absence of a physical support structure. Users can perform anything within Green Move only having an app installed on their smartphones (currently only an Android prototype has been developed). There are no physical keys to be retrieved for renting a vehicle, nor a RFID card must be shipped to users upon registration to the system. This approach dramatically lowers the barriers for new users and at the same time allows the system to function with less fixed costs. For an electric vehicle-sharing system, a key factor for success is a good number of vehicles, so the vast majority of users will find one available when asking for a reservation. The cost of electric vehicles is high at the moment, so it would be unfeasible for a single entity to have enough funding to buy a whole fleet. The vehicles are equipped with a Green E-Box (GEB), which is the only addition to each vehicle required to enter the Green Move system. Green E-Boxes provide the vehicle internet access and a GPS sensor. GEB are physically connected to the vehicle, with various level of integration depending on the specific vehicle model, and can collect data from various sources. Typical data includes battery level, instantaneous current and voltage, speed and door status (if any). GEB are responsible of acting both as a replacement for the traditional key to access the vehicle and as an enabler of additional services for the users.

2.0.1 Green Move service description

Green Move users can play two roles in the system, which are not mutually exclusive. They can be both owners which share their vehicles or regular customers. Each vehicle is provided with a GEB, which is configured upon installation to be uniquely and securely identified and reachable in Green Move.

There are no physical offices required for reservation. Users go on the Green Move website or use the downloaded app on their smartphone to search and reserve a vehicle. There is no limit on the starting time of a reservation: a user may need a vehicle in the very next moment or in the next month.

Green Move takes care of finding the best fit, given the date and time in which the user wishes to start the rental. A ticket is electronically generated and made available for download by the user's smartphone, along with other information useful for the user to spot the vehicle in a parking lot (model, color, license plate). The smartphone acts as a key, given the vehicle has network access or got a copy of the ticket in any previous moment with network access.

The user can temporarily leave the vehicle at any time during her reservation, even if no internet connection is available (i.e. in a subterranean parking lot). The vehicle will allow her to close the doors and open them back later even in absence of network connection, leveraging a direct communication channel between the GEB and the user's smartphone via bluetooth or NFC.

When the users no longer needs the vehicle, she can park it back where she picked it and put it on charge by connecting it to a power outlet of the Green Move system. The ticket will then expire and the user will no longer be able to open the doors until the next reservation.

Green Move takes care to find a vehicle which is compatible with the user request, ensuring that the vehicle will have the following characteristics:

- enough battery to cover the distance and the duration of the travel. Green Move can compute automatically these values performing data mining and time series analysis on the user rental history, or ask the user for more details on her travel to help the system find the best solution

- proper services available, since the user might require additional services (e.g. a stroller, wheelchair or baby's chair)

— CHAPTER 3 —

GREEN MOVE INFRASTRUCTURE

Green Move operation is almost fully automatized with respect to user transactions and system monitoring. This chapter focuses on the underlying hardware and software required to operate the system. The next two chapters describe two components of Green Move which are the main topic of interest for this thesis: the Green Move Center [4] and the Green Move Applications [5].

3.1 HARDWARE INFRASTRUCTURE

This section covers all of the physical resources needed by the Green Move system, which include but are not limited to hardware and supplies owned by the Green Move entity.

3.1.1 VEHICLES

Being a vehicle sharing system, vehicles are of course the main resource. As cleared before in this document, Green Move adopts an innovative, multi owned ownership model. Vehicles can be owned by Green Move itself, a municipality, a car-renting business partner of Green Move or lended by users.

Here we cover some of the vehicles which compose the current fleet of Green Move.

- *Tazzari Zero Evo* is a two-seat electric car with a driving range of 140 km, a maximum speed of 100 km/h and a touchscreen control panel on which four different driving modes (Eco, Rain, Standard and Sport) can be selected. The lithium-ion battery pack requires about 9 hours for a full charge (0-100). This vehicle is suitable for urban mobility and/or short-range interurban trips.
- *Estrima Birò* is a two-seat electric vehicle with a maximum speed of 45 km/h and a range of about 50 km. The Pb-Gel battery pack takes about 9 hours to be fully charged. Its extremely compact size makes it suitable for urban (or low-speed suburban) streets.
- *Piaggio Liberty e-mail* is an electric scooter with a top speed of 45 km/h and a range of 70 km. The lithium battery pack requires about 4 hours for a full charge. The scooter is suitable for city driving, but, unlike the other two vehicles, it is not for all weather conditions.

3.1.2 POWER OUTLETS

Power outlets are, with vehicles, a critical resource for the success of this vehicle sharing initiative. The number and the dislocation of power outlets, along with the time needed to charge a vehicle, are a key factor for Green Move.

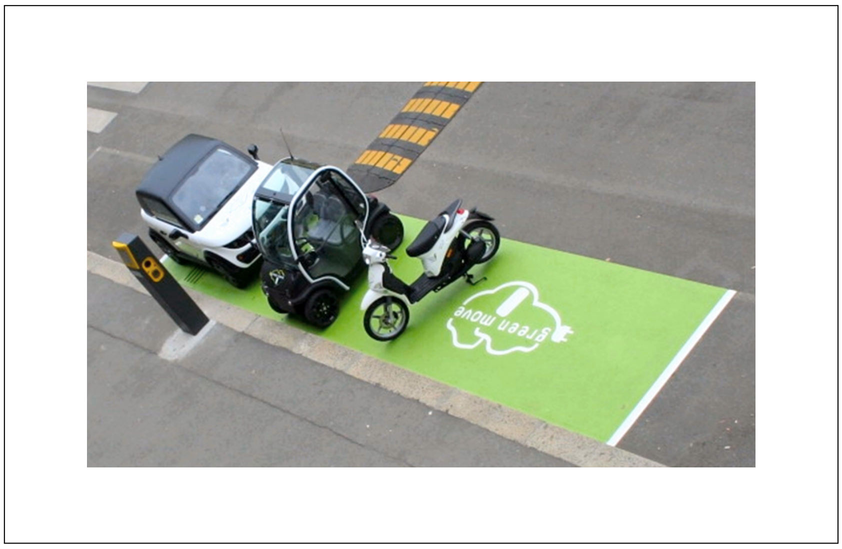


FIGURE 3.1: Tazzari Zero (left), Estrima Birò and Piaggio Liberty e-mail. A power outlet of A2A is visible next to the Tazzari Zero.

In the first phase of the project, a power outlet of A2A was installed in front of DEI. It has a screen, a RFID card reader, some physical buttons and two outlets. The outlets are of two different kinds: MENNEKES and a SCAME. MENNEKES provides two different outputs: a single-phase and a triple-phase one. SCAME provides a single-phase output. Both the single-phase outputs are of 3,5 KiloWatt (16 Ampere, 220 Volt), while the triple-phase is of 42 KiloWatt (64 Ampere, 380 Volt). The italian law admits only the use of the single-phase for recharging vehicles in the public space, which is significantly slower. The use of the SCAME or MENNEKES outlet depends on the vehicle to be charged. There can be communication between the vehicle and the outlet if this is supported on the vehicle. Information exchanged helps the power grid to handle consumption peaks, and allows the Green E-Box on the other side to know that the vehicle is charging. This information is very useful to forecast vehicle range and availability for upcoming reservations.

3.1.3 Green E-Boxes

Each vehicle is equipped with a Green E-Box (GEB). The GEB is the device which allows each vehicle to interact with the GM system: it is composed of an embedded board and an Android board. The main tasks performed by the GEB are the acquisition of vehicle signals and the handling of the connection both with the GMC and the user's smartphone. The acquisition of the signals is carried out by the embedded board which is directly connected to the vehicle ECU.

Afterwards, the acquired data are processed by the Android board and sent to the GMC. To correctly manage this complex system and meet the needed requirements of transparency and availability, abstraction mechanisms

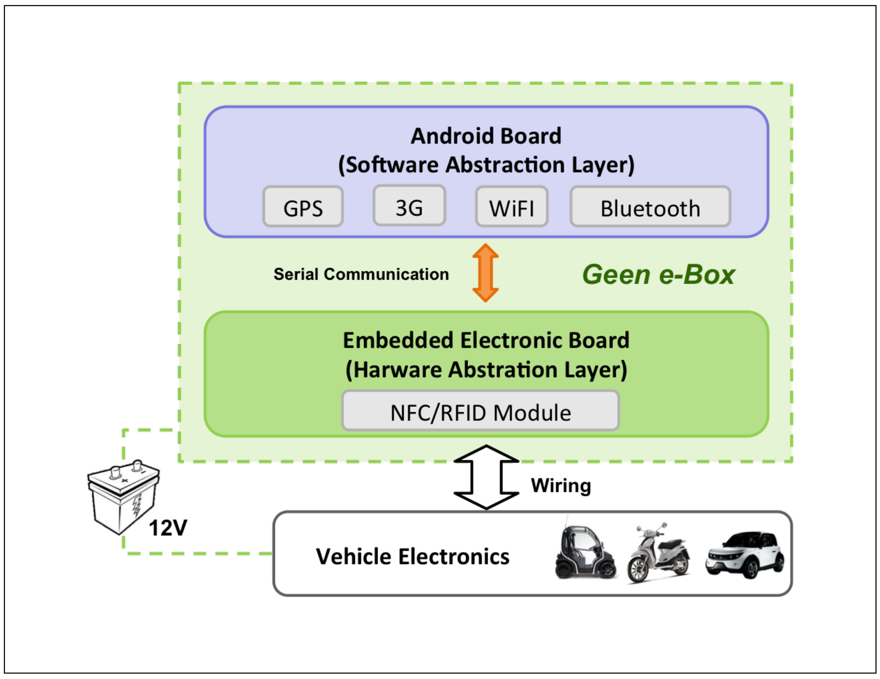


FIGURE 3.2: Green E-Box architecture.

must be implemented, which allow the seamless use of technologically different vehicles, characterized by different available signals, different onboard networks, a different split between digital and analog signals and so on.

In order to have a constant monitoring of each vehicle, even when turned off and not in use, the GEB is directly connected to the permanent 12V line of the vehicle. The GEB is then wired to the vehicle, it communicates with the GMC via a 3G channel and with the users' smartphone via Bluetooth or NFC technology.

GM users employ their smartphones to interact with the GM system and retrieve electronic keys, which are necessary to take possession of vehicles, open/close their doors, enable the drive and take advantage of the extra features offered by additional services.

The primary goal of the GEB is to ease the management of the heterogeneous fleet of vehicles by permitting a vehicle- independent communication among Green Move elements. It is composed of a low-level embedded board and a high-level Android board.

The embedded board acts as a Hardware Abstraction Layer (HAL) and it is designed to abstract the vehicle-specific details, thus providing a general communication protocol to the high-level layers built on top of it. To achieve this, the embedded electronic board is connected to the vehicle's CAN-bus to retrieve data directly from the vehicle ECU and provides several analog and digital input/output channels so that the GEB can be installed on a large variety of heterogeneous vehicles, even those without an ECU (e.g., Tazzari and Birò).

A microcontroller handles each signal, acquiring the vehicle data at a constant rate and, since the set of available signals is strongly vehicle-dependent, it collects them into standardized packets so that they can be easily transmitted to the high-level Android board. The vehicle signals are clustered into 6 categories: battery, doors, speed, faults, commands and others. Each signal available on the vehicle must belong to one of the previous category. For instance, the actual provided current, the state of charge of the battery, and the battery's state (charging or not) belong to the battery group.

The Android board provides the Software Abstraction Layer (SAL) which receives (in a vehicle-independent way) the data from the low-level board, stores them into a suitable data structure, and exposes a standardized interface that allows other GM applications residing on the GEB, to easily access the vehicle information.

The GEB decouples the high-level fleet management functionalities from those, implemented in the vehicle ECU, related to the control of the vehicle motion, thus establishing the separation and non- interference of the former with respect to to the latter, which guarantees the necessary safety requirements.

3.1.4 Servers running the Green Move Center

Green Move requires servers connected to the Internet to run the Green Move Center. The size and number of these servers can largely vary depending on the number of users and GM partners accessing the service, thus can't be determined upfront.

In an effort to build a system which is *green* even in its own insfrastructure, a cloud architecture has been chosen as the base for building the Green Move hardware infrastructure. This means that Green Move does not need its own data center, but can be deployed in the cloud and scaled by adding hardware only when needed. Of course, hardware resources can be freed when not needed by Green Move.

A level of abstraction has been added on top of the hardware resources by leveraging the virtualization techniques available in the market. The open source virtualization software Virtual Box [VirtualBox, n.d.] has been adopted. Any number of virtual machines can be run on top of Virtual Box. The Green Move Center has been designed to run on any cloud with minimum adaptations.

3.1.5 User Smartphones

Although not owned by Green Move, its crucial to note that it would be impossibile to operate the GMVSS without users' smartphones. They are essential in the interaction between the user and the vehicle. They run the software which make them act as the key for opening/closing the vehicles' doors (where available) and to enable the drive. Currently, for the sake of openness in the initial stage of the GM project, only smartphones running the Android operating system are supported. Also, the minimum required version of Android is 2.1.

Support for Apple smartphones, along with BlackBerry and Windows Phone is planned for the near future.

3.2 Software Infrastructure

This section covers the different software running in the GM system, except for the Green Move Center and the Green Move Applications functionalities which are separately treated in the following chapters.

3.2.1 Databases

Dealing with large amounts of data, Green Move needs to store them appropriately. Also, these data is different in kind and purpose, and thus require different software approaches for an efficient storage, access and use.

PostgreSQL

PostgreSQL is an object-relational database management system (ORDBMS) available for many platforms. It is released under the PostgreSQL License, which is an MIT-style license, and is thus free and open source software. PostgreSQL is developed by the PostgreSQL Global Development Group, consisting of a handful of volunteers employed and supervised by companies such as Red Hat and EnterpriseDB. It implements the majority of the SQL:2008 standard, is ACID-compliant, is fully transactional (including all DDL statements), has extensible data types, operators, index methods, functions, aggregates, procedural languages, and has a large number of extensions written by third parties.

The PostgreSQL database management system has been chosen as the database to store the administrative and operative information of Green Move. This information includes the users login information, the vehicles details and the reservations in the system, along with various other data and configurations needed to operate the vehicle-sharing service.

NoSQL Databases

Log data collected through the system does not require a structured and transactional database. This, along with the performance benchmarks of NoSQL Databases, lead to the choice to use this kind of technology to store the logs.

Part of the logs are the positions of the vehicles, which are collected like any other information in the GM infrastructure.

Currently, the evaluation of various NoSQL database is going on in the GM project.

3.2.2 Android

Android is a popular operating system for smartphones, whose development started by Android Inc. in 2003. Google acquired Android Inc. in 2005 and in 2007 constituted the Open Handset Alliance with other companies and telcos.

Android differs from iOS and Blackberry OS because it is open source software. This openness made Android the natural choice for the development of the initial prototype of the users' smartphone app. Also, Android had already been chosen as the platform for the development of the Green E-Box software stack.

Android provides both a Native Development Kit (NDK) and a Software Development Kit (SDK) for the Java language. The software running on GEBs is written in Java and so is the application on the users' smartphone.

3.2.3 Complex Event Processing Engine

A huge amount of data is generated, gathered, treated and collected in Green Move. Much of this data can been seen as *event* data, since it is strictly connected with the time, position, and *context* in which it is generated. Events can be even related one with another, since many events of a certain kind in the same time and area may be of interest to the system (i.e. many vehicles rapidly stopping on the same street might mean that an accident occurred).

Many events are generated in proximity of the user, especially when she is in the vehicle, or by the vehicle itself (position, charge level, hardware faults). With a fleet of many vehichles circulating in the system, there is the need to have a system capable of handling many events per second and to process them to build new events out of a set of rules.

This component must be a high-performance, configurable service reachable by all of the actors involved in the event generation and consumption in the GM system.

T-Rex

T-Rex is a CEP middleware developed at DEI - Politecnico di Milano which has the requirements outlined above in terms of performance and configurability.

T-Rex uses the TESLA[Cugola & Margara, 2010] language, an event definition language that provides a high degree of expressiveness to users, while keeping a simple and easy to use syntax. TESLA has also been developed at DEI - Politecnico di Milano with the precise goal of providing a simple yet powerful way to express event and rules for T-Rex.

T-Rex runs on top of the GM infrastructure. Any GEB and the Green Move Center can estabilish a connection to the T-Rex server, and so can any GM partner which is interested in sending/receiving events. At the time of writing, users' smartphones do not interact directly with the T-Rex middleware.

T-Rex Server is a software layer built around T-Rex to provide a set of API to clients. It uses the TCP/IP protocol to obtain an acknowledged, resilient communication channel with clients. The protocol used to communicate with

T-Rex Server has been developed to be simple and fast, given that it can be rewritten with little effort without changing dramatically client applications (only a the T-Rex client library must be substituted, API remains the same).

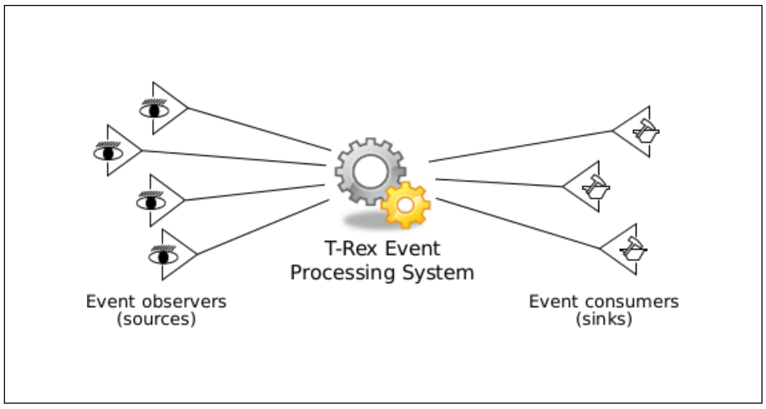


FIGURE 3.3: T-Rex Complex Event Processing

T-Rex Server exchanges packets with its client, which are summarized here:

- Ping Packet: an empty packet which is periodically sent from T-Rex Server to each connected client. A client must reply with a Ping Packet or will be considered crashed/offline, its subscriptions will be removed and the TCP connection terminated.
- Publication Packet: it is the *event* packet. It contains informations as a key-value list, along with a packet type id which is useful to better separate events based on their purpose.
- Subscription Packet: a packet carrying the information useful to perform a subscription. It includes the packet type id and constraints on the event content.
- Unsubscription Packet: a packet suitable for cancelling subscriptions.
- Rule Packet: rules for T-Rex can be dynamically added at run time by sending a Rule Packet.

T-Rex Server adopts a content based publish-subscribe paradigm: any client can subscribe for events and publish events. Subscriptions are based on the content of the event (i.e. certain attribute with given value)

T-Rex Server startup process can be summarized in this steps:

- The rules defined within the T-Rex Server source files are loaded and added to the T-Rex Engine.
- A log file is initialized.

- The internal state of T-Rex is set to a clean state, since it has no memory of any previous interaction with the GM system.
- T-Rex Server opens a listening TCP socket on port 50254 and starts listening for event publications and subscriptions.

Interaction with T-Rex in GM is performed by using its APIs. A C++ and a Java client library have been developed for this purpose.

— CHAPTER 4 — GREEN MOVE CENTER

The Green Move Center (GMC) is a key component of the Green Move architecture. It provides the following functionalities to Green Move:

- Users' registration and login: users can register to the GMC simply filling a form on the GMC website.
- Vehicle reservations: users login to the GMC and perform a research. The GMC is in charge of finding a suitable solution.
- Broker for Green Move Applications: GMAs are uploaded on the GMC and distributed to Green E-Boxes.
- Secure transmission of the ticket: the electronic ticket for a reservation is generated by the GMC and then transmitted encrypted to both the vehicle and the user's smart-phone.
- API for the smart-phones: the GMC provides the same registration and reservation functionalities via an HTTP API.
- Data collected from the vehicles is stored in the GMC database

Such a complex system is composed of various pieces. In this chapter we focus on the web-application which provides the website and the APIs, along with the support for the Green Move Applications.

