Università degli Studi di Cagliari

DOTTORATO DI RICERCA

INGEGNERIA ELETTRONICA ED INFORMATICA

XXIX Ciclo

CLACSOON: CARPOOLING IN URBAN AREAS

Settore scientifico disciplinare di afferenza

ING-INF/03 Telecomunicazioni

Presentata da: **Matteo Mallus**

Coordinatore Dottorato **Prof. Fabio Roli**

Tutor **Prof. Maurizio Murroni**

Esame finale anno accademico 2015 – 2016

Tesi discussa nella sessione d'esame marzo – aprile 2017

Contents

List of Figures

List of Tables

Abstract

The research presented in this thesis focuses on ICT technologies for urban mobility, with particular attention to the study, design and the development of applications for dynamic carpooling services in smart cities. The work has been done within the PhD programme in apprenticeship, during the development of the CLACSOON project while working for the startup company GreenShare SRL.

Nowadays, the development of a Smart City represents an approved way to improve the quality of life in the urban context. The sustainable transport is a key service for a smart city, and carpooling solutions have gained more and more popularity in the last years. This thesis mainly focuses on the challenges of a real-time carpooling service. A lot of effort has been made on the design and the development of a Cloud-Mobile Dynamic Ridesharing platform named "CLACSOON". The proposed solution is an application that automatizes the arrangement of the shared ride, automatically notifying the presence of suitable travel companions and suggesting the pick-up points and the meeting times. The pick-up and drop-off point are computed by a route matching algorithm that aim to introduce two novel features for a ridesharing application. The former is the partial ridesharing, according to which the riders can walk to reach the driver along his/her route when driving to the destination. The latter consists in the possibility to share the ride when the driver has already started the ride by modeling the mobility to reach the driver destination. To evaluate performances of the service and the Quality of Experience (QoE) provided to the users, an emulation system has been implemented to evaluate the performance of the CLACSOON platform. The performances, expressed in terms of Key Performance Indicators (KPI), show that introducing these features in a route matching algorithm leads to a substantial performances improvement; moreover, the results can be considered to evaluate the requirements to build a successful urban carpooling service. Additionally, the *carpoolers*' cars, with the coming of the IoT, could be seen as a formidable sensor platform. Since one of the aspects of smart cities is the optimal use of the resources, the information coming from these sensors can be processed and analyzed to improve the efficiency and quality of the urban services and to support smart transportation and sustainable mobility. In this context, Machine to Machine (M2M) and Device to Device (D2D) communications play an important role. Starting from this vision, a distributed algorithm for the task allocation and assignment, which can be executed by a group of real IoT devices, has been developed with the aim of maximizing the lifetime of groups of nodes involved while ensuring the fulfillment of the requested Quality of Information (QoI) requirement. In the field of infomobility and support of the urban mobility, it could be run in real devices on the vehicles .

Chapter 1

Introduction

In the last decades, the Smart City paradigm is emerging as a way to improve the quality of life in the urban context, while mitigating the problems generated by the population increase [1]. In future Smart Cities, the quality of life will be mostly determined by the availability of sustainable and environmental-friendly mobility services. Nowadays a fully sustainable transport model is not available indeed , and cities all over the world are facing the challenge of reducing greenhouse gas emissions and air pollution that results from the vehicular traffic congestion. Sustainable transport is a broad subject, because a transportation mean can be seen as sustainable in a social, or environmental, or climate sense.[2]. Nowadays the majority of the trips is made with the use of personal cars: the daily urban mobility that is made from individuals going to work leads to relevant problems, starting from the congestion of the urban roads, to the car occupancy of the parking, as well as the pollution generated from the use of the personal cars. The growing sense of responsibility in respect of the environment is contributing in determining the decreasing of the use of the car as a primary mode of transit, in favour of alternative and sustainable modes. Different alternative transport modes had been implemented for reducing the air pollution: in particular, public transport services have received a great attention in this respect. Public transport infrastructure is for sure one of the best solutions to face the challenge of the vehicular traffic congestion, but still many people prefer the use of the personal car (or other personal transportation means) over public transportation. A report by the US Environmental Protection Agency revealed that light-duty vehicles are the source of nearly 25% of the country's greenhouse gas emissions [3]. Consequently, tuning down this significant source of emissions is crucial. Carpooling has been a widely accepted concept to implement Intelligent Transport Systems (ITS) in Smart Cities and to reduce the gas emissions caused by the use of the personal car. Carpooling (also known as ride-sharing or covoiturage), consist of the sharing of rides so that more than one person travels in a private car. Due to the fact that more than one person travel with the same private vehicle, carpooling reduces each single person's travel costs such as fuel costs,

tolls, and the stress of driving. [4] Carpooling is often encouraged by public administrations, especially during periods of high air pollution.