Figure 4.1 shows the interactions of the three key actors in the GM system: users, vehicles (Green E-Boxes) and the GMC.

In the following sections, we analyse interactions of every actor of Green Move with the GMC and present the choices made in the implementation of the latter.

4.1 OVERVIEW OF USERS' INTERACTION WITH THE GMC

Users can sign up to Green Move by visiting the GMC website. Currently the user's credentials are the email used for the registration and the chosen password but two factor authentication, including sending a confirmation code via SMS to the user, is easily implementable in the system.

Users have a list of their reservations, both the past and the current ones. Reservations must be confirmed, since unconfirmed reservations act as a lock on a certain vehicle: after 15 minutes, the GMC purges these reservations to maximize to number of available vehicles[1]. Cleaning unconfirmed reservations also helps prevent a denial of service attack, in which numerous searches are performed and no confirmation (and payment) is done.

[1]The removal of unconfirmed reservations is scheduled as a cron task on the server on which the GMC currently runs.

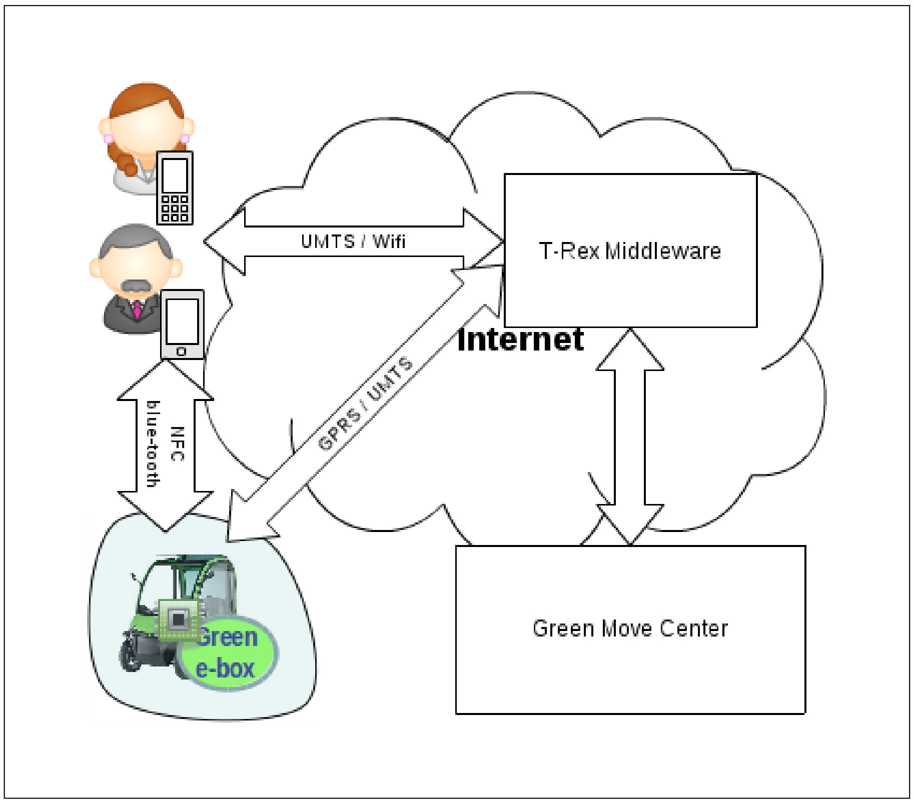


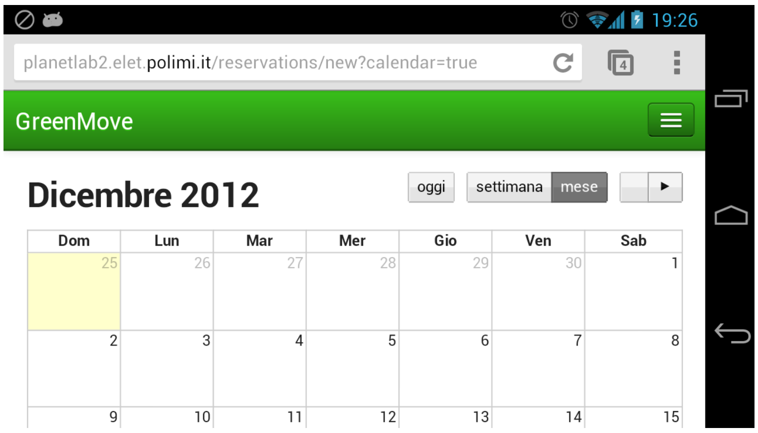
FIGURE 4.1: An overview of the interactions between the GMC, users and vehicles.

Users search for vehicles with a calendar based view, as shown in figure 4.2 and figure 4.3. The user first chooses the day in which she wants to start the rental, the model of vehicle she prefers and optionally any required additional services (eg. baby seat, assistance). Search results are presented to her, which is then invited to confirm one of the proposals and input an estimate of the time when she will no longer need the vehicle. This time may be extended later, if it does not conflict with other users' reservations for the same vehicle.

Users use their smart-phones to search for free vehicles or just to retrieve the electronic ticket. The former is possible both visiting the Green Move Center site with the smart-phone, as shown in figure 4.2 or using the Green Move smart-phone app, while the latter is available only with a running GM client application. Users are required to register and login in the GMC to perform any operation, such as reserving vehicles. The same applies for the smart-phone application, which stores securely the user's credentials using the means provided by the Android operating system.

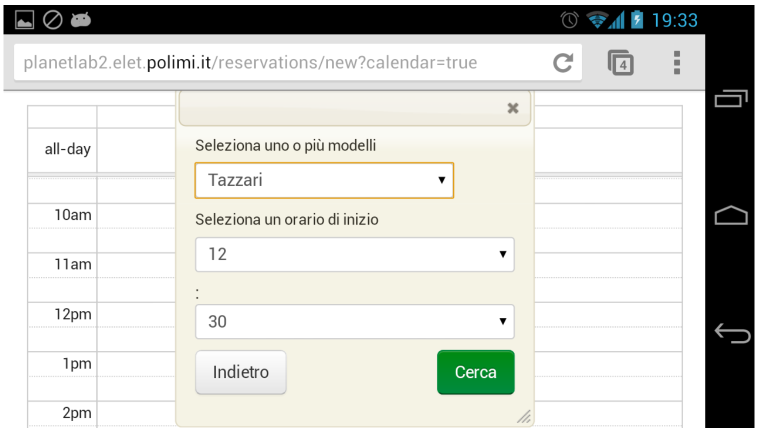
This means that any other action apart from registration requires a login on the website or the insertion of the login credentials in the application running on the user's smart-phone.

The download of the ticket is handled on user's demand, although the support for pushing the ticket directly from the GMC to the user's smart-



Dom	Lun	Mar	Mer	Gio	Ven	Sab
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15

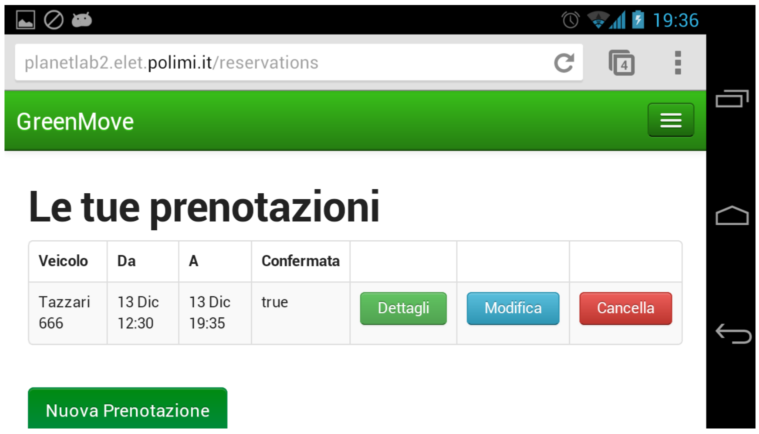
FIGURE 4.2: Calendar view when searching for a vehicle on the GMC, seen from the Google Chrome browser on Android



Seleziona uno o più modelli: Tazzari

Seleziona un orario di inizio: 12 : 30

FIGURE 4.3: Vehicle model and date selection when searching for a vehicle on the GMC, seen from the Google Chrome browser on Android



Le tue prenotazioni

Veicolo	Da	A	Confermata			
Tazzari 666	13 Dic 12:30	13 Dic 19:35	true	Dettagli	Modifica	Cancella

Nuova Prenotazione

FIGURE 4.4: List of the reservations of the current user on the GMC, seen from the Google Chrome browser on Android

phone has already been experimented successfully.

When the user tries to open the doors of a vehicle, the latter downloads its own list of valid reservations from the GMC and uses the ticket information provided with each valid reservation to authenticate the user and allow commands on a direct communication via blue-tooth or NFC.

4.2 OVERVIEW OF VEHICLES' INTERACTION WITH THE GMC

A vehicle is registered on the GMC after the installation of its Green E-Box. Vehicle registration requires the following data to be available and inserted:

- Vehicle license plate
- MAC address of the blue-tooth module of the green e-box (which is found on a label applied to the green e-box)
- Vehicle model
- Cryptographic public key for the vehicle. This key is generated along with the private key and made available by the green e-box or can be pre-generated[2]

When a user attempts to pick up a vehicle from its parking position, the vehicle downloads a list of its confirmed reservations from the GMC. Each

[2]The key must be a RSA key with 4096 bit of entropy.

reservation contains the relative ticket. The vehicle then uses the current time to determine the active ticket and attempts to establish a secure connection with the user's smart-phone, using a shared secret which is contained in the ticket (the session-key).

When the vehicle is temporary parked, no connection with the GMC is necessary. When the user leaves the vehicle back in its parking position, the vehicle communicates again with the GMC to mark the end of the reservation.

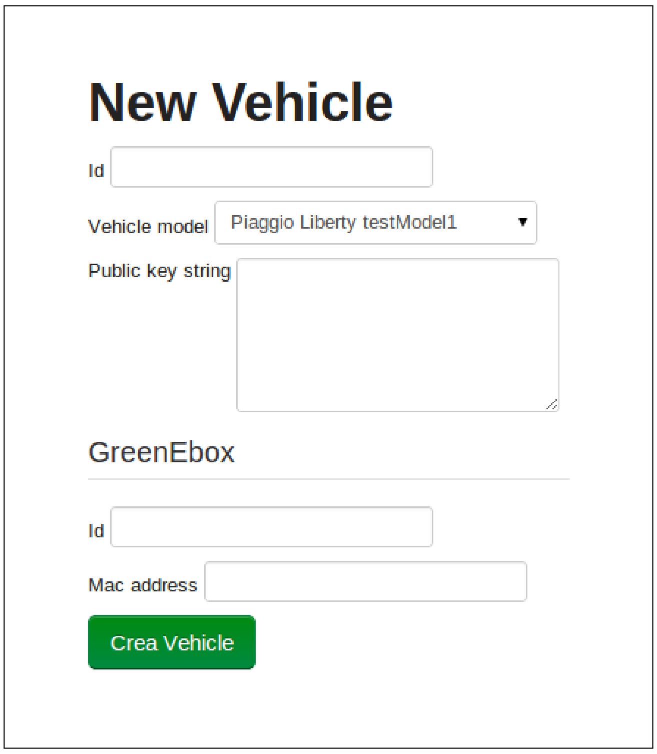
During the rental, sensors collect data on the vehicle and the Green E-Box leverages the T-Rex middle-ware to transmit these information to the GMC, which logs and stores them in the database[3].

4.2.1 Overviews of admins' interaction with the GMC

The GMC also provides administration functionalities. These include adding (see Figure 4.5), editing and removing users and vehicles, monitoring vehicles' position (see Figure 4.6) and faults.

Green Move Applications, treated in the next chapter, are also uploaded to Green E-Boxes via the Green Move Center.

Figure 4.5 shows the form prompted to an admin when registering a new vehicle on the GMC.



New Vehicle

Id	
Vehicle model	Piaggio Liberty testModel1
Public key string	

GreenEbox

Id	
Mac address	

FIGURE 4.5: Registering a new vehicle on the GMC.

[3]The NoSQL database for storing some of the collected data has not yet been chosen.

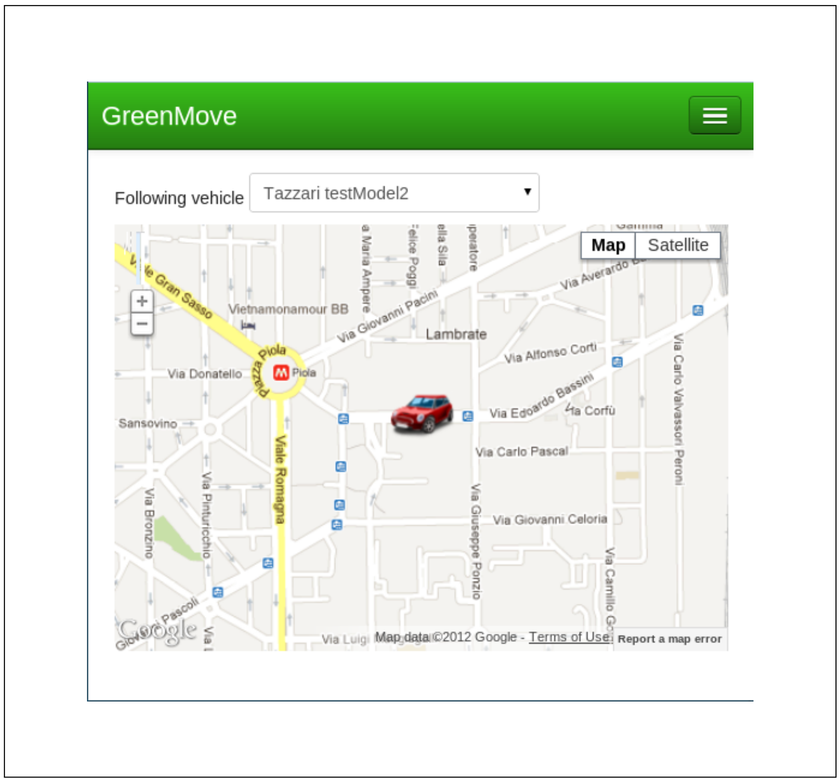


FIGURE 4.6: Position of a vehicle is showed on a map in the admin section of the GMC.

4.3 GREEN MOVE CENTER ARCHITECTURE

Green Move Center has been designed to be modular, ready for deployment on a cloud platform, easy to change, extend and fast to develop.

The sum of this characteristics, for a modern web application as the GMC, is difficult to achieve all at the same time. A number of choices had to been performed to develop the GMC, detailed in the next sections of this chapter.

The GMC has been developed with cloud computing technologies in mind, so its architecture reflects the flexibility required to run in these kind of environments. Figure 4.7 shows the network services architecture of the GMC. The components represented are not tied to a particular physical structure or software implementation, in effect they could be all running on the same physical or virtual machine.

In a cloud environment each component is to be thought as a service. The database service, the storage service, the load balancer service and so on. This poses some restrictions on the implementation (e.g. files should be stored on the storage service rather than on the local disk), but allows greater flexibility and scalability.

Instances of the GMC can be added to the instance pool of any of the web servers depicted in figure 4.7, regardless of the hardware which the instance is running on. Of course, for performance reasons, the choice won't be random.

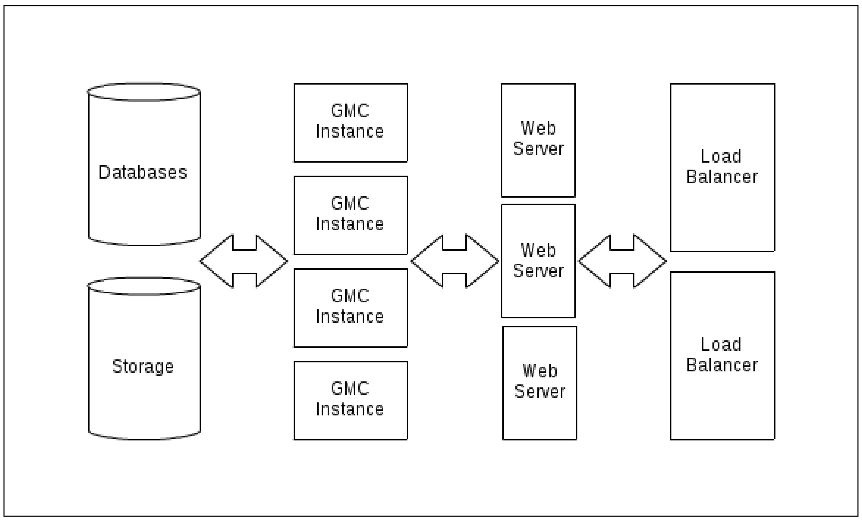


FIGURE 4.7: The Green Move Center network architecture

4.3.1 WEB APPLICATION FRAMEWORKS

We already discussed the database choice, which obviously impacted on the choice of the technologies required to build the GMC.

The most important choice in the development of the GMC is the web application framework. The choice was a-priori restricted to open source software, as for any other technology of Green Move. A modern web application can't be effectively coded without a sound, well tested framework, better if with a large community and online resources. There are a number of good frameworks available, in a variety of programming languages.

4.3.2 RUBY ON RAILS

Ruby on Rails (RoR, or Rails), the application framework chosen for the GMC, is an open source web application framework, written in the Ruby[4] programming language. Ruby is a general purpose language, first released in 1993 by the Japanese programmer Yukihiro Matsumoto. Ruby is object-oriented, interpreted and multi-paradigm (imperative, reflective, functional). It has a variety of intepreters available, which follow different approaches in the realization of the Virtual Machine (VM) which executes the Ruby code.

Citing from Wikipedia: "Ruby on Rails is a full-stack framework, meaning that it gives the web developer the ability to gather information from the web server, talk to or query the database, and render templates out of the box. As a result, Rails features a routing system that is independent of the web server. Ruby on Rails emphasize the use of well-known software engineering patterns and principles, as Active record pattern, Convention over Configuration, Don't Repeat Yourself and Model-View-Controller."

[4]Ruby [http://ruby-lang.org]

During the development of Green Move Center, various updates of RoR have been made available and seamlessly applied, testifying the solidity of the choice performed.

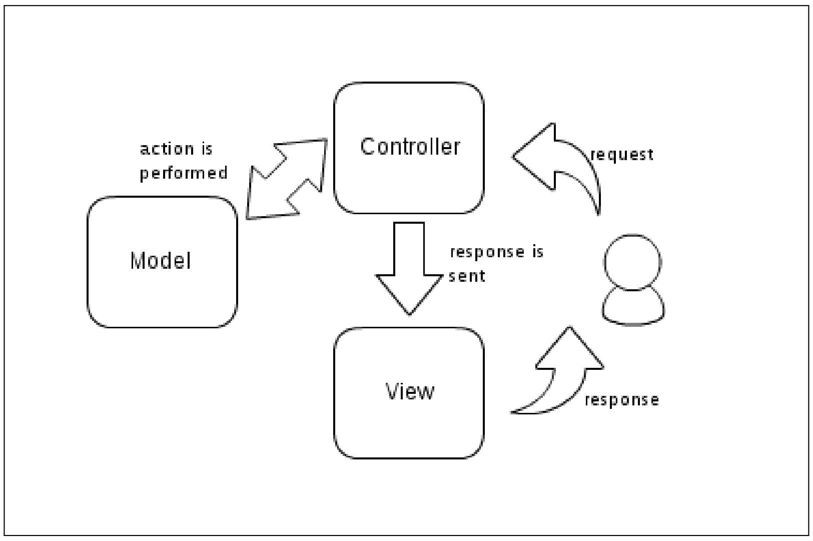


FIGURE 4.8: Model View Controller. The user request reaches the controller, which performs operation on with the model and renders the view, which is sent as thre response.

4.3.3 RUBY AND JRUBY

Ruby is the language in which Rails is written. It is also the language chosen for the development of the web-application part of the GMC. Ruby has an almost unique feature among programming languages, which is the amount of different implementations of virtual machines and interpreters for its execution. The development of the GMC levered this feature to reduce development time and the risk of incompatibilities and errors in future releases, with respect to the T-Rex middle-ware.

JRuby[5] is an implementation of the Ruby VM which runs on top of the Java Virtual Machine (JVM). The main benefits of JRuby are true concurrency of Ruby code (Ruby threads are mapped to Java threads) and a seamleass integration with Java libraries. It is possible to call Java functions from Ruby code. This permitted to use the already available Java client library for T-Rex without coding it from scratch in Ruby[6].

4.3.4 JQUERY, SASS, COFFEESCRIPT AND BOOTSTRAP

Rails strongly promotes the Convention over Configuration principle, so it was a natural choice to follow Rails development conventions in GMC: jQuery

[5]JRuby [http://jruby.org/]

[6]A Ruby client library for T-Rex would have required at least two weeks of coding, and would then require maintenance and updates as the reference library changed.

has been adopted as the JavaScript framework, Sass as the language for Cascading Style Sheets (CSS) and CoffeeScript as the client side scripting language. Sass is automatically compiled to plain CSS by a component of Rails, while CoffeeScript code is compiled to JavaScript.

Since Green Move Center targets all the desktop browsers and the mobile ones, it is crucial to have a consistent look and behaviour and all the browsers and devices. For this reason, a CSS framework has been adopted: Bootstrap (formerly named Twitter Bootstrap).

4.3.5 Devise

Registration core functionality is provided by a Ruby gem named Devise [7]. Devise provides an API for the following use cases:

- sign up
- login and logout, with session management
- password recovery with a request time-out (requires a working SMTP server)
- password change, optionally requiring the user to confirm the previous password
- multi step registration (i.e. users must be approved by an admin or click a link in a confirmation email)

Devise has currently a growing community on Github[8] which counts more than a thousand *forks* and more than 1000 *commits*. It has been chosen for its proven stability and security and to speed up the development process, since such a basic set of functionalities still requires a lot of coding time, and it's an error prone process.

Green Move Center wraps the Devise API and exposes it in two points: a sign up form on the website and a HTTP API accessible by the smart-phone application.

4.3.6 Reservation Process

Reservation starts with a research performed by an user. At the time of writing, the user can provide the following parameters:

- Start date and time
- Type and model of the vehicle (can be more than one)
- A location and a distance in kilometers, for proximity search on the vehicles. If the user wishes to have a vehicle immediately, this can be automatically computed using the GPS on the smart-phone or the geolocation API in a browser (see figure 4.9)

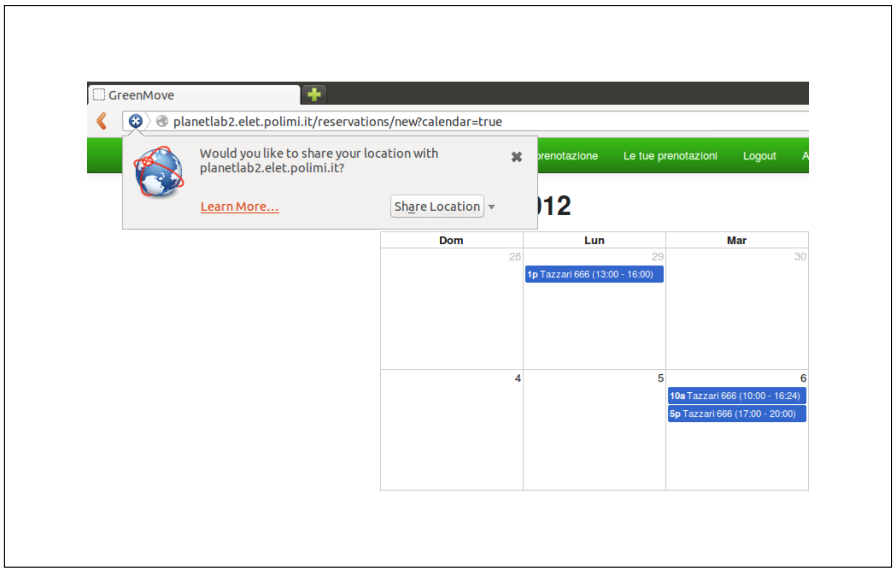

GreenMove
planetlab2.elet.polimi.it/reservations/new?calendar=true
Would you like to share your location with planetlab2.elet.polimi.it?
Learn More...
Share Location
prenotazione
Le tue prenotazioni
Logout
Dom
Lun
Mar
1p Tazzari 666 (13:00 - 16:00)
10a Tazzari 666 (10:00 - 16:24)
5p Tazzari 666 (17:00 - 20:00)

FIGURE 4.9: The user is asked by her browser to allow the GMC to use her approximated physical position. Firefox browser on Ubuntu Linux

The GMC uses the provided parameters to find suitable vehicles, optionally limiting to those present in proximity of the user. Constraints on the remaining charge of the vehicle and additional services are not considered in the currently implemented reservation logic, although a more comprehensive system is being developed by other participants in the Green Move project.

The GMC creates an unconfirmed reservation if at least one vehicle is found to be available in the given location and time. This reservation needs to be confirmed by the user, or will expire in 15 minutes and will be purged by the system.

This is basically the same logic which is adopted in the train and airline business. The unconfirmed reservation acts as a lock on the resource, which here is a vehicle. The resource needs to be freed if the user does not wish to continue with the reservation and does not explicitly cancel the reservation procedure, thus a time-out is needed. 15 minutes is a common value for this kind of use case.

After a reservation is confirmed, its ticket is available. The ticket is not a record stored on the database, it is instead an object computed every time it is needed.

Every time the ticket is requested, it is computed from scratch with the data associated with the reservation. This allows a greater flexibility in the reservation process, since allows to have a fresh ticket if the timing of the reservation changes without impacting the database. This design choice does not, however, add significant computational overhead on ticket retrieval.

[7]Devise: [https://github.com/plataformatec/devise]

[8]Github [https://www.github.com] is a popular platform for hosting open source projects.

The ticket is represented as a set of key-value pairs. Here we enumerate the keys and the relative meaning:

- id: the id of the reservation, which can be used to fetch a new copy of the ticket in case of need
- user-id: the id of the user which created the reservation
- start-datetime: the date and time in which this ticket becomes valid
- end-datetime: the date and time in which this ticket becomes no longer valid
- session-key: the shared key which the vehicle and the smart-phone will use to communicate securely. It is generated with a cryptographic hashing algorithm (SHA1) and salted with the id of the reservation, which is a random string.

When the application on the smart-phone of the user downloads the tickets, additional information is added to each one to aid the user in finding the vehicle, such as the license plate, the color and the model.

When the Green E-Box downloads the tickets, no additional info is required.

As a security measure, every time the tickets are transferred from the GMC, to the smart-phone or the Green E-Box, they are ciphered with asymmetric encryption (RSA with 4096 bit keys). This kind of encryption intrinsically adds security, because the result of encryption is different at each iteration (while the result of decryption remains the same). This helps prevent reply attacks to the crypto-system.

Since encryption generates unprintable characters and the HTTP API of the GMC uses JSON as the standard format, an additional step of encoding the ticket in a Base64 string has been added to the process.

4.3.7 Interactions between the GMC and the Green E-Box during a reservation

During usage, the vehicle will generate a continuous stream of data. The T-Rex based middle-ware presented in the previous chapter is used to publish this data. Any interested party can subscribe to specific events.

A logging mechanism has been developed which subscribes to the currently generated events and log them in the GMC database. In case of errors, faults or user behaviour in contrast with the terms of the GM service, the GMC can perform any useful actions such as calling external services, send emails or other kind of alerts.

This requires that the GMC is somehow subscribed to events produced by the vehicles. In absence of native Ruby library, the java client library for T-Rex has been used in conjunction with the JRuby interpreter, which allows Ruby code to be run on top of the Java VM. The GMC can act both as a regular web application and a daemon, continuously waiting for a subset of the events and performing operations accordingly.

The daemonized mode is used for any activity which is too long to be performed during an HTTP request - with the user waiting on the other side of the network - or batch processing. This includes pushing reservation and ticket data to vehicles and smart-phones and distribution of the GMAs.

4.4 SOFTWARE ARCHITECTURE OF THE GMC - CONTROLLERS

A (simplified) diagram of the controllers of the GMC is shown in figure 4.10.

Methods in controllers are called *actions*, and usually correspond to a single functionality which is reachable by the end user with an HTTP request. Some actions may be declared as private, so their code is not directly reachable but can be called by other actions in the same controller. Private methods are often used to build shared logic, which is then called before, after or around the code of other actions. Rails provides a simple way to manage the execution order of this methods, which is called the *filters pipe*.

A Rails' convention wants the all controller classes to be in the *controllers* folder. Controllers should be declared in files with the name ending in *_controller* (e.g. the Application controller is defined in the file *application_controller.rb* in the controllers folder). Another convention prescribes to use camel case names for class names, and the corresponding snake case for file names, so that the Application controller class is named ApplicationController.

The Application controller is the default controller. It subclasses the base class of controllers in the Rails framework, which is *ActionController::Base*. The main functionality provided by the Application controller is to set the locale, so that proper translations can be loaded and used in the views.

As any other code which is shared among various actions, the *set_locale* method is declared in the super class and inserted in the filters pipe. In this way, it can be executed before the code of each other action.

The *RegistrationsController* subclasses the ApplicationController class and provides methods for the user sign up, login and update. This includes the ability to change the user′ password, the public key and any other user related data.

The *ReservationsController* is essentially a wrapper around the *Reservation* model, described later in this chapter, which contains all the reservation process logic. The *search* action is responsible of parsing the parameters submitted by the user, then asking the *Reservation* class for free vehicles and rendering the view with the results.

The administrative functions live in a name-space, which has a principal controller (*Admin::BaseController*), which is then subclassed by all the others in the *Admin* name-space. This choice gives proper code isolation and also allows the definition of a filter in the *Admin::BaseController* which redirects all the unauthorized users to the home page of the GMC website.

The four controllers defined in the *Admin* name-space provide the follow-

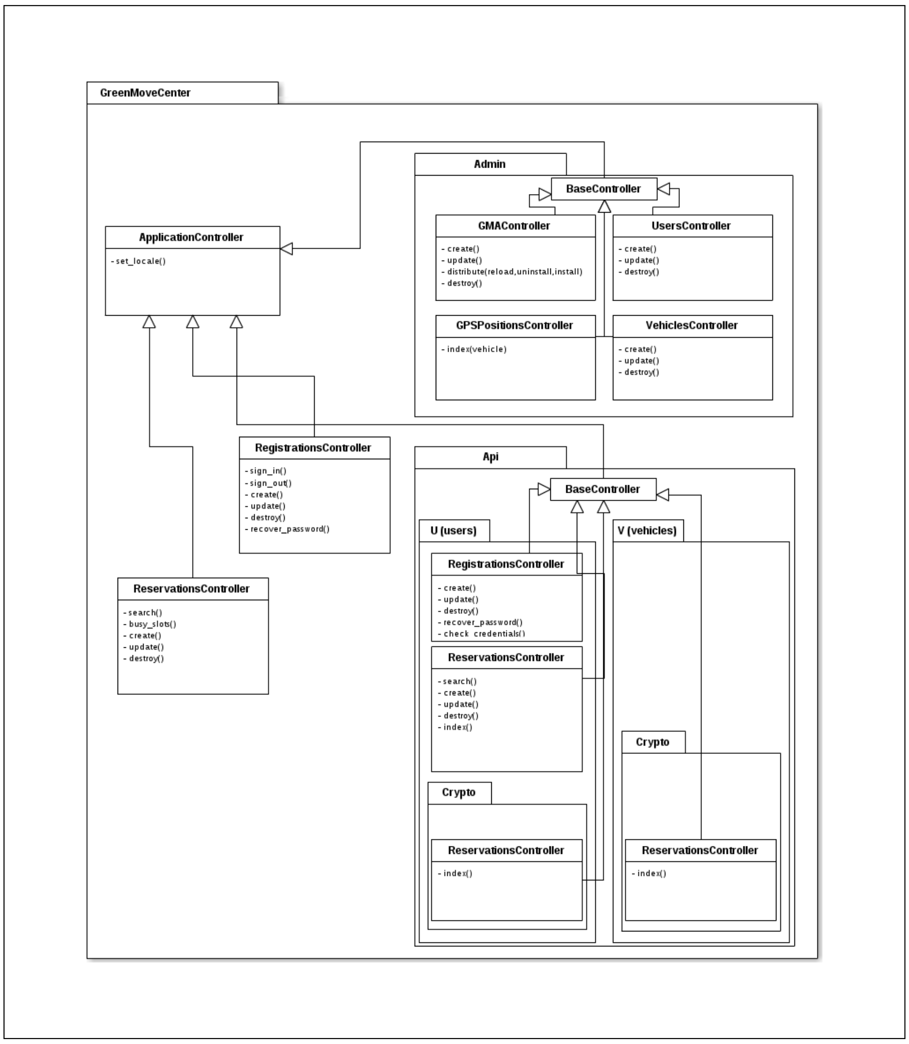


FIGURE 4.10: Controllers of the GMC

ing functionalities:

- GPSPositionsController: displays a map (using the JavaScript APIs available from Google) with the current position of each vehicle, updated in near real-time
- UsersController: provides the admins a way to add, edit or delete users
- VehiclesController: same as the users controller, for vehicles
- GreenMoveApplicationsController: allows selected users to upload GM Applications and schedule operations for their distribution / removal in the Green E-Boxes

The *Api* name-space contains the controllers responsible of providing APIs exposed to the smart-phone client applications (*U* name-space) and the GEBs (*V* name-space). Both of the name-spaces defined under the *Api* name-space contain a *Crypto* name-space, which contains the Reservation controller responsible of the encrypted exchange of the reservation (and ticket) data.

The *Api::U::ReservationsController* mimics the behaviour of the *ReservationsController* described above, with the only differences that responses are all JSON objects and that user's authentication information must be supplied with every request, since the APIs are state-less.

The *Api::U::RegistrationsController* wraps the logic provided by the *Devise* gem, allowing a user to sign-up by using the smart-phone application rather than the website.

4.5 SOFTWARE ARCHITECTURE OF THE GMC - MODELS

Most of the models defined in the GMC, in its current state of implementation, are only used to comply with the underlying database structure and constraints (e.g. referential integrity).

Figure 4.11, 4.12 and 4.13 show the models of the GMC and the relationships among them, which reflect the foreign keys present on the Postgres database. Attributes are not shown for the sake of readability. A detailed representation of the database schema is given in Appendix D. The choice of PostgreSQL allowed the developers of the database structure to enforce foreign keys integrity checks among the tables. This mean that the database is not a mere data store with a layer of software to perform queries, but is in fact an application itself. This is a good design choice when heterogeneous clients might access and modify the stored data, since allows to keep access authorization logic in a shared place. As a side effect, this choice gives a significant performance increase w.r.t. duplicating the same logic with a scripting language as Ruby.

However, the logic which ensures referential integrity has to be repeated in the Ruby code of the models, otherwise the database will deny transactions (e.g. deleting a vehicle without prior deletion of its reservations).

Since Rails does not provide a way to distinguish *views* from *tables* on the underlying database, a model must be declared to access data provided by the database *views*. Proper attention must be put when implementing these models in not performing write operations on such views, since this is not possible and an exception will be raised by the database adapter.

4.5.1 THE RESERVATION MODEL

The *Reservation* model deserves a special mention, since it provides the functionality both for searching and for reserving vehicles. This model maps the *reservations* table on the database.

As previously stated, the *reservations* table is used both to store the confirmed reservations and the temporary reservations, which are purged after 15 minutes if the user does not confirm them. The *proposals* method of the