Thanks to the recent advancements in the Information and Communication Technology sector, carpooling is having a bootstrap in the last years. This have been made possible thanks to the diffusion of web platforms that allows passengers and drivers to meet each other and to make an arrangement for a shared ride, for scheduled trips as well as for trips arranged on the fly. Such services have recently became popular in the form of smartphone and tablet applications, made available for the contemporary mobile devices. Nowadays the most widespread implementations of ridesharing services rely on a ”static” Carpooling approach: the carpoolers post the request and the offers several hours in advance for a future transportation need. Although the most popular carpooling services have implemented their service in the form of a smarphone application, the more wide-used solutions on the market still don't take full advantage of the potential provided by mobile devices. More in depth, the advantages of the mobile devices can be used to allow the users to schedule one-time arranged trips on the fly, whenever they want, whenever they are. This particular form of carpooling is relatively new and it is called "Real Time Ridesharing". Real-time ridesharing (also known as dynamic ridesharing) is a service where an automated process employed by a ride-share provider matches up drivers and riders on a very short notice [5], which can range from a few minutes to a few hours before departure time. Dynamic Ridesharing highly relies on recent technologies, such as smartphones, Global Positioning System (GPS) devices, third-party payment gateways and social networks. Early real-time ridesharing projects started several years ago, but they faced obstacles such as the lack of convenient, efficient, high diffused means of communication.

Dynamic ridesharing clearly brings several advantages over the static ridesharing approach [6], but it requires an higher *critical mass* of users in order for the system to work; otherwise it would be difficult to reach adequate success rate in finding a travel mate, with a consequent desertion of the service by the users.

The research activity developed during these three years focused on ICT technologies for urban mobility, with particular attention to the study, design and the development of applications for dynamic carpooling services in smart cities.

1.1 Objectives

GreeenShare SRL, an academic spin-off of the University of Cagliari, is developing an efficient dynamic ridesharing application. GreenShare is intended to offer services for the sustainable mobility in the sector of mobile applications. Its leading project is named CLAC-SOON, which is a dynamic ridesharing platform that aims to overcome the main limitations of the current dynamic ridesharing services. This project has been conceived in the 2012

by a group of researches of the Multimedia & Communications Laboratory (MCLab) of the University of Cagliari, which mainly operates in the field of the Internet of Things.

The work in this Thesis has been made within the PhD programme in apprenticeship, during the development of the CLACSOON project at GreenShare SRL. The PhD program in apprenticeship allowed the author of this thesis to be hired by GreenShare SRL with an apprenticeship contract, and to be jointly enrolled in the PhD programme, in order to develop the CLACSOON project while supervised by both the University of Cagliari and GreenShare.

The thesis mainly focuses on the challenge of implementing an efficient real-time carpooling service. With this aim, the following works have been done:

- the design and the development of the CLACSOON platform, with particular emphasis on the design of the system architecture and the development of both the mobile application and the backend of the service, with the aim of automatizing the arrangement of the shared rides
- the design and the development of an efficient route matching algorithm for a dynamic ridesharing provider, with the aim to overcome some limitations of the current ridesharing applications, to improve the quality of the route matches
- the analysis of the performances of the proposed service, as well as the Quality of Experience (QoE) provided to the users, by testing the platform in a simulated smart urban scenario, against the characteristics of the population
- the implementation of a distributed algorithm for the task allocation and assignment, with the aim of maximizing the lifetime of groups of IoT devices. This algorithm can run on real devices, to perform sensing tasks while ensuring the fulfillment of the requested Quality of Information (QoI) requirements

1.2 Approach and Proposed Innovation

During the work made within the scope of this Thesis, the very first preliminary step consisted of the collecting and the analyzing of the system requirements. The development of the CLACSOON platform then started with the realization of the main requirements, expressed and documented through mockups and flow diagrams to describe application flow. After a preliminary design and planning phase, the development phase started. During this phase, a project and a development iterative approach has been followed. As before mentioned, the development consisted of the realization of both the client-side and the backend-side of the system. After a first development cycle, the Android CLACSOON platform was released in the first of October 2014. The publication of a first beta release allowed CLACSOON to achieve a some thousand of users, from which a precious feedback have been obtained.

This feedback allowed us to undertake an User-centered design (UCD) life-cycle, with the aim to improve the platform to make it more pleasant from the user point of view.