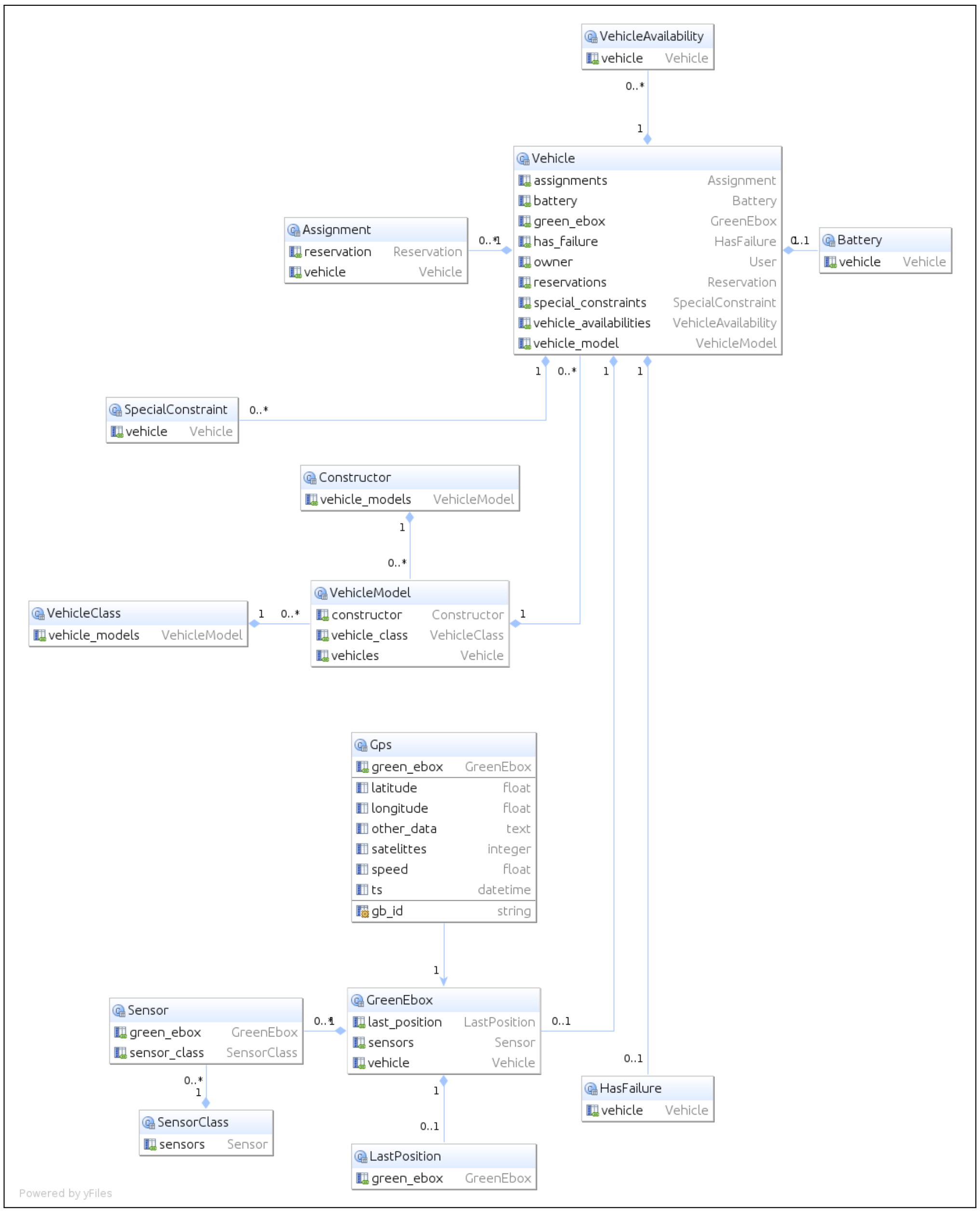


FIGURE 4.11: Models of the GMC - part 1

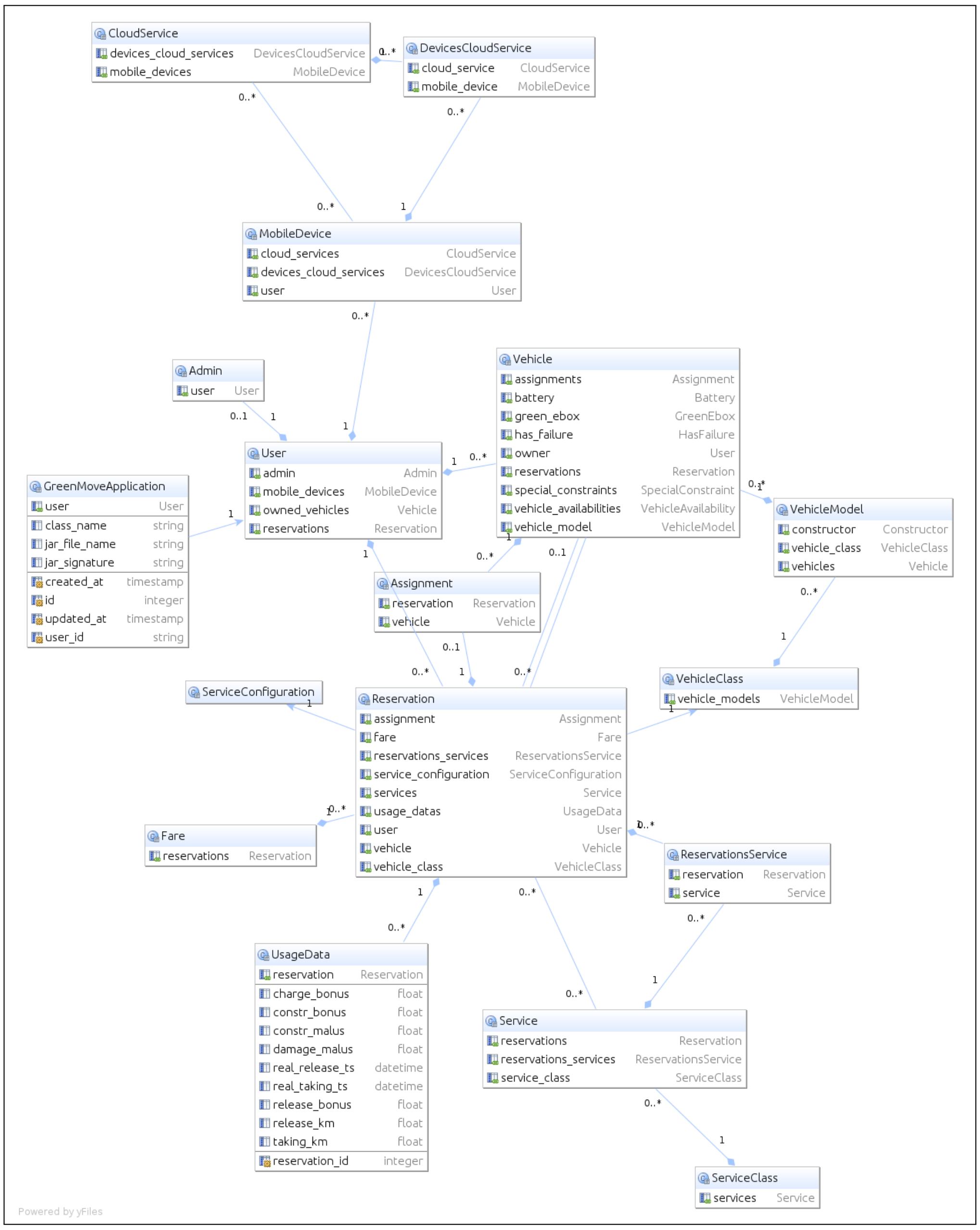


FIGURE 4.12: Models of the GMC - part 2

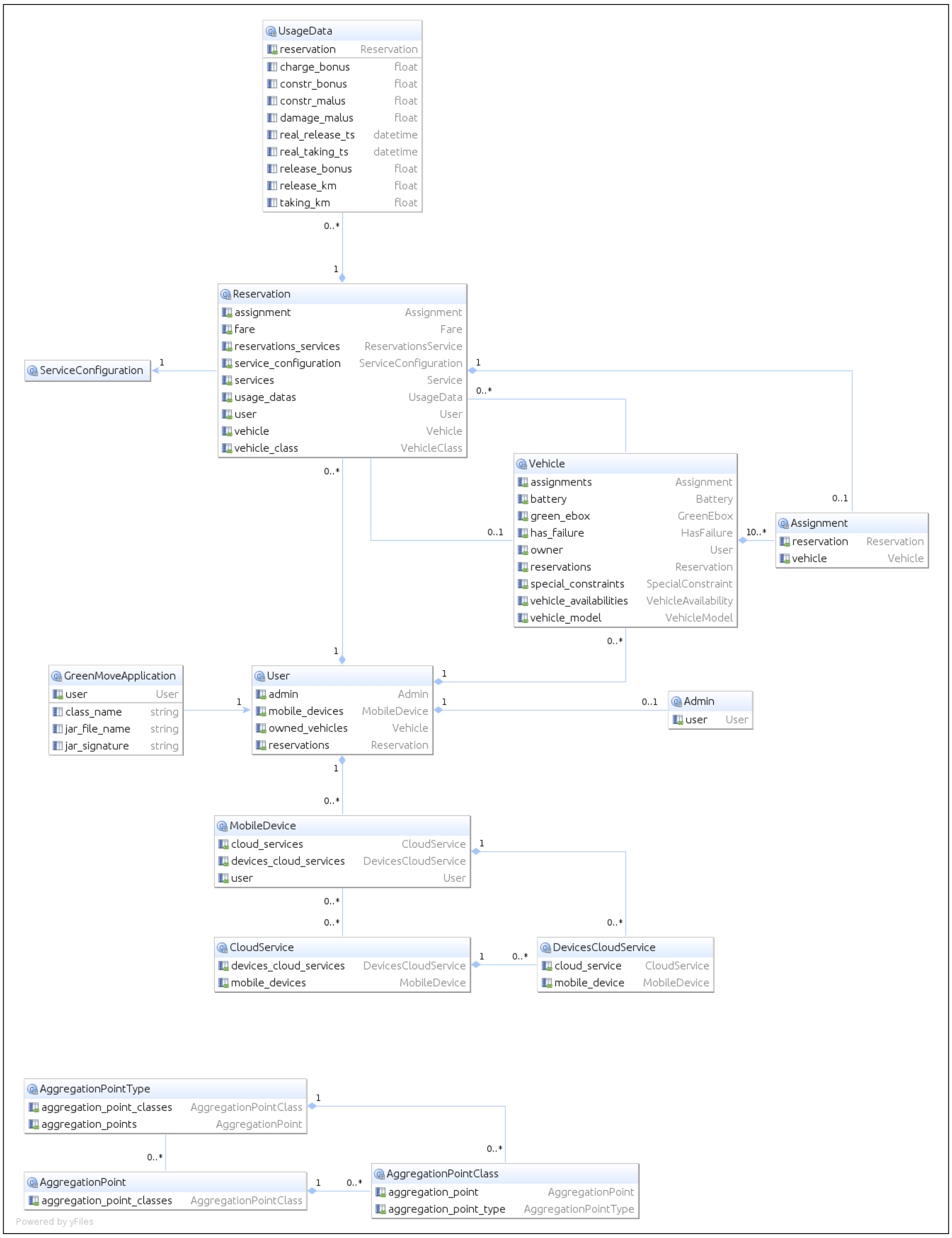


FIGURE 4.13: Models of the GMC - part 3

Reservation model is responsible of creating temporary reservations, given the user input[9].

The current implementation of the *proposals* method does the following:

- Vehicles near to a user's specified address are considered for the research. the user may specify a maximum distance which will affect this operation
- Vehicles are optionally restricted to a given model or specific vehicle, if specified
- If more than one vehicle is found in the previous steps, only the first one is considered
- An attempt to save the reservation is made, triggering validation logic which ensures that the given vehicle has no overlapping reservations in the user's supplied time frame
- The list of the unconfirmed reservations for the given user is returned, so the user can continue the reservation process confirming one of the temporary reservations before they get deleted

Needless to say, the logic described above is pretty basic and not suitable for a production system. For example, the first vehicle might be busy in the given time but a second one might be available. The above logic will fail to find the free vehicle. Also, a user is allowed to perform various searches without its temporary reservations getting deleted. This is not safe in a production environment w.r.t. to denial of service attacks.

The problems presented are easily solvable. The described logic, however, is just a placeholder for the upcoming reservation logic. Green Move aims to integrate various systems under its standardized protocols, so the actual logic will depend on a great number of factors and conditions. The simple logic currently implemented has been good enough to perform tests in this first phase of experimentation.

The *Reservation* model is also responsible of providing ticketing data to both the Green E-Boxes and the client applications on the users' smart-phone.

The method semantically named *data_for_smartphone* assolves the last duty. It returns a dictionary with two key-pairs.

- additional_info: a dictionary containing the license plate, mac address, color of the vehicle and other human readable informations to help the user find the correct vehicle in a parking lot
- ticket: the ticket is a dictionary containing information about the start end the end times of the current reservation, along with its unique id and the id of the user. The ticket also contains the session-key, which

[9]A more complex reservation system is ongoing. The user will be granted a reservation with no particular vehicle, and GM will take care of finding the optimum solution just in time for the start of the rental.

will be pre-shared secret between the Green E-Box and the smart-phone application when communicating over blue-tooth.

The *data_for_greenebox* method returns a dictionary containing only the ticket data, which is the same of the *data_for_smartphone* method.

Both methods convert the dictionary in the JSON format, then cipher the resulting string with the public key - of the user and of the vehicle, respectively - and encode the binary output of the cipher operation in a Base64 string.

The session-key is the result of a SHA1 sum operation, currently involving a fixed salt and the unique identifier of the reservation.

The session-key generation mechanism can be furtherly hardened. However, an attacker which can guess the session-key will only be able to establish a blue-tooth connection with the Green E-Box, since the vehicle has its own copy of the reservation, which contains the timing information.

Knowing the session-key will allow the attacker to perform a man in the middle attack, which will require to be in the range of the blue-tooth tranmission. Having the session-key allows the attacker to read the communication between the Green E-Box and the smart-phone in clear text, so this could be used to inject commands to the vehicle.

The problem with this attack is that the id of the reservation is random and only the vehicle and the user can read it, because is transmitted ciphered with their public keys (and over a SSL connection, when running in production).

This said, of course hardening proposals are possible and welcome.

— CHAPTER 5 —

GREEN MOVE APPLICATIONS

While developing the Green Move Center and the Green E-Box software many ideas and use cases have emerged with respect to the potential of Green Move. One of these ideas was about the interest of third parties in interacting with Green Move users or vehicles.

T-Rex provides a standardized protocol for exchanging complex events, but the overall level of interaction for third parties is limited to subscribing and publishing events. This might or might not be enough interesting for some actors.

Consider the case of an advertising agency which decides to provide some targeted ads to Green Move users. Currently there is not a standard in Green Move to achieve this. Sure we can imagine advertisements being sent as events, with a pre-defined component on the Green E-Box which performs subscriptions based on the kind of advertisements suitable for the current user on board.

However, other needs may arise in the future and we can't think upfront of a solution for all of them. This is especially true for the Green E-Boxes. While the code of the Green Move Center is easily extendable and new versions can be deployed anytime, the process of upgrading the software on all the Green E-Boxes can be slow, with uncertain results and at worse even very expensive due to the number of the vehicles and the multi-ownership model which characterize Green Move.

Android applications can be made auto updatable. This can appear to be a viable solution to the problem of providing new functionalities on the Green E-Box applications. There are two main reasons for which this is not an ideal solution:

- Dependency on a third party for updates distribution and timing. Although Google is a great example of stability and speed, it is an hazardous choice to rely exclusively on Google to deliver new features to the Green E-Box application.

  Specifically, the timing of the distribution of updates is subject to the acceptance of the latter in the Android market, which may be unsuitable for urgent updates. Also, in case of a failed update, manual intervention will be required. This means that a vehicle on which the update has failed must be identified, reached (online or offline) and fixed.

  This is critic even for an approach where GMA are shipped as separate applications, rather than updates of the existing Green E-Box app.

- Poor flexibility and inefficient use of resources of the Android board. Since the application runs on an Android board it has limited resources, both in terms of memory and of computational power. Even if all of the use cases were covered with appropriate functionalities embedded in the application, the latter would become very large and complex. Also, most of the functionalities would be unused because the vehicle may not be rented for days or always taken by users for whom there is no need to have all that functionalities in place.

Ideally, we would like to have a minimal application running on Green E-Boxes which handles the core tasks (take and release of the vehicle, collection of vehicle data and transmission of data to T-Rex Server). For any other need, it would be better to have the chance to add functionalities only when there is need of them for a particular user, location or period. When no longer needed, they could be simply removed from the device. This would help the application to remain light and eliminate the need to think upfront of all the use cases, at cost of having a new problem of building a system capable to achieve this goal. In analogy with the Android applications, we named Green Move Applications (GMA) all of the extensions to the Green E-Box application which can be installed on demand.

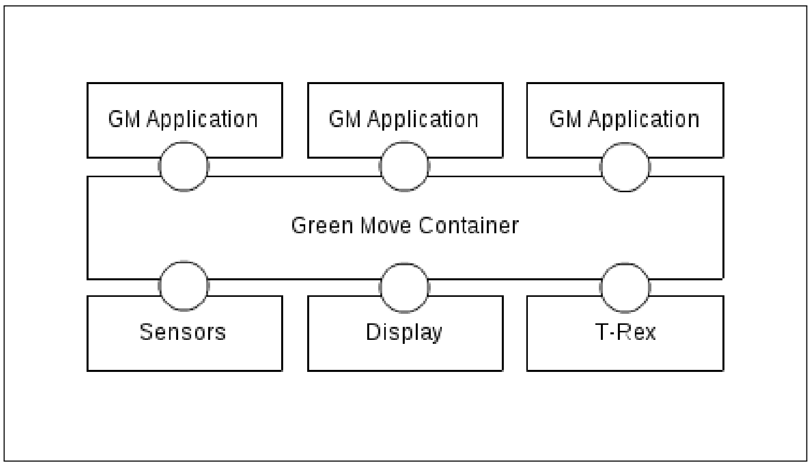


FIGURE 5.1: Green Move Applications interfaces.

The GMA system is composed of a component which resides on the Green E-Box (Green Move Code Agent) and a module of the Green Move Center (Green Move Applications Server). Green Move Applications are uploaded on the Green Move Applications Server and distributed to Green Move Code Agents leveraging the T-Rex middleware.

In the next section we describe the structure and the conventions necessary to build a GMA.

5.1 GREEN MOVE APPLICATION STRUCTURE

A GMA is essentially a standard Android Library which adheres to a set of conventions. The conventions are required for the correct operation of the GMA loading mechanism. The code of the GMA must be precompiled for the

Dalvik Virtual Machine and packaged in a JAR file to be distributable. Details on this process are provided in appendix C.

The critical points in the operation of loading a Java class are the following:

- the complete package name must be known a-priori
- only the default constructor can be called on the target class

Having this constraints, the GMA must provide a single point of entry in the application code which must be a class, and have only the default constructor. Additionally, a *init* method must implemented to receive a reference to the Green Move Container object, which provides API for interacting with the vehicle and the T-Rex Server. In order to quit the execution of a GMA, a *stop* method must be implemented.

To enforce this conventions and provide the necessary dependencies for the development of GMAs, an Android library named GMCodeLibrary has been developed and made available, along with its documentation.

Given this pre-requisite, a GMA can perform any operation allowed by the Android OS.

Operations requiring data collected from the vehicle can subscribe to updates from the embedded board present in the Green E-Box. The Green Move Container, a component of the Green Move Code Agent described later, provides a reference to the component responsible for the interactions with the vehicle.

Each Green Move Application can use the T-Rex middleware to publish and subscribe to events generated in any other part of the GM system.

In the next section we describe the Green Move Code Agent, a module for the Green E-Box application which provides the functionality for adding Green Move Applications along with a standardized API for the latter.

5.2 GREEN MOVE CODE AGENT

Since the Green E-Box application runs on Android, it has most benefits of the code running on a Java Virtual Machine. Java classes can be loaded programmatically when needed, so it is possibile to think of a system which downloads code from the network and loads it to extend the functionalities of the Green E-Box application. Green Move Code Agent (GMCA) is the component responsible of such behaviour.

GMCA require different Android components:

- GMContainer: an Android service. It runs in the background and waits for input. It is responsible of providing an API for the Green Move Applications to interact with the vehicle and the GMC.
- GMTRexClient: an Android service. It runs in the background. It is responsible of keeping a connection with T-Rex Server to receive input about the GMA to load. Will use the JarRetriever to perform the task, and notify the GMContainer when done.

- JarRetriever: a class responsible of downloading and loading the code of a GMA.

5.2.1 GREEN MOVE CONTAINER API

The Green Move Container is the component responsible of providing a standard API for the GMAs, both for loading/unloading and for interacting with the vehicle and the T-Rex Server. It is an Android Service, always running in the background, since it only performs tasks when prompted. The Green Move Container keeps a list of references to every GMA loaded on its device, since it is the only way to perform the unloading of that code.

It provides an administrative API which is required to ask loading, unloading and listing of GMAs:

- loadClass(<url of jar file>,<class name>): fetch the GMA from the given url and attempt to load the class with the given name
- unloadClass(<class name>): ask the GMA to terminate, if loaded
- listClasses(): list the currently loaded GMAs

Green Move Container also provides an API for GMAs, consisting in the following:

- getDataModel(): returns a reference to an object which can be subscribed for updates in sensor readings, providing data about GPS position, speed, etc.
- getTransportManager(): provides the caller a reference to its own instance of a component required to communicate with the T-Rex Server.
- toast(message): gives the GMA a method to interact with the user on board, showing a message in the display of the Green E-Box (where available).

5.3 GREEN MOVE APPLICATIONS SERVER

The component responsible of the distribution of GMAs to Green E-Boxes is the Green Move Applications Server (GMAS). The GMAS is available, for selected users, inside the GMC. Jar files containing the applications are uploaded on the GMAS, along with the information required to load them (class name).

The user may choose among three operations on each of her GMAs:

- load: the application should be loaded on the Green E-Boxes
- reload: the application should be updated on the Green E-Boxes
- unload: the application should be uninstalled from the Green E-Boxes

5.3.1 Distribution of an application

The GMAS stores the uploaded jar in a directory and puts the jar url and class name in a queue, along with the operation to be performed.

A background job, which uses the JRuby interpreter, estabilish a connection to the T-Rex Server and processes the queue. For each item in the queue an event is published.

Currently, only broadcasting or unicasting GMAs is supported, due to an intrinsic limitation of any publish/subscribe protocol, such as the one adopted by T-Rex.

Subscribers shall subscribe to specific events, optionally providing filters on the event contents. This is not suitable in the scenario in which is the publisher who wants to choose specific receivers for its messages, such this case.

Instead of duplicating the infrastructure and the connections between the GMC and the Green E-Boxes, a query-advertise mechanism will be added to the GMA system in order to overcome this limit.

— Chapter 6 — Testing

Such a complex system as Green Move requires a big effort in testing all of its functionalities. The earlier tests are started, the better the resulting system will be.

This chapter focuses on the testing activities conducted on the Green Move Center and the Green Move Applications. The testing activities included the interactions between the aforementioned components and the T-Rex Server and the Green E-Box.

A great effort has been put in place to automatize testing since the very start of the development activities, especially for the Green Move Center.

The approach adopted in the development process is known as Test Driven Development (TDD) and it's briefly described in the next section. In literature, a slightly different approach to testing is known as Behaviour Driven Development, which is in effect the same thing of TDD.

6.1 Test Driven Development

TDD is a software development process, often found in Agile Development practices, that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test and finally refactors the new code to acceptable standards.

The general rule of TDD is to write tests before the code whose behaviour is intended to be tested. This is not mandatory, however, and it's largely discussed among the open source communities of developers.

This topic is somehow connected to another one, which is about the amount of code which should be covered by a test (code coverage). It is impossible to test every possible behaviour of a method, and for really simple methods this can result useless. A good rule of thumb about code coverage is the following:

> *You should test something until the boredom overcomes the fear of unintended behaviour*

This rule, although not formally provable to be right, has been chosen as the guide for the testing of the Green Move Center.

Another important practice of the Test Driven Development process is running tests continuously, because the test should pass as soon as the minimum amount of code required to achieve the described functionality has been written. The last phase of the cycle, refactoring, should also be guided by the test. At the end of the cycle, the test should be passing again.

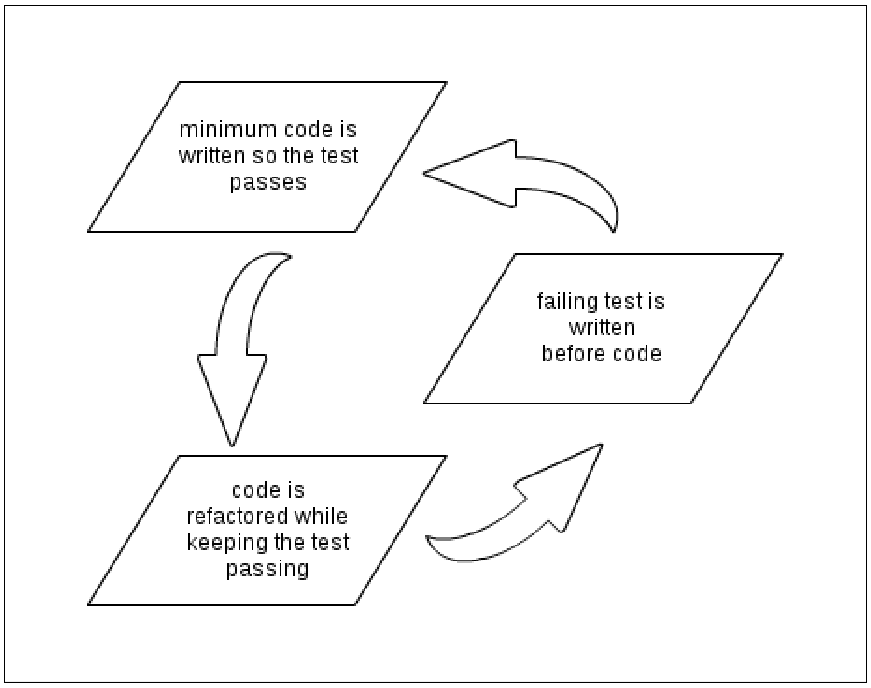

minimum code is written so the test passes
failing test is written before code
code is refactored while keeping the test passing

FIGURE 6.1: Test Driven Development cycle.

Obviously, this means that a growing set of tests will be exists in any given time during the development. A common way to introduce bugs in code is to change the behaviour of a method without considering where that method is called, or which side effects of the method are required. This kind of bug is usually a pointer of bad design in the software itself, but this goes beyond the scope of this work of thesis.

To avoid bugs introduced by refactorings, each test in the test suite should be executed whenever a change in the code is performed. This can rapidly lead to a situation in which the amount of time required to run the test suite is too long.

As we have seen, building a test suite for a software project is a complex task, so proper tools have to be chosen and used.

6.1.1 RSPEC

RSpec[1] is testing tool for the Ruby programming language. RSpec can be used to perform unit, integration, acceptance and functional testing. Testing with RSpec is a common practice in the development of a Rails application, since it is very flexible and easy to use with a software developed following the Model View Controller paradigm.

RSPec provides a Domain Specific Language (DSL) capable of expressing assertions in a very readable manner. This DSL is just an expressive layer,

[1]Rspec - [http://rspec.info]

so RSpec tests are written in pure Ruby. The main benefit of this DSL is that complex assertions can be expressed without writing boilerplate code which might compromise the clarity and the purpose of the test itself.

Below are some examples of RSpec assertions:

- Testing that the result of an operation is true

  ```
  resultOfOperation.should be_true
  ```

- Testing that a controller redirects the user instead of rendering a page

  ```
  controllerResponse.should redirect_to someOtherController
  ```

- Testing equality between objects

  ```
  someObject.should eq someOtherObject
  ```

Another important characteristics of RSpec is that it allows *objects mocking*. Mocking consists in substituting the behaviour of a method dinamically, so complex and time consuming operations can be avoided and the time of the test execution reduced. This of course implies a trade-off, because mocking might introduce bugs in the test code.

An example of mocking is replacing a call to an external web service, which might be slow or unavailable when running the test suite, with a sample response.

Different tools have been developed for the purpose of providing a DSL for mocking. RSpec allows the developer to choose among various mocking tools, including its own implementation. In this work, Mocha[2] has been chosen.

6.1.2 Factories

Test cases require different classes to be instantiated and used, so there is need of a set of sample records in the database. To avoid dependencies between tests, which may lead to unpredictable behaviour based on test race order, a good approach is to reset the sample records before each test starts.

A static set of sample records is not ideal for testing in a TDD fashion, as it might be unnecessarily big for many test cases, leading to a slow execution time for larger test suites.

Factories are components responsible of providing template instances of a model class. Using factories, the set of sample records can be built on a per-test basis, while keeping the logic needed to build such record in a unique place.

FactoryGirl[3] is a Ruby gem which provides factories, and it is integrated with Rails with the FactoryGirlRails[4] gem.

FactoryGirl factories are tied to the model for which they provide a template, so that the logic associated with that model remains in place when a

[2]Mocha [http://gofreerange.com/mocha]
[3]FactoryGirl - [https://github.com/thoughtbot/factory_girl]
[4]FactoryGirlRails - [https://github.com/thoughtbot/factory_girl_rails]

factory is used (e.g. callbacks after the creation of a record). FactoryGirl also provides a simple DSL to manipulate factories, which is in pure Ruby. Examples of factories are reported in Appendix B.

6.2 GREEN MOVE CENTER API TESTING

The APIs of the Green Move Center have a complete test suite. They are a core component of the GM system, so special care has been put in the effort of covering every possible execution path, including edge cases. The APIs are stateless and have no view, so the automatized testing activity was focused on the controllers. Automated testing has been conducted continuously along manual testing, performed *consuming* the APIs with the smartphone clients.

The functionalities provided by the Devise gem (authentication, with respect to the API) are already covered with the gem own tests, so only the integration behaviour with GMC has been considered.

The following is a summary of currently implemented tests on the GMC APIs:

- The public key of an user should be downloadable, so the smartphone application can use it to verify that the private key stored locally is correct
- An action for checking the correctness of the password should be present
- A list of the reservations of the user must be downloadable
- APIs should deny access to users with wrong username/password combination
- The ending time of a reservation should be recognized when supplied as a timestamp in the UTC time zone
- A registered user should be allowed to perform a search for a reservation
- The Green E-Box App should be able to download a list of the reservations for its vehicle, ciphered with the public key of the latter

— Chapter 7 —
Conclusions

The work presented in this document covers a period of more than a year, involving various people and only limiting the focus on the development of the GMC and the GMA subsystems of Green Move.

A complete summary of the activities is impossible, so some of them have just been discarded when writing this thesis (e.g. the presentation of Green Move with the test drive occurred on October 13th or the realization of a video clip).

Green Move represents an innovative approach to tackle the mobility problem described in chapter 2. The technologies developed by the author of this thesis and others component of the work group at DEI, discussed in the previous chapters, are aimed at providing a working prototype (a minimum viable product) of what could become the real system when implemented for massive use.

The choice of a cloud-ready architecture, with state of the art technologies in many fields (web application framework, database, NFC) along with experimental technologies developed at DEI for research purposes and used in the GMC (e.g T-Rex) puts a solid basis on which build the future Green Move vehicle sharing system.

The openness of this system, which is built on APIs and standards, along with the multi-ownership - multi-vehicle model and the strong automatization of business operations, puts Green Move a step forward w.r.t to the existing vehicle sharing systems.

7.1 Development considerations

The use of the Android operating system has been chosen upfront for the sake of openness, and it surely posed some constraints in the development which influenced the decisions taken later.

7.1.1 Android

Android does not provide a way to be sure that the Green E-Box application software won't ever be backgrounded or even closed by the OS. Also, interaction between components of the application on the Green E-Box must strictly adhere to the Android conventions.

In some circumstances, a completely custom device with the same form factor as the Android board part of the GEB running a customized GNU/Linux (or BSD) distribution would have simplified the development process.

7.1.2 T-REX

The T-Rex middle-ware has shown many aspects of immaturity, which are typical of a system developed in the academia and never used in a real situation. The T-Rex protocol used to communicate with the T-Rex Server is very light and fast but lacks basic resilience to errors. Currently, a malformed packet can cause the crash of the T-Rex Server process, thus a watchdog system had to be put in place to avoid downtime in this component.

Also, numerous security problems are present. Sending events with attribute values longer than the maximum length may result in buffer overflows.

Luckily, all these problems are solvable with a security minded hardening of the T-Rex Server and a complete rewrite of the communication protocol (thus releasing a new client library, keeping the same APIs).

7.1.3 GREEN MOVE CENTER

The development of the GMC has seen a complete rewrite of the web-application, since the schema of the database was completely replaced from a basic MySQL one to the complete PostgreSQL one which is reported in appendix D.

In such a circumstance, with API clients already being developed, the TDD approach has shined in great light. Tests developed were keeped, along with URLs of the API endpoints and format of the messages exchanged with clients.

All the changes required by the new schema were performed at the model and controller level, untils the tests were passing again.

7.2 FUTURE WORK

Green Move is undergoing great development, both on the IT side and on the others side of the project. On the IT side, there are very interesting open topics, some of which are described below.

The next months will see the experimentation of PerLa[1], a context aware system to query data from heterogeneous, pervasive systems. PerLa might be used in place of the current T-Rex based system for collecting sensors data, since in allows to move filtering logic in the GEBs and this can potentially help a lot w.r.t to network usage and performance of the overall system.

The GMC needs to be completed with the flexible reservation logic which is being developed. As described in chapter 4, the current implementation is very basic.

Also, the NoSQL database has to be chosend and integrated with the GMC. This poses some interesting problems of integration of heterogeneous data stores in a single application. As a need for a more complete geographic information system (GIS) might arise, integration of such system with the GMC may require a good amount of work on the database adapters and the application framework. This is a good research topic, which might also result in useful contributions to the open source software community.

[1]PerLa - [http://perlawsn.sourceforge.net/documentation.php?official=1]

The GMA subsystem is currently implemented as an additional component w.r.t the Green E-Box software, so an integration of the two softwares is necessary. Also, the efficient distribution of GMAs require a query-advertise middleware which has to be chosen or customly developed for Green Move.

As the number of vehicles and users circulating in the GM system grows, the cloud-ready architecture might be leveraged to switch from the currently deployed physical server to a true cloud infrastructure.

Administrative functionalities of the GMC are only stubbed, so functionalities such as billing and fault monitoring/response are to be developed.

Another important work to be done, although it does not pose particular technologic difficulties, is the development (or the integration) of a content management system (CMS) within the GMC. It is in fact impossible to imagine Green Move without a user facing site with other informations and services except for the reservations.

— CHAPTER A —

T-REX RULES AND EVENTS

The currently implemented events in the GM system are two, plus one which is subjectible to changes since it's only used in the GMAs experimentation.

A brief description of the events is supplied along with some background information on the T-Rex rules.

Position Event

- id: 13
- source: Any vehicle
- contents: Vehicle position represented by two doubles, with information about the source (gps, network)
- event kind: Periodic
- id of the Green E-Box

The Position Event is used to have a real time map of the vehicles in the GM system. The GMC needs this information to perform the research of vehicles in the nearbies of the user, and to display them on the map in figure 4.6.

Sensor Data Event

- id: 17
- source: Any vehicle
- contents:
  - battery status
  - current
  - direction sense (backward, forward)
  - door status
  - faults
  - gps data
  - id of the Green E-Box
  - heating status (on, off)
  - state of charge
  - speed
  - timestamp

- event kind: Periodic

The Sensor Data Event provides a complete set of information on anything that may be collected with the sensors on board of the vehicles. This event does not always contain all of the attributes presented above.

Rules Currently, only a simple set of rules is installed on the T-Rex engine. They basically allow the events received by the engine to be forwarded to the subscribers for that kind of events, since subscriptions are currently done only on a per event-id basis. TESLA rules with more complex logic are under development to leverage the potential of the T-Rex middle-ware.

A rule allows to use a temporal first order logic to express conditions on the incoming stream of events and perform actions accordingly. The actions are, usually, the generation of new events which use the data contained in the input events.

The code which allows the event with id 13 to be forwarded is reported below. This rules simply takes an event with id 13 and builds a new one with the same attributes.

```

#include "RuleEvent13.hpp"

using concept::test::RuleEvent13;

RulePkt* RuleEvent13::buildRule(){
        RulePkt* rule= new RulePkt(false);

        int indexRootPos= 0;

        // root Position predicate
        Constraint constr[1];
        strcpy(constr[0].name, "greenBox_id");
        constr[0].type = STRING;
                constr[0].op = IN;
        strcpy(constr[0].stringVal, "-");
        rule->addRootPredicate(13, constr, 1);

        // template
        CompositeEventTemplate* templ= new CompositeEventTemplate(13);

        OpTree* idTree = new OpTree(new RulePktValueReference(indexRootPos, "greenBox_id"),
             STRING);
        templ->addAttribute("greenBox_id", idTree);

                OpTree* latitudeTree = new OpTree(new RulePktValueReference(indexRootPos, "
                     latitudine"), FLOAT);
        templ->addAttribute("latitudine", latitudeTree);

        OpTree* longitudeTree = new OpTree(new RulePktValueReference(indexRootPos, "
             longitudine"), FLOAT);
        templ->addAttribute("longitudine", longitudeTree);

        OpTree* sourceTree = new OpTree(new RulePktValueReference(indexRootPos, "sorgente"),
             STRING);
        templ->addAttribute("sorgente", sourceTree);

        rule->setCompositeEventTemplate(templ);
        return rule;
}
```

GMC support tools During the manual test phasis, the need for a tool to debug rules and events arised. Since T-Rex Server allows to install rules at runtime, this functionality would have been nice to have in such tool.

This tool is part of the GMC, although it is usable only when running the JRuby interpreter since it requires the T-Rex client library. The tool is named TRexClient and is located in the *lib* folder of the GMC.

It provides the following functionalities:

- connect to T-Rex Server
- send a Position Event
- send a Sensor Data Event
- install a rule for both the above events on the T-Rex engine
- subscrive to events
- install a listener to process incoming events
- the connection with the T-Rex Server is kept alive automatically

— CHAPTER B —
RSPEC TESTS

Factories Reservation Factory. This factory builds a Reservation model.

```
FactoryGirl.define do

    factory :fare do
        type { Faker::Company.catch_phrase }
        description { Faker::Company.bs }
        price { rand * 1000 }
        valid_from { 1.year.ago }
        valid_to { 1.year.from_now }
    end

    factory :service_configuration do
        name { Faker::Company.catch_phrase }
        description { Faker::Company.bs }
    end

    sequence :reservation_id do |n|
      "#{n}"
    end

    factory :reservation do
        id { FactoryGirl.generate :reservation_id }
        taking_ts { Time.now }
        user { FactoryGirl.create :user}
        vehicle_class { FactoryGirl.create :vehicle_class}
        fare { FactoryGirl.create :fare }
        service_configuration { FactoryGirl.create :service_configuration }


        factory :confirmed_reservation do
            confirmed true
            release_ts { taking_ts + 3.hours }
        end
        factory :unconfirmed_reservation do
            confirmed false
        end

    end

end
```

User factory.

```
FactoryGirl.define do

  sequence :user_id do |n|
    "#{n}"
  end

  factory :user do
    email { Faker::Internet.email }
    name  { Faker::Name.first_name}
    surname { Faker::Name.last_name }
    username { Faker::Internet.user_name }
    #key size should really be 4096
    k=OpenSSL::PKey::RSA.generate( 4096 )
    public_key_string  k.public_key.to_s
    private_key_string  k.to_pem.to_s
    password "pippo123"
    password_confirmation "pippo123"
    owner false
    customer true
    id { FactoryGirl.generate(:user_id)}
  end

end
```

Vehicle factory.

```
FactoryGirl.define do

  sequence :license_plate do |n|
    "targa#{n}"
  end

  sequence :green_ebox_id do |n|
    n.to_s
  end

  factory :constructor do
    name { Faker::Company.name }
  end

  factory :vehicle_class do
    description { ['car' ,'scooter'].sample}
  end

  factory :vehicle_model do
    name { Faker::Company.bs }
    constructor FactoryGirl.create(:constructor)
    vehicle_class FactoryGirl.create(:vehicle_class)
  end

  sequence :vehicle_id do |n|
    "#{n}"
  end

  factory :vehicle do
    k=OpenSSL::PKey::RSA.generate( 4096 )
    public_key_string k.public_key.to_s
    private_key_string k.to_pem.to_s
    vehicle_model { FactoryGirl.create(:vehicle_model)}
    id { FactoryGirl.generate(:vehicle_id)}
  end

  factory :green_ebox do
    id { FactoryGirl.generate(:green_ebox_id) }
  end

end
```

Some of the RSpec tests are reported in this appendix along with comments to help understand the structure and the logic.

Test assuring that the user can view the list of its reservations.

```
require "spec_helper"

describe ReservationsController do
  include Devise::TestHelpers
  it "should list the reservations for the current user" do
    @user=FactoryGirl.create :user, :email=>"test@gmc.it",:password=>"123pippo",
    :password_confirmation=>"123pippo"
    sign_in @user
    get "index"
    response.should be_success
  end
end
```

Test assuring that a vehicle can download a list of its reservations, ciphered with the vehicle key and encoded in Base64

```
require "spec_helper"

describe Api::V::Crypto::ReservationsController do
  include Devise::TestHelpers
  before :each do
    @user=FactoryGirl.create :user
    @car= FactoryGirl.create :vehicle
  end

  it "should respond with the base64 encoded data of the reservation, ciphered with the
       vehicle public key" do
    @reservation = FactoryGirl.create :confirmed_reservation,
     :user=>@user
    @reservation.vehicle=@car

    get "index", :format=>"json", :vehicle_id=>@car.id
    response.should be_success
    assigns[:reservations].should eq [@reservation]
  end

end
```

Test assuring that public key of a user should be downloadable and that a user is allowed to check its password against the system.

```
require "spec_helper"

describe Api::U::RegistrationsController do
  include Devise::TestHelpers


  it "should allow public_key_string downloading" do
    @user=FactoryGirl.create :user
    @request.env["devise.mapping"] = Devise.mappings[:user]

        post 'public_key', :format=>:json, :user=>{:email=>@user.email,:current_password=>
             @user.password}

        response.should be_success
        response.body.should eq @user.public_key_string

  end


  it "should allow password checking" do
    @user=FactoryGirl.create :user
    @request.env["devise.mapping"] = Devise.mappings[:user]

        post 'check_credentials', :format=>:json, :user=>{:email=>@user.email,:
             current_password=>@user.password}

        response.should be_success

  end

end
```

The following code tests actions for the Green Move APIs:

- An authenticated user should be able to retrieve the list of her reservations
- The system should deny access to users with wrong username/password combination
- GMC should recognize the end time of a reservation when supplied as UTC timestamp
- A registered user should be able to search for vehicles

```
require "spec_helper"

describe Api::U::ReservationsController do
  include Devise::TestHelpers
  before :each do
    @vehichle_class_car = FactoryGirl.create :vehicle_class, :description=>"car"
    @vehicle_model = FactoryGirl.create :vehicle_model, :vehicle_class_id=>@vehichle_class_car
         .id
    @user=FactoryGirl.create :user
```

```ruby
    @car= FactoryGirl.create :vehicle, :vehicle_model=>@vehicle_model
    @car_gb=FactoryGirl.create :green_ebox, :vehicle => @car
    FactoryGirl.create :fare
    FactoryGirl.create :service_configuration
  end

  it "should list the reservations for the current user" do
    @reservation = FactoryGirl.create :reservation, :user=>@user

    get "index", :email=>@user.email,:password=>@user.password, :format=>"json"
    response.should be_success
    assigns(:reservations).should eq(@user.reservations)
  end

  it "should deny access to a wrong username/password combination" do
    get "index", :email=>@user.email,:password=>"wrongPassword", :format=>"json"
    response.should_not be_success
    get "index", :email=>"wrong_mail@gmc.it",
    :password=>@user.password, :format=>"json"
    response.should_not be_success
  end


  it "should understand the end time as a number" do
    @reservation=FactoryGirl.create :reservation, :user=>@user
    end_time=(Time.now + 1.day)
    put "update", :id=>@reservation.id, :format=>:json,  :email=>@user.email,:password=>@user.
         password, :reservation=> { :confirmed=>true, :end=>end_time.to_i }
    assigns(:reservation).should eq(@reservation)
    @reservation.reload
    #seems to be an issue here if I do not force reloading
    @reservation.confirmed.should be_true
    @reservation.end.to_i.should eq end_time.to_i
  end


  it "should allow a registered user to search for reservations" do

    Gps.create :latitude=>45.4811679,:longitude=>9.229389, :ts=>Time.now, :gb_id=>@car.
         green_ebox.id #this is needed for the position
    LastPosition.stubs(:near).returns(LastPosition.all)

    post "search", :email=>@user.email, :password=>@user.password,
     :address=>"via porpora, milano",:vehicle_type=>["car"], :format=>:json
    assigns(:reservation_proposals).should eq(@user.reservations.where(:confirmed=>false))
    assigns(:reservation_proposals).count.should eq 1
    @user.reservations.count.should eq 1
  end

end
```

— CHAPTER C —

GREEN MOVE APPLICATIONS DEVELOPMENT WORKFLOW

GMAs are just Android libraries. A minimal SDK is provided to aid their development, which consists in two libraries: The Android SDK is required to be installed on the development machine to code GMAs.

- The GMCodeLibrary library, which contains the GMComponent interface definition. GMAs must have a single entry point, which is a class implementing this interface. Also the GMContainer interface is provided within this library, so the GMA has a list of the methods exposed by the GMContainer.
- The T-Rex java client library, which contains class definitions required to use the T-Rex middle-ware.

The developer can use any number of classes and packages in its own code, given that everything needed for the application to work properly will be distributed in the same JAR file.

When the application is ready to be tested or distributed, the developer must export a JAR file (see figure C.1).

The next step is performed using utilities of the Android SDK, to convert the classes in the JAR file in the Dalvik VM bytecode.

Use the following commands:

- This command creates a file named *classes.dex* from the file *remoteClasses.jar*, which is the JAR file previously exported.

  ```
  dx --dex --output=classes.dex remoteClasses.jar
  ```

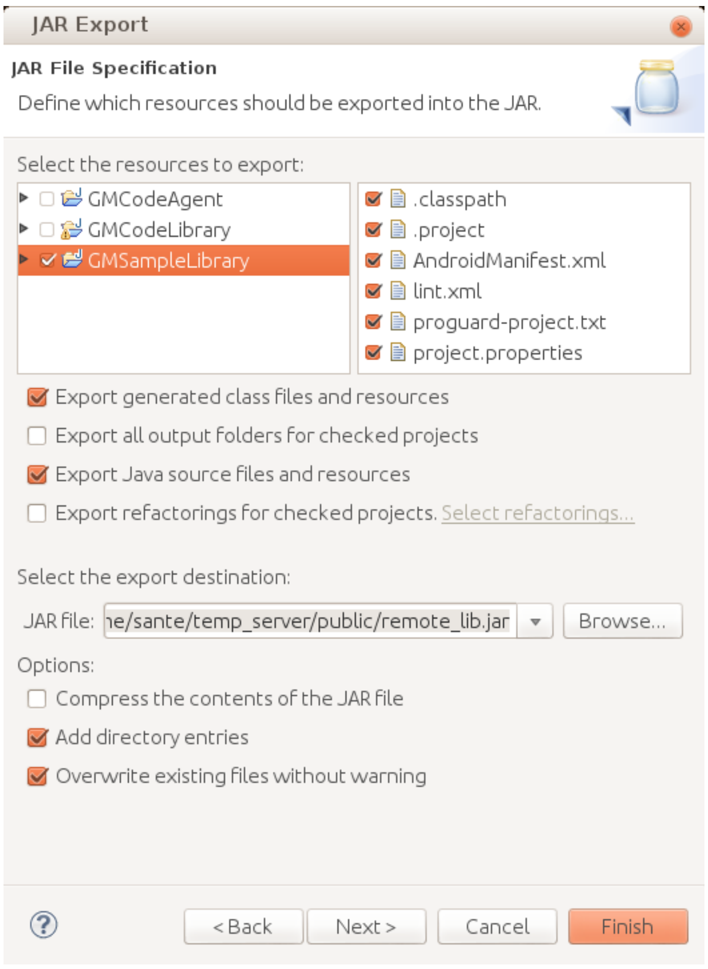
- This command creates a new JAR file, and adds the classes.dex file to the latter

  ```
  aapt add gma.jar classes.dex
  ```

The *gma.jar* file obtained is distributable as a GMA.

Figure C.2 and C.3 show the form for GMA upload on the GMC and the panel with the available options for distribution, once the GMA has been uploaded.

The distribution is handled by the daemon mode of the GMC, which currently uses the T-Rex middle-ware to deliver an experimental event (with type 1337) with the necessary attributes: url of the JAR file, class name of the entry point, action to be performed on the GMA.



JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- [ ] GMCodeAgent
- [ ] GMCodeLibrary
- [x] GMSampleLibrary

- [x] .classpath
- [x] .project
- [x] AndroidManifest.xml
- [x] lint.xml
- [x] proguard-project.txt
- [x] project.properties

- [x] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [x] Export Java source files and resources
- [ ] Export refactorings for checked projects. Select refactorings...

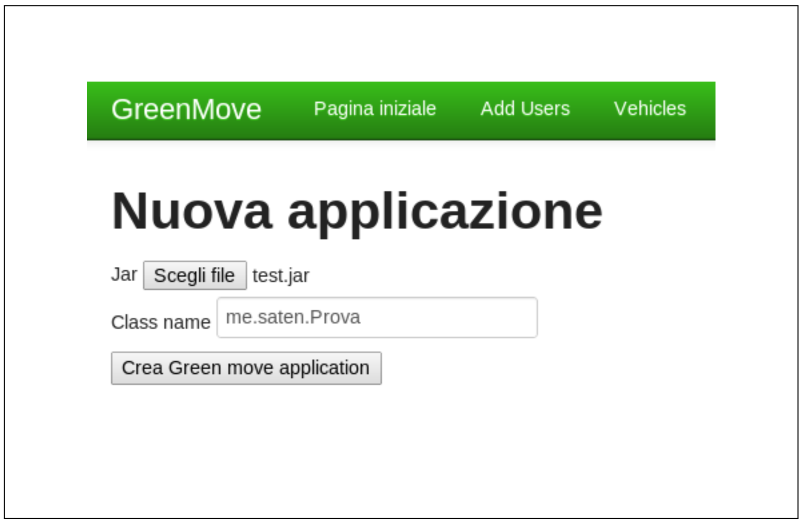
Select the export destination:

JAR file: ne/sante/temp_server/public/remote_lib.jar

Options:

- [ ] Compress the contents of the JAR file
- [x] Add directory entries
- [x] Overwrite existing files without warning

FIGURE C.1: Exporting a GMA with the Eclipse IDE.

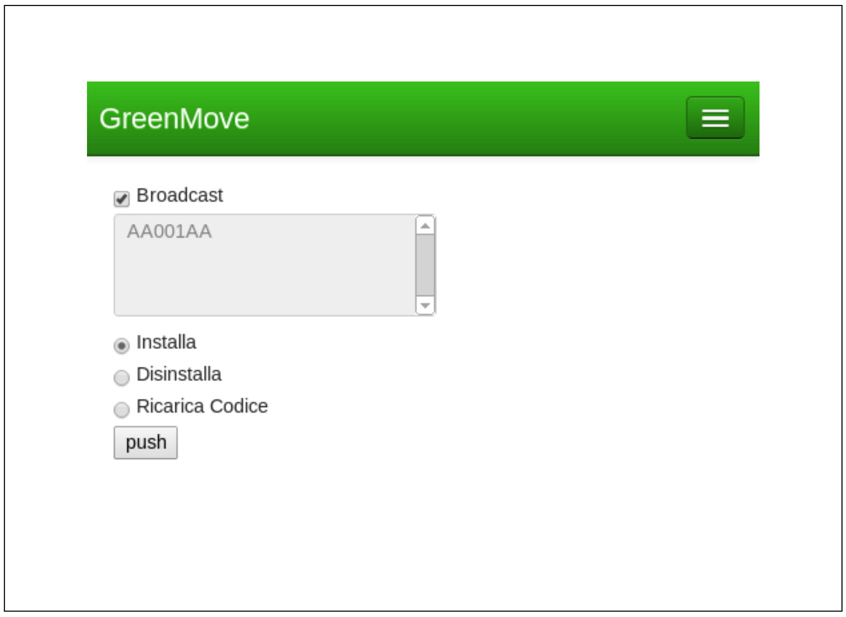


Nuova applicazione

Jar Scegli file test.jar

Class name me.saten.Prova

FIGURE C.2: A Green Move Application can be uploaded on the Admin section of the GMC.



☑ Broadcast

AA001AA

● Installa

○ Disinstalla

○ Ricarica Codice

FIGURE C.3: The GMC allows to install, reinstall or uninstall the GMA. Currently it only supports broadcasting o unicasting.

The daemon mode of the GMC requires the JRuby interpreter, and currently must be launched manually when needed (this is only to save RAM on the development virtual machine).

The command to start the daemon mode is the following:

```
rvm use jruby; rake jobs:work
```

The daemon mode processes a queue of jobs, which are just Ruby objects with a *perform* method.

— Chapter D —

chap:Database Schema

```
--
-- PostgreSQL database dump
--

SET statement_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;

--
-- Name: greenmove; Type: COMMENT; Schema: -; Owner: -
--

COMMENT ON DATABASE greenmove IS 'Green Move project DB';


--
-- Name: plpgsql; Type: EXTENSION; Schema: -; Owner: -
--

CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg_catalog;


--
-- Name: EXTENSION plpgsql; Type: COMMENT; Schema: -; Owner: -
--

COMMENT ON EXTENSION plpgsql IS 'PL/pgSQL procedural language';


SET search_path = public, pg_catalog;

--
-- Name: avail_asp_view_gen(timestamp without time zone); Type: FUNCTION; Schema: public;
     Owner: -
--

CREATE FUNCTION avail_asp_view_gen(start_point timestamp without time zone) RETURNS void
    LANGUAGE sql
    AS $_$DELETE FROM avail_asp_view;
INSERT INTO avail_asp_view(id, vehicle_id, begin_day, begin_hour, end_day, end_hour)
 SELECT a.id, a.vehicle_id, date_part('day'::text, a.begin_ts::timestamp with time zone - $1)
      ::integer AS startday, date_part('hour'::text, a.begin_ts)::integer AS starthour,
      date_part('day'::text, a.end_ts::timestamp with time zone - $1 )::integer AS endday,
      date_part('hour'::text, a.end_ts)::integer AS endhour
   FROM vehicle_availability a
  WHERE a.begin_ts IS NOT NULL AND a.end_ts IS NOT NULL AND
        (a.begin_ts >= $1 OR a.end_ts >= $1);$_$;


--
-- Name: onvehicleop_asp_view_gen(timestamp without time zone); Type: FUNCTION; Schema: public
     ; Owner: -
--

CREATE FUNCTION onvehicleop_asp_view_gen(start_point timestamp without time zone) RETURNS void
    LANGUAGE sql
    AS $_$DELETE FROM vehicleop_asp_view;
INSERT INTO vehicleop_asp_view(vehicle_id, begin_day, begin_hour, end_day, end_hour)
 SELECT o.vehicle_id, date_part('day'::text, o.begin_ts::timestamp with time zone - $1)::
      integer AS startday, date_part('hour'::text, o.begin_ts)::integer AS starthour,
      date_part('day'::text, o.end_ts::timestamp with time zone - $1 )::integer AS endday,
      date_part('hour'::text, o.end_ts)::integer AS endhour
   FROM on_vehicle_op o
  WHERE o.begin_ts IS NOT NULL AND o.end_ts IS NOT NULL AND
        (o.begin_ts >= $1 OR o.end_ts >= $1);$_$;


--
```

```
-- Name: precedences_asp_view_gen(timestamp without time zone); Type: FUNCTION; Schema: public
     ; Owner: -
--

CREATE FUNCTION precedences_asp_view_gen(start_point timestamp without time zone) RETURNS void
    LANGUAGE sql
    AS $_$DELETE FROM precedences_asp_view;
INSERT INTO precedences_asp_view(resrv_id_bf, resrv_id_af)
SELECT r1.id, r2.id
FROM reservation r1, reservation r2
WHERE r2.taking_ts =
        (SELECT MIN(r.taking_ts)
         FROM reservation r
         WHERE r.taking_ts > r1.release_ts)
        AND (r1.taking_ts > $1 OR r1.release_ts > $1)$_$;


--
-- Name: resrvasp_view_gen(timestamp without time zone); Type: FUNCTION; Schema: public; Owner
     : -
--

CREATE FUNCTION resrvasp_view_gen(start_point timestamp without time zone) RETURNS void
    LANGUAGE sql
    AS $_$DELETE FROM resrvasp_view;
INSERT INTO resrvasp_view(id, user_id, taking_day, taking_hour, release_day, release_hour,
     taking_location, release_location)
 SELECT r.id, r.user_id, date_part('day'::text, r.taking_ts::timestamp with time zone - $1)::
      integer AS startday, date_part('hour'::text, r.taking_ts)::integer AS starthour,
      date_part('day'::text, r.release_ts::timestamp with time zone - $1 )::integer AS endday,
       date_part('hour'::text, r.release_ts)::integer AS endhour, r.taking_position, r.
      release_position
   FROM reservation r
  WHERE r.taking_ts IS NOT NULL AND r.release_ts IS NOT NULL AND
        (r.taking_ts >= $1 OR r.release_ts >= $1);$_$;


SET default_tablespace = '';

SET default_with_oids = false;

--
-- Name: ad_provider_admin; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE ad_provider_admin (
    id character varying(50) NOT NULL,
    user_id character varying(16) NOT NULL
);


--
-- Name: admin; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE admin (
    id character varying(50) NOT NULL,
    user_id character varying(16)
);


--
-- Name: TABLE admin; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE admin IS 'Admin users list';


--
-- Name: age_class; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE age_class (
    id character varying(50) NOT NULL,
    lower_value character varying(255),
    upper_value character varying(255),
    class_name character varying(50)
);


```

```
--
-- Name: TABLE age_class; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE age_class IS 'Age classes for the Green Move project users';


--
-- Name: aggregation_point; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE aggregation_point (
    id integer NOT NULL,
    name character varying(255)
);


--
-- Name: TABLE aggregation_point; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE aggregation_point IS 'Aggregation points'' descriptions.';


--
-- Name: aggregation_point_class; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE aggregation_point_class (
    aggregation_point_id integer NOT NULL,
    aggregation_point_type integer NOT NULL
);


--
-- Name: TABLE aggregation_point_class; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE aggregation_point_class IS 'Classes for each aggregation point';


--
-- Name: aggregation_point_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE aggregation_point_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: aggregation_point_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE aggregation_point_id_seq OWNED BY aggregation_point.id;


--
-- Name: aggregation_point_type; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE aggregation_point_type (
    id integer NOT NULL,
    type character varying(255)
);


--
-- Name: TABLE aggregation_point_type; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE aggregation_point_type IS 'Types of aggregation points';


--
-- Name: aggregation_point_type_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--
```

```

CREATE SEQUENCE aggregation_point_type_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: aggregation_point_type_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE aggregation_point_type_id_seq OWNED BY aggregation_point_type.id;


--
-- Name: assignement; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE assignement (
    reservation_id integer NOT NULL,
    vehicle_id character varying(12),
    confirmed boolean
);


--
-- Name: assignment; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE assignment (
    reservation_id integer NOT NULL,
    vehicle_id character varying(12),
    confirmed boolean
);


--
-- Name: TABLE assignment; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE assignment IS 'Vehicle assigned to each reservation data';


--
-- Name: avail_asp_view; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE avail_asp_view (
    id integer NOT NULL,
    vehicle_id character varying(255),
    begin_day integer,
    begin_hour integer,
    end_day integer,
    end_hour integer
);


--
-- Name: TABLE avail_asp_view; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE avail_asp_view IS 'DO NOT FILL.
Automatically filled by function.';


--
-- Name: charge; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE charge (
    vehicle_id character varying(12) NOT NULL,
    begin_ts timestamp without time zone NOT NULL,
    end_ts timestamp without time zone,
    charge_station_id integer
);


--
```

```
-- Name: TABLE charge; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE charge IS 'Charge operations'' details';


--
-- Name: charge_station; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE charge_station (
    id integer NOT NULL,
    name character varying(255),
    latitude double precision,
    longitude double precision,
    avail_slots integer
);


--
-- Name: TABLE charge_station; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE charge_station IS 'List of all available charging stations';


--
-- Name: COLUMN charge_station.id; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON COLUMN charge_station.id IS '
';


--
-- Name: available_slots; Type: VIEW; Schema: public; Owner: -
--

CREATE VIEW available_slots AS
    (SELECT s.id, (s.avail_slots - count(*)) AS free FROM (charge_station s JOIN charge c ON
         ((c.charge_station_id = s.id))) WHERE (c.end_ts IS NULL) GROUP BY s.id, s.avail_slots
          ORDER BY s.id) UNION (SELECT s.id, s.avail_slots AS free FROM charge_station s WHERE
          (NOT (s.id IN (SELECT k.charge_station_id FROM charge k WHERE (k.end_ts IS NULL))))
         ORDER BY s.id);


--
-- Name: VIEW available_slots; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON VIEW available_slots IS 'Available charge stations'' slots.';


--
-- Name: battery; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE battery (
    id integer NOT NULL,
    current double precision,
    voltage double precision,
    vehicle_id character varying(12),
    charge integer
);


--
-- Name: TABLE battery; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE battery IS 'Vehicle batteries'' information.';


--
-- Name: battery_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE battery_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: battery_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE battery_id_seq OWNED BY battery.id;


--
-- Name: belongs_to; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE belongs_to (
    user_id character varying(16) NOT NULL,
    community_id integer NOT NULL
);


--
-- Name: TABLE belongs_to; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE belongs_to IS 'Users belonging to a community list';


--
-- Name: charge_station_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE charge_station_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: charge_station_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE charge_station_id_seq OWNED BY charge_station.id;


--
-- Name: gps; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE gps (
    ts timestamp without time zone NOT NULL,
    gb_id character varying(50) NOT NULL,
    latitude double precision,
    longitude double precision,
    speed double precision,
    satelittes integer,
    other_data text
);


--
-- Name: TABLE gps; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE gps IS 'Tables containing vehicles GPS positions'' log.';


--
-- Name: green_ebox; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE green_ebox (
    id character varying(50) NOT NULL,
    vehicle_id character varying(12)
);


```

```
--
-- Name: TABLE green_ebox; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE green_ebox IS 'Green e-Boxes'' related data';


--
-- Name: last_position_view; Type: VIEW; Schema: public; Owner: -
--

CREATE VIEW last_position_view AS
    SELECT g.gb_id, g.latitude, g.longitude FROM gps g WHERE (g.ts IN (SELECT max(gp.ts) AS ts
          FROM gps gp WHERE ((gp.gb_id)::text = (g.gb_id)::text) GROUP BY gp.gb_id));


--
-- Name: VIEW last_position_view; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON VIEW last_position_view IS 'Vehicle last position available (within 50 minutes).';


--
-- Name: mobile_device; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE mobile_device (
    id integer NOT NULL,
    user_id character varying(16),
    model character varying(255),
    type character varying(255)
);


--
-- Name: TABLE mobile_device; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE mobile_device IS 'Data about users'' personal devices.';


--
-- Name: reservation; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE reservation (
    id integer NOT NULL,
    taking_ts timestamp without time zone,
    release_ts timestamp without time zone,
    taking_position character varying(255),
    release_position character varying(255),
    vehicle_class_id integer,
    fare_id integer,
    confirmed boolean,
    planned_travel_distance double precision,
    service_configuration_id integer,
    user_id character varying(16)
);


--
-- Name: TABLE reservation; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE reservation IS 'Users'' reservations data';


--
-- Name: user_view; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE user_view (
    id character varying(16) NOT NULL,
    name character varying(255),
    surname character varying(255),
    class_name character varying(255),
    gender character varying(1),
    email character varying(255),
    customer boolean NOT NULL,
    owner boolean NOT NULL,
    username character varying(20),
    vat_info text,
    billing_info text
);


--
-- Name: TABLE user_view; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE user_view IS 'Table containing users'' data (customers'' and owners'' data)
      .';


--
-- Name: vehicle; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE vehicle (
    id character varying(12) NOT NULL,
    seats_number integer,
    insurance character varying(255),
    pub_key character varying(255),
    engine_type character varying(255),
    model_id integer,
    owner_id character varying(16)
);


--
-- Name: TABLE vehicle; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE vehicle IS 'Table containing vehicle related data';


--
-- Name: client_data_view; Type: VIEW; Schema: public; Owner: -
--

CREATE VIEW client_data_view AS
    SELECT m.id, u.class_name, u.gender, l.latitude, l.longitude FROM user_view u, reservation
          r, vehicle v, mobile_device m, green_ebox g, last_position_view l, assignment a
         WHERE ((((((m.user_id)::text = (u.id)::text) AND ((r.user_id)::text = (u.id)::text))
          AND (r.id = a.reservation_id)) AND ((a.vehicle_id)::text = (v.id)::text)) AND ((g.
         vehicle_id)::text = (v.id)::text)) AND ((l.gb_id)::text = (g.id)::text));


--
-- Name: VIEW client_data_view; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON VIEW client_data_view IS 'Last position related client data (user anonimous data)
      .';


--
-- Name: cloud_service; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE cloud_service (
    id integer NOT NULL,
    name character varying(255),
    description character varying(255)
);


--
-- Name: TABLE cloud_service; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE cloud_service IS 'placeholder for possibly offered cloud services...';


--
-- Name: cloud_service_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE cloud_service_id_seq
```

```
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: cloud_service_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE cloud_service_id_seq OWNED BY cloud_service.id;


--
-- Name: community; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE community (
    id integer NOT NULL,
    name character varying(255),
    type character varying(255)
);


--
-- Name: TABLE community; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE community IS 'Community of users data';


--
-- Name: community_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE community_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: community_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE community_id_seq OWNED BY community.id;


--
-- Name: vehicle_constructor; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE vehicle_constructor (
    id integer NOT NULL,
    name character varying(255),
    address character varying(255),
    vat_info text,
    phone character varying(255)
);


--
-- Name: TABLE vehicle_constructor; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE vehicle_constructor IS 'Table containing constructors'' references.';


--
-- Name: constructor_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE constructor_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: constructor_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE constructor_id_seq OWNED BY vehicle_constructor.id;


--
-- Name: delayed_jobs; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE delayed_jobs (
    id integer NOT NULL,
    priority integer DEFAULT 0,
    attempts integer DEFAULT 0,
    handler text,
    last_error text,
    run_at timestamp without time zone,
    locked_at timestamp without time zone,
    failed_at timestamp without time zone,
    locked_by character varying(255),
    queue character varying(255),
    created_at timestamp without time zone NOT NULL,
    updated_at timestamp without time zone NOT NULL
);


--
-- Name: delayed_jobs_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE delayed_jobs_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: delayed_jobs_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE delayed_jobs_id_seq OWNED BY delayed_jobs.id;


--
-- Name: failure; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE failure (
    id integer NOT NULL,
    description character varying(255)
);


--
-- Name: TABLE failure; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE failure IS 'Failures'' types.';


--
-- Name: failure_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE failure_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: failure_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
```

```
--

ALTER SEQUENCE failure_id_seq OWNED BY failure.id;


--
-- Name: fare; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE fare (
    id integer NOT NULL,
    type character varying(255),
    description text,
    price double precision,
    valid_from timestamp without time zone,
    valid_to timestamp without time zone
);


--
-- Name: TABLE fare; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE fare IS 'Fare list';


--
-- Name: fare_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE fare_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: fare_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE fare_id_seq OWNED BY fare.id;


--
-- Name: green_move_applications; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE green_move_applications (
    id integer NOT NULL,
    jar_file_name character varying(255),
    user_id character varying(255),
    class_name character varying(255),
    jar_signature character varying(255),
    created_at timestamp without time zone NOT NULL,
    updated_at timestamp without time zone NOT NULL
);


--
-- Name: green_move_applications_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE green_move_applications_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: green_move_applications_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE green_move_applications_id_seq OWNED BY green_move_applications.id;


--
```

```
-- Name: has_failure; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE has_failure (
    vehicle_id character varying(12) NOT NULL,
    failure_id integer NOT NULL,
    ts timestamp without time zone NOT NULL
);


--
-- Name: TABLE has_failure; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE has_failure IS 'Vehicles failures'' log.';


--
-- Name: interest_topic; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE interest_topic (
    id integer NOT NULL,
    name character varying(255),
    description text
);


--
-- Name: TABLE interest_topic; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE interest_topic IS 'Possible interest topics'' list.';


--
-- Name: interest_topic_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE interest_topic_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: interest_topic_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE interest_topic_id_seq OWNED BY interest_topic.id;


--
-- Name: interested; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE interested (
    ts timestamp without time zone NOT NULL,
    user_id character varying(16) NOT NULL,
    topic_id integer NOT NULL
);


--
-- Name: TABLE interested; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE interested IS 'Users related interests.';


--
-- Name: mobile_device_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE mobile_device_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: mobile_device_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE mobile_device_id_seq OWNED BY mobile_device.id;


--
-- Name: vehicle_model; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE vehicle_model (
    id integer NOT NULL,
    name character varying(255),
    constructor_id integer,
    vehicle_class_id integer
);


--
-- Name: TABLE vehicle_model; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE vehicle_model IS 'Table containing vehicle models'' descriptions';


--
-- Name: model_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE model_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: model_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE model_id_seq OWNED BY vehicle_model.id;


--
-- Name: nfc; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE nfc (
    tag_id character varying(255) NOT NULL,
    device_id integer,
    "check" character varying(255),
    data text
);


--
-- Name: TABLE nfc; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE nfc IS 'NFC tag related data.';


--
-- Name: on_vehicle_op; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE on_vehicle_op (
    vehicle_id character varying(12) NOT NULL,
    begin_ts timestamp without time zone NOT NULL,
    end_ts timestamp without time zone
);


--
```

```
-- Name: TABLE on_vehicle_op; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE on_vehicle_op IS 'Performed on-vehicle operations'' list.';


--
-- Name: op_type; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE op_type (
    id integer NOT NULL,
    description text
);


--
-- Name: TABLE op_type; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE op_type IS 'Available operations'' list';


--
-- Name: op_type_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE op_type_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: op_type_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE op_type_id_seq OWNED BY op_type.id;


--
-- Name: other_op; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE other_op (
    vehicle_id character varying(12) NOT NULL,
    begin_ts timestamp without time zone NOT NULL,
    end_ts timestamp without time zone,
    cause text,
    op_id integer
);


--
-- Name: TABLE other_op; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE other_op IS 'Other operations'' details';


--
-- Name: path_event; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE path_event (
    id integer NOT NULL,
    begin_ts timestamp without time zone,
    end_ts timestamp without time zone,
    latitude double precision,
    longitude double precision,
    cause character varying(255)
);


--
-- Name: TABLE path_event; Type: COMMENT; Schema: public; Owner: -
--

```

```
COMMENT ON TABLE path_event IS 'List of all path/road related issues (work in progress, ...)';


--
-- Name: path_event_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE path_event_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: path_event_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE path_event_id_seq OWNED BY path_event.id;


--
-- Name: precedences_asp_view; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE precedences_asp_view (
    resrv_id_bf integer NOT NULL,
    resrv_id_af integer NOT NULL
);


--
-- Name: TABLE precedences_asp_view; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE precedences_asp_view IS 'DO NOT FILL!
Automatically filled by function.';


--
-- Name: provided_by; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE provided_by (
    aggregation_poind_id integer NOT NULL,
    service_id integer NOT NULL
);


--
-- Name: TABLE provided_by; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE provided_by IS 'Aggregation points available services';


--
-- Name: request; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE request (
    reservation_id integer NOT NULL,
    service_id integer NOT NULL
);


--
-- Name: TABLE request; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE request IS 'Data about additional services included in the reservation';


--
-- Name: reservationASP_view; Type: VIEW; Schema: public; Owner: -
--

CREATE VIEW "reservationASP_view" AS
```

```
    SELECT r.user_id, (date_part('day'::text, ((r.taking_ts)::timestamp with time zone - (now
         () - '7 days'::interval))))::integer AS startday, (date_part('hour'::text, r.
         taking_ts))::integer AS starthour, (date_part('day'::text, ((r.release_ts)::timestamp
          with time zone - (now() - '7 days'::interval))))::integer AS endday, (date_part('
         hour'::text, r.release_ts))::integer AS endhour FROM reservation r WHERE ((r.
         taking_ts IS NOT NULL) AND (r.release_ts IS NOT NULL));


--
-- Name: reservation_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE reservation_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: reservation_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE reservation_id_seq OWNED BY reservation.id;


--
-- Name: resrvasp_view; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE resrvasp_view (
    id integer NOT NULL,
    taking_day integer,
    taking_hour integer,
    release_day integer,
    release_hour integer,
    user_id character varying(255),
    taking_location character varying(255),
    release_location character varying(255)
);


--
-- Name: TABLE resrvasp_view; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE resrvasp_view IS 'DO NOT FILL.
Automatically filled by function.';


--
-- Name: schema_migrations; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE schema_migrations (
    version character varying(255) NOT NULL
);


--
-- Name: sense; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE sense (
    sensor_id integer NOT NULL,
    failure_id integer NOT NULL,
    ts timestamp without time zone
);


--
-- Name: TABLE sense; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE sense IS 'Sensor ability to sense a failure.';


--
-- Name: sensor; Type: TABLE; Schema: public; Owner: -; Tablespace:
```

```
--

CREATE TABLE sensor (
    id integer NOT NULL,
    ts timestamp without time zone NOT NULL,
    gb_id character varying(50),
    value character varying(255),
    sensor_class integer
);


--
-- Name: TABLE sensor; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE sensor IS 'Table containing sensors retrieved data';


--
-- Name: sensor_class; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE sensor_class (
    id integer NOT NULL,
    value_type character varying(25),
    description text
);


--
-- Name: TABLE sensor_class; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE sensor_class IS 'Available classes of sensors.';


--
-- Name: sensor_class_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE sensor_class_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: sensor_class_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE sensor_class_id_seq OWNED BY sensor_class.id;


--
-- Name: sensor_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE sensor_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: sensor_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE sensor_id_seq OWNED BY sensor.id;


--
-- Name: service; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE service (
    id integer NOT NULL,
    name character varying(255),
    service_class_id integer
);


--
-- Name: TABLE service; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE service IS ’Available services’;


--
-- Name: service_class; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE service_class (
    id integer NOT NULL,
    description character varying(255)
);


--
-- Name: TABLE service_class; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE service_class IS ’Available classes of services’;


--
-- Name: service_class_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE service_class_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: service_class_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE service_class_id_seq OWNED BY service_class.id;


--
-- Name: service_configuration; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE service_configuration (
    id integer NOT NULL,
    name character varying(255),
    description text
);


--
-- Name: TABLE service_configuration; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE service_configuration IS ’Possible Green Move use-configurations’;


--
-- Name: service_configuration_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE service_configuration_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: service_configuration_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
```

```
--

ALTER SEQUENCE service_configuration_id_seq OWNED BY service_configuration.id;


--
-- Name: service_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE service_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: service_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE service_id_seq OWNED BY service.id;


--
-- Name: special_constraint; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE special_constraint (
    id integer NOT NULL,
    vehicle_id character varying(12),
    "constraint" text
);


--
-- Name: TABLE special_constraint; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE special_constraint IS 'Placeholder for special type o constraint';


--
-- Name: special_constraint_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE special_constraint_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: special_constraint_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE special_constraint_id_seq OWNED BY special_constraint.id;


--
-- Name: static_context; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE static_context (
    ts timestamp without time zone NOT NULL,
    user_id character varying(16) NOT NULL
);


--
-- Name: TABLE static_context; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE static_context IS 'Data about user static context.';


--
-- Name: supports; Type: TABLE; Schema: public; Owner: -; Tablespace:
```

```
--

CREATE TABLE supports (
    device_id integer NOT NULL,
    cloud_service_id integer NOT NULL
);


--
-- Name: TABLE supports; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE supports IS 'List of device supported cloud services';


--
-- Name: usable_in; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE usable_in (
    service_configuration_id integer NOT NULL,
    service_id integer NOT NULL
);


--
-- Name: TABLE usable_in; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE usable_in IS 'Specific service configuration available services.';


--
-- Name: usage_data; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE usage_data (
    reservation_id integer NOT NULL,
    real_taking_ts timestamp without time zone,
    real_release_ts timestamp without time zone,
    taking_km double precision,
    release_km double precision,
    constr_malus double precision,
    constr_bonus double precision,
    damage_malus double precision,
    charge_bonus double precision,
    release_bonus double precision
);


--
-- Name: TABLE usage_data; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE usage_data IS 'Reservation related vehicles'' usage data';


--
-- Name: user; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE "user" (
    id character varying(16) NOT NULL,
    name character varying(255),
    surname character varying(255),
    email character varying(255),
    pub_key character varying(255),
    ident_url character varying(255),
    username character varying(20),
    passwd character varying(150),
    vat_info text DEFAULT 'none'::text,
    billing_info text DEFAULT 'none'::text,
    customer boolean NOT NULL,
    owner boolean NOT NULL,
    birthdate date,
    gender character varying(1)
);


--
```

```
-- Name: TABLE "user"; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE "user" IS 'Table containing users'' data (customers'' and owners'' data).';


--
-- Name: vehicle_availability; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE vehicle_availability (
    id integer NOT NULL,
    vehicle_id character varying(12),
    begin_ts timestamp without time zone,
    end_ts timestamp without time zone
);


--
-- Name: TABLE vehicle_availability; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE vehicle_availability IS 'Table containing eventual constraints on vehicle
     availability';


--
-- Name: vehicle_availability_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE vehicle_availability_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: vehicle_availability_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE vehicle_availability_id_seq OWNED BY vehicle_availability.id;


--
-- Name: vehicle_class; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE vehicle_class (
    id integer NOT NULL,
    description character varying(255)
);


--
-- Name: TABLE vehicle_class; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE vehicle_class IS 'Classes of vehicles';


--
-- Name: vehicle_class_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE vehicle_class_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: vehicle_class_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE vehicle_class_id_seq OWNED BY vehicle_class.id;

```

```

--
-- Name: vehicle_constructor_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE vehicle_constructor_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: vehicle_constructor_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE vehicle_constructor_id_seq OWNED BY vehicle_constructor.id;


--
-- Name: vehicle_model_id_seq; Type: SEQUENCE; Schema: public; Owner: -
--

CREATE SEQUENCE vehicle_model_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


--
-- Name: vehicle_model_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: -
--

ALTER SEQUENCE vehicle_model_id_seq OWNED BY vehicle_model.id;


--
-- Name: vehicleop_asp_view; Type: TABLE; Schema: public; Owner: -; Tablespace:
--

CREATE TABLE vehicleop_asp_view (
    vehicle_id character varying(255) NOT NULL,
    begin_day integer NOT NULL,
    begin_hour integer NOT NULL,
    end_day integer NOT NULL,
    end_hour integer NOT NULL
);


--
-- Name: TABLE vehicleop_asp_view; Type: COMMENT; Schema: public; Owner: -
--

COMMENT ON TABLE vehicleop_asp_view IS 'DO NOT FILL.
Automatically filled by function.';


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY aggregation_point ALTER COLUMN id SET DEFAULT nextval('
      aggregation_point_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY aggregation_point_type ALTER COLUMN id SET DEFAULT nextval('
      aggregation_point_type_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

```

```
ALTER TABLE ONLY battery ALTER COLUMN id SET DEFAULT nextval('battery_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY charge_station ALTER COLUMN id SET DEFAULT nextval('charge_station_id_seq'::
      regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY cloud_service ALTER COLUMN id SET DEFAULT nextval('cloud_service_id_seq'::
      regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY community ALTER COLUMN id SET DEFAULT nextval('community_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY delayed_jobs ALTER COLUMN id SET DEFAULT nextval('delayed_jobs_id_seq'::
      regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY failure ALTER COLUMN id SET DEFAULT nextval('failure_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY fare ALTER COLUMN id SET DEFAULT nextval('fare_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY green_move_applications ALTER COLUMN id SET DEFAULT nextval('
      green_move_applications_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY interest_topic ALTER COLUMN id SET DEFAULT nextval('interest_topic_id_seq'::
      regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY mobile_device ALTER COLUMN id SET DEFAULT nextval('mobile_device_id_seq'::
      regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY op_type ALTER COLUMN id SET DEFAULT nextval('op_type_id_seq'::regclass);


--
```

```
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY path_event ALTER COLUMN id SET DEFAULT nextval('path_event_id_seq'::regclass)
      ;


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY reservation ALTER COLUMN id SET DEFAULT nextval('reservation_id_seq'::
      regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY sensor ALTER COLUMN id SET DEFAULT nextval('sensor_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY sensor_class ALTER COLUMN id SET DEFAULT nextval('sensor_class_id_seq'::
      regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY service ALTER COLUMN id SET DEFAULT nextval('service_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY service_class ALTER COLUMN id SET DEFAULT nextval('service_class_id_seq'::
      regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY service_configuration ALTER COLUMN id SET DEFAULT nextval('
      service_configuration_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY special_constraint ALTER COLUMN id SET DEFAULT nextval('
      special_constraint_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY vehicle_availability ALTER COLUMN id SET DEFAULT nextval('
      vehicle_availability_id_seq'::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY vehicle_class ALTER COLUMN id SET DEFAULT nextval('vehicle_class_id_seq'::
      regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--
```

```

ALTER TABLE ONLY vehicle_constructor ALTER COLUMN id SET DEFAULT nextval('constructor_id_seq
      '::regclass);


--
-- Name: id; Type: DEFAULT; Schema: public; Owner: -
--

ALTER TABLE ONLY vehicle_model ALTER COLUMN id SET DEFAULT nextval('model_id_seq'::regclass);


--
-- Name: admin_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY admin
    ADD CONSTRAINT admin_pkey PRIMARY KEY (id);


--
-- Name: age_class_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY age_class
    ADD CONSTRAINT age_class_pkey PRIMARY KEY (id);


--
-- Name: aggregation_point_class_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY aggregation_point_class
    ADD CONSTRAINT aggregation_point_class_pkey PRIMARY KEY (aggregation_point_id,
         aggregation_point_type);


--
-- Name: aggregation_point_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY aggregation_point
    ADD CONSTRAINT aggregation_point_pkey PRIMARY KEY (id);


--
-- Name: aggregation_point_type_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY aggregation_point_type
    ADD CONSTRAINT aggregation_point_type_pkey PRIMARY KEY (id);


--
-- Name: avail_asp_view_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY avail_asp_view
    ADD CONSTRAINT avail_asp_view_pkey PRIMARY KEY (id);


--
-- Name: battery_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY battery
    ADD CONSTRAINT battery_pkey PRIMARY KEY (id);


--
-- Name: belongs_to_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY belongs_to
    ADD CONSTRAINT belongs_to_pkey PRIMARY KEY (user_id, community_id);


--
-- Name: charge_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--
```

```

ALTER TABLE ONLY charge
    ADD CONSTRAINT charge_pkey PRIMARY KEY (vehicle_id, begin_ts);


--
-- Name: charge_station_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY charge_station
    ADD CONSTRAINT charge_station_pkey PRIMARY KEY (id);


--
-- Name: cloud_service_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY cloud_service
    ADD CONSTRAINT cloud_service_pkey PRIMARY KEY (id);


--
-- Name: community_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY community
    ADD CONSTRAINT community_pkey PRIMARY KEY (id);


--
-- Name: delayed_jobs_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY delayed_jobs
    ADD CONSTRAINT delayed_jobs_pkey PRIMARY KEY (id);


--
-- Name: failure_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY failure
    ADD CONSTRAINT failure_pkey PRIMARY KEY (id);


--
-- Name: fare_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY fare
    ADD CONSTRAINT fare_pkey PRIMARY KEY (id);


--
-- Name: gps_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY gps
    ADD CONSTRAINT gps_pkey PRIMARY KEY (ts, gb_id);


--
-- Name: green_ebox_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY green_ebox
    ADD CONSTRAINT green_ebox_pkey PRIMARY KEY (id);


--
-- Name: green_move_applications_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY green_move_applications
    ADD CONSTRAINT green_move_applications_pkey PRIMARY KEY (id);


--
-- Name: has_failure_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--
```

```

ALTER TABLE ONLY has_failure
    ADD CONSTRAINT has_failure_pkey PRIMARY KEY (vehicle_id, failure_id, ts);


--
-- Name: interest_topic_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY interest_topic
    ADD CONSTRAINT interest_topic_pkey PRIMARY KEY (id);


--
-- Name: interested_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY interested
    ADD CONSTRAINT interested_pkey PRIMARY KEY (ts, user_id, topic_id);


--
-- Name: mobile_device_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY mobile_device
    ADD CONSTRAINT mobile_device_pkey PRIMARY KEY (id);


--
-- Name: nfc_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY nfc
    ADD CONSTRAINT nfc_pkey PRIMARY KEY (tag_id);


--
-- Name: on_vehicle_op_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY on_vehicle_op
    ADD CONSTRAINT on_vehicle_op_pkey PRIMARY KEY (vehicle_id, begin_ts);


--
-- Name: onvehicleop_asp_view_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY vehicleop_asp_view
    ADD CONSTRAINT onvehicleop_asp_view_pkey PRIMARY KEY (vehicle_id, begin_day, begin_hour,
         end_day, end_hour);


--
-- Name: op_type_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY op_type
    ADD CONSTRAINT op_type_pkey PRIMARY KEY (id);


--
-- Name: other_op_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY other_op
    ADD CONSTRAINT other_op_pkey PRIMARY KEY (vehicle_id, begin_ts);


--
-- Name: path_event_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY path_event
    ADD CONSTRAINT path_event_pkey PRIMARY KEY (id);


--
-- Name: pkey_ad_provider_admin; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
```

```
--

ALTER TABLE ONLY ad_provider_admin
    ADD CONSTRAINT pkey_ad_provider_admin PRIMARY KEY (id, user_id);


--
-- Name: precedences_asp_view_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY precedences_asp_view
    ADD CONSTRAINT precedences_asp_view_pkey PRIMARY KEY (resrv_id_bf, resrv_id_af);


--
-- Name: provided_by_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY provided_by
    ADD CONSTRAINT provided_by_pkey PRIMARY KEY (aggregation_poind_id, service_id);


--
-- Name: request_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY request
    ADD CONSTRAINT request_pkey PRIMARY KEY (reservation_id, service_id);


--
-- Name: reservation_id; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY assignment
    ADD CONSTRAINT reservation_id PRIMARY KEY (reservation_id);


--
-- Name: reservation_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY reservation
    ADD CONSTRAINT reservation_pkey PRIMARY KEY (id);


--
-- Name: resrvASP_view_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY resrvasp_view
    ADD CONSTRAINT "resrvASP_view_pkey" PRIMARY KEY (id);


--
-- Name: sense_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY sense
    ADD CONSTRAINT sense_pkey PRIMARY KEY (sensor_id, failure_id);


--
-- Name: sensor_class_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY sensor_class
    ADD CONSTRAINT sensor_class_pkey PRIMARY KEY (id);


--
-- Name: sensor_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY sensor
    ADD CONSTRAINT sensor_pkey PRIMARY KEY (id);


--
-- Name: service_class_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
```

```
--

ALTER TABLE ONLY service_class
    ADD CONSTRAINT service_class_pkey PRIMARY KEY (id);


--
-- Name: service_configuration_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY service_configuration
    ADD CONSTRAINT service_configuration_pkey PRIMARY KEY (id);


--
-- Name: service_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY service
    ADD CONSTRAINT service_pkey PRIMARY KEY (id);


--
-- Name: special_constraint_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY special_constraint
    ADD CONSTRAINT special_constraint_pkey PRIMARY KEY (id);


--
-- Name: static_context_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY static_context
    ADD CONSTRAINT static_context_pkey PRIMARY KEY (ts, user_id);


--
-- Name: supports_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY supports
    ADD CONSTRAINT supports_pkey PRIMARY KEY (device_id, cloud_service_id);


--
-- Name: usable_in_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY usable_in
    ADD CONSTRAINT usable_in_pkey PRIMARY KEY (service_configuration_id, service_id);


--
-- Name: usage_data_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY usage_data
    ADD CONSTRAINT usage_data_pkey PRIMARY KEY (reservation_id);


--
-- Name: user_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY "user"
    ADD CONSTRAINT user_pkey PRIMARY KEY (id);


--
-- Name: user_view_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY user_view
    ADD CONSTRAINT user_view_pkey PRIMARY KEY (id);


--
-- Name: vehicle_availability_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
```

```
--

ALTER TABLE ONLY vehicle_availability
    ADD CONSTRAINT vehicle_availability_pkey PRIMARY KEY (id);


--
-- Name: vehicle_class_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY vehicle_class
    ADD CONSTRAINT vehicle_class_pkey PRIMARY KEY (id);


--
-- Name: vehicle_constructor_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY vehicle_constructor
    ADD CONSTRAINT vehicle_constructor_pkey PRIMARY KEY (id);


--
-- Name: vehicle_model_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY vehicle_model
    ADD CONSTRAINT vehicle_model_pkey PRIMARY KEY (id);


--
-- Name: vehicle_pkey; Type: CONSTRAINT; Schema: public; Owner: -; Tablespace:
--

ALTER TABLE ONLY vehicle
    ADD CONSTRAINT vehicle_pkey PRIMARY KEY (id);


--
-- Name: delayed_jobs_priority; Type: INDEX; Schema: public; Owner: -; Tablespace:
--

CREATE INDEX delayed_jobs_priority ON delayed_jobs USING btree (priority, run_at);


--
-- Name: fki_vehicle_model_fkey; Type: INDEX; Schema: public; Owner: -; Tablespace:
--

CREATE INDEX fki_vehicle_model_fkey ON vehicle USING btree (model_id);


--
-- Name: fki_vehicle_user_fkey; Type: INDEX; Schema: public; Owner: -; Tablespace:
--

CREATE INDEX fki_vehicle_user_fkey ON vehicle USING btree (owner_id);


--
-- Name: index_green_move_applications_on_user_id; Type: INDEX; Schema: public; Owner: -;
     Tablespace:
--

CREATE INDEX index_green_move_applications_on_user_id ON green_move_applications USING btree (
     user_id);


--
-- Name: unique_schema_migrations; Type: INDEX; Schema: public; Owner: -; Tablespace:
--

CREATE UNIQUE INDEX unique_schema_migrations ON schema_migrations USING btree (version);


--
-- Name: admin_user_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY admin
```

```
    ADD CONSTRAINT admin_user_fkey FOREIGN KEY (user_id) REFERENCES "user"(id);


--
-- Name: aggregation_point_class_aggregation_point_fkey; Type: FK CONSTRAINT; Schema: public;
     Owner: -
--

ALTER TABLE ONLY aggregation_point_class
    ADD CONSTRAINT aggregation_point_class_aggregation_point_fkey FOREIGN KEY (
         aggregation_point_id) REFERENCES aggregation_point(id) ON UPDATE CASCADE ON DELETE
         RESTRICT;


--
-- Name: aggregation_point_class_aggregation_point_type_fkey; Type: FK CONSTRAINT; Schema:
     public; Owner: -
--

ALTER TABLE ONLY aggregation_point_class
    ADD CONSTRAINT aggregation_point_class_aggregation_point_type_fkey FOREIGN KEY (
         aggregation_point_type) REFERENCES aggregation_point_type(id) ON UPDATE CASCADE ON
         DELETE RESTRICT;


--
-- Name: assignement_reservation_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY assignment
    ADD CONSTRAINT assignement_reservation_fkey FOREIGN KEY (reservation_id) REFERENCES
         reservation(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: assignement_vehicle_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY assignment
    ADD CONSTRAINT assignement_vehicle_fkey FOREIGN KEY (vehicle_id) REFERENCES vehicle(id) ON
          UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: avail_asp_view_vehicle_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY avail_asp_view
    ADD CONSTRAINT avail_asp_view_vehicle_fkey FOREIGN KEY (vehicle_id) REFERENCES vehicle(id)
          ON UPDATE CASCADE ON DELETE CASCADE;


--
-- Name: battery_vehicle_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY battery
    ADD CONSTRAINT battery_vehicle_fkey FOREIGN KEY (vehicle_id) REFERENCES vehicle(id) ON
         UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: belongs_to_community_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY belongs_to
    ADD CONSTRAINT belongs_to_community_fkey FOREIGN KEY (community_id) REFERENCES community(
         id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: belongs_to_user_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY belongs_to
    ADD CONSTRAINT belongs_to_user_fkey FOREIGN KEY (user_id) REFERENCES "user"(id) ON UPDATE
         CASCADE ON DELETE RESTRICT;


--
```

```sql
-- Name: charge_charge_station_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY charge
    ADD CONSTRAINT charge_charge_station_fkey FOREIGN KEY (charge_station_id) REFERENCES
         charge_station(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: fkey_user; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY ad_provider_admin
    ADD CONSTRAINT fkey_user FOREIGN KEY (user_id) REFERENCES "user"(id);


--
-- Name: gps_green_ebox_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY gps
    ADD CONSTRAINT gps_green_ebox_fkey FOREIGN KEY (gb_id) REFERENCES green_ebox(id);


--
-- Name: green_ebox_vehicle_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY green_ebox
    ADD CONSTRAINT green_ebox_vehicle_fkey FOREIGN KEY (vehicle_id) REFERENCES vehicle(id) ON
         UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: has_failure_failure_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY has_failure
    ADD CONSTRAINT has_failure_failure_fkey FOREIGN KEY (failure_id) REFERENCES failure(id) ON
          UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: has_failure_vehicle_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY has_failure
    ADD CONSTRAINT has_failure_vehicle_fkey FOREIGN KEY (vehicle_id) REFERENCES vehicle(id) ON
          UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: interested_interest_topic_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY interested
    ADD CONSTRAINT interested_interest_topic_fkey FOREIGN KEY (topic_id) REFERENCES
         interest_topic(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: interested_static_context_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY interested
    ADD CONSTRAINT interested_static_context_fkey FOREIGN KEY (ts, user_id) REFERENCES
         static_context(ts, user_id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: mobile_device_user_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY mobile_device
    ADD CONSTRAINT mobile_device_user_fkey FOREIGN KEY (user_id) REFERENCES "user"(id) ON
         UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: model_constructor_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
```

```
--

ALTER TABLE ONLY vehicle_model
    ADD CONSTRAINT model_constructor_fkey FOREIGN KEY (constructor_id) REFERENCES
         vehicle_constructor(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: model_vehicle_class_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY vehicle_model
    ADD CONSTRAINT model_vehicle_class_fkey FOREIGN KEY (vehicle_class_id) REFERENCES
         vehicle_class(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: nfc_mobile_device_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY nfc
    ADD CONSTRAINT nfc_mobile_device_fkey FOREIGN KEY (device_id) REFERENCES mobile_device(id)
          ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: on_vehicle_op_vehicle_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY on_vehicle_op
    ADD CONSTRAINT on_vehicle_op_vehicle_fkey FOREIGN KEY (vehicle_id) REFERENCES vehicle(id)
         ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: onvehicleop_asp_view_vehicle_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY vehicleop_asp_view
    ADD CONSTRAINT onvehicleop_asp_view_vehicle_fkey FOREIGN KEY (vehicle_id) REFERENCES
         vehicle(id) ON UPDATE CASCADE ON DELETE CASCADE;


--
-- Name: other_op_op_typpe_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY other_op
    ADD CONSTRAINT other_op_op_typpe_fkey FOREIGN KEY (op_id) REFERENCES op_type(id) ON UPDATE
          CASCADE ON DELETE RESTRICT;


--
-- Name: precedences_asp_view_resrv1; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY precedences_asp_view
    ADD CONSTRAINT precedences_asp_view_resrv1 FOREIGN KEY (resrv_id_bf) REFERENCES
         reservation(id) ON UPDATE CASCADE ON DELETE CASCADE;


--
-- Name: precedences_asp_view_resrv2; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY precedences_asp_view
    ADD CONSTRAINT precedences_asp_view_resrv2 FOREIGN KEY (resrv_id_af) REFERENCES
         reservation(id) ON UPDATE CASCADE ON DELETE CASCADE;


--
-- Name: provided_by_aggregation_point_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY provided_by
    ADD CONSTRAINT provided_by_aggregation_point_fkey FOREIGN KEY (aggregation_poind_id)
         REFERENCES aggregation_point(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
```

```
-- Name: provided_by_service_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY provided_by
    ADD CONSTRAINT provided_by_service_fkey FOREIGN KEY (service_id) REFERENCES service(id) ON
          UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: request_reservation_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY request
    ADD CONSTRAINT request_reservation_fkey FOREIGN KEY (reservation_id) REFERENCES
         reservation(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: request_service_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY request
    ADD CONSTRAINT request_service_fkey FOREIGN KEY (service_id) REFERENCES service(id) ON
         UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: reservation_fare_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY reservation
    ADD CONSTRAINT reservation_fare_fkey FOREIGN KEY (fare_id) REFERENCES fare(id) ON UPDATE
         CASCADE ON DELETE RESTRICT;


--
-- Name: reservation_service_configuration_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY reservation
    ADD CONSTRAINT reservation_service_configuration_fkey FOREIGN KEY (
         service_configuration_id) REFERENCES service_configuration(id) ON UPDATE CASCADE ON
         DELETE RESTRICT;


--
-- Name: reservation_user_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY reservation
    ADD CONSTRAINT reservation_user_fkey FOREIGN KEY (user_id) REFERENCES "user"(id) ON UPDATE
          CASCADE ON DELETE RESTRICT;


--
-- Name: reservation_vehicle_class_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY reservation
    ADD CONSTRAINT reservation_vehicle_class_fkey FOREIGN KEY (vehicle_class_id) REFERENCES
         vehicle_class(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: resrvASP_view_reservation_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY resrvasp_view
    ADD CONSTRAINT "resrvASP_view_reservation_fkey" FOREIGN KEY (id) REFERENCES reservation(id
         ) ON UPDATE CASCADE ON DELETE CASCADE;


--
-- Name: resrvASP_view_user; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY resrvasp_view
    ADD CONSTRAINT "resrvASP_view_user" FOREIGN KEY (user_id) REFERENCES "user"(id) ON UPDATE
         CASCADE ON DELETE CASCADE;

```

```

--
-- Name: sense_failure_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY sense
    ADD CONSTRAINT sense_failure_fkey FOREIGN KEY (failure_id) REFERENCES failure(id) ON
         UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: sensor_green_ebox_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY sensor
    ADD CONSTRAINT sensor_green_ebox_fkey FOREIGN KEY (gb_id) REFERENCES green_ebox(id) ON
         UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: sensor_sensor_class_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY sensor
    ADD CONSTRAINT sensor_sensor_class_fkey FOREIGN KEY (sensor_class) REFERENCES sensor_class
         (id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: service_service_class_id; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY service
    ADD CONSTRAINT service_service_class_id FOREIGN KEY (service_class_id) REFERENCES
         service_class(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: special_constraint_vehicle_pkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY special_constraint
    ADD CONSTRAINT special_constraint_vehicle_pkey FOREIGN KEY (vehicle_id) REFERENCES vehicle
         (id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: static_context_user_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY static_context
    ADD CONSTRAINT static_context_user_fkey FOREIGN KEY (user_id) REFERENCES "user"(id) ON
         UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: supports_cloud_service_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY supports
    ADD CONSTRAINT supports_cloud_service_fkey FOREIGN KEY (cloud_service_id) REFERENCES
         cloud_service(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: supports_mobile_device_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY supports
    ADD CONSTRAINT supports_mobile_device_fkey FOREIGN KEY (device_id) REFERENCES
         mobile_device(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: usable_in_service_configuration_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY usable_in
    ADD CONSTRAINT usable_in_service_configuration_fkey FOREIGN KEY (service_configuration_id)
          REFERENCES service_configuration(id) ON UPDATE CASCADE ON DELETE RESTRICT;
```

```
--
-- Name: usable_in_service_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY usable_in
    ADD CONSTRAINT usable_in_service_fkey FOREIGN KEY (service_id) REFERENCES service(id) ON
         UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: usage_data_reservation_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY usage_data
    ADD CONSTRAINT usage_data_reservation_fkey FOREIGN KEY (reservation_id) REFERENCES
         reservation(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: vehicle_availability_vehicle_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY vehicle_availability
    ADD CONSTRAINT vehicle_availability_vehicle_fkey FOREIGN KEY (vehicle_id) REFERENCES
         vehicle(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- Name: vehicle_model_fkey; Type: FK CONSTRAINT; Schema: public; Owner: -
--

ALTER TABLE ONLY vehicle
    ADD CONSTRAINT vehicle_model_fkey FOREIGN KEY (model_id) REFERENCES vehicle_model(id);


--
-- PostgreSQL database dump complete
--

INSERT INTO schema_migrations (version) VALUES ('0');

INSERT INTO schema_migrations (version) VALUES ('20121007130334');

INSERT INTO schema_migrations (version) VALUES ('20121007152713');
```

Bibliographic references

Car2Go. http://www.car2go.com.

Cugola, Gianpaolo, & Margara, Alessandro. 2010. *Tesla: A formally defined event specification language*.

Katzev, R. 2003. *Car sharing: A new approach to urban transportation problems*.

RelayRides, Google. https://relayrides.com.

Sharing, Mobility Car. http://www.mobility.ch/de/pub.

VirtualBox, Oracle. https://www.virtualbox.org.

Yelòmobile. http://www.yelomobile.fr.

Zipcar. http://www.zipcar.com.