Nowadays the service has been designed and implemented and it is publicly available. Since the current population of CLACSOON users was limited, we were interested in analyzing the performance of the service with different population characteristics, which cannot be controlled in real scenarios. Therefore, an emulation system has been developed to evaluate the Quality of Experience (QoE) provided to the users. The performance has been analyzed and expressed in terms of the following Key Performance Indicators (KPI): the passenger success rate, the driver success rate, the passenger waiting time and the total system-wide CO2 saved.

The system with the major design choices have been presented in [7]. In this work, the evaluation of some Key Performance Indicators (KPI) for the carpooling service has been studied in terms of passenger success rate and the driver success rate, varying also the time distribution of the service requests.

In the work [8] the study is more in-depth and in addition to the aforementioned KPIs, also the passenger waiting time and the total system-wide CO2 saved have been evaluated against the time distribution of trips (within a fixed time window), and the timeout (T) which is the maximum time that a user can wait before he decides to retire his/her offer of a ride or his/her request for a ride. The result allowed extracting important information about the challenges to be addressed for successful deployments of a real-time carpooling service.

With the aim of improving the quality of the matches, a successive work focused on the improvement of the CLACSOON's matching algorithm, introducing two novel features: the partial ridesharing mode and a method for estimating the varying position of the driver's vehicle in an urban context. Since the partial ridesharing mode avoids the driver to take a detour when possible, the aim of this feature is to lead an increment in the total system-wide CO_2 savings. On the other hand, by modeling the position of the driver's vehicle on the basis of his/her destination, only the remaining part of the route that a driver has to travel is considered when evaluating the matching: this feature enables the possibility for shared rides to be agreed on the fly after the starting of the driver's trip, when a rider happens to be close to the remaining part of a driver's route. This work is described in the [9].

In addiction, thanks to the coming of the IoT paradigm the *carpoolers'* vehicles, which may or may not be involved in a shared ride, can represent a good sensor platform. Since one of the aspects of smart cities is the optimal use of the resources, the information coming from these sensors can be processed and analyzed to improve the efficiency and quality of the urban services and to support smart transportation and sustainable mobility. In this context, Machine to Machine (M2M) and Device to Device (D2D) communications play an important role. Starting from this vision, the author contributed to the development of a distributed algorithm for the task allocation and assignment, which can be executed by a group of real IoT devices. The result of this work, in the field of infomobility and support of

the urban mobility, could be run in real devices on the vehicles, to perform tasks such as the sensing of the speeds and temperature in a given geographical area. These devices can perform the same task required by the applications, allowing for a flexible and dynamic binding of the requested services with the physical IoT devices. It is based on a consensus approach, which maximizes the lifetime of groups of nodes involved and ensures the fulfillment of the requested Quality of Information (QoI) requirements. Experiments have been conducted with real devices, and this work is described in [10].

1.2.1 Contribution in joint works

In the scope of this thesis, the author provides the following contributions:

- the design and the development of the CLACSOON platform ([7]), for what regards the architectural design and the related major choices; the design and implementation of the Android mobile application, the backend of the service and the REST APIs
- the design and the development of the route matching algorithm ([9]) and its novel features
- the design and implementation of the emulation system, the data collection and the analysis of the performances and QoE of the service ([7], [8], [9])
- the development and testing, on Arduino devices, of the distributed algorithm for the task allocation and assignment ([10])

1.3 Thesis Structure

In the Chapter 2 the Carpooling and the Dynamic Ridesharing will be further explained, and the most widespread carpooling solutions will be depicted. In the Chapter 3 the CLACSOON platform with the major design choices is presented. The chapter 4 focuses on the improvement of the CLACSOON's matching algorithm, introducing two novel features: partial ridesharing, according to which the riders can walk to reach the driver along his/her route when driving to the destination; possibility to share the ride when the driver has already started the ride by modeling the mobility to reach the driver destination. The chapter 5 presents the study on the performances and the Quality of Experience (QoE) provided to the users, in which the CLACSOON's matching algorithm is tested and simulated in a smart urban scenario . Then in the chapter 6 the development and testing, on Arduino devices, of the distributed algorithm for the task allocation and assignment is described.

Conclusions and future works are drawn in the chapter 7.

Chapter 2

State of Art in Carpooling

Vehicular traffic congestion is one of the main problems of most of our cities and towns [11]: it degrades the quality of life, leading to a wide set of social, economic and environmental impacts. It calls for a great effort in studying and deploying innovative and ambitious urban transport modes to reach a less car-dependent life-style, which is one of the main causes of urban traffic congestion. Nowadays a fully sustainable transport model has not been conceived yet, and cities all over the world are facing the challenge of reducing greenhouse gas emissions and air pollution that results from the vehicular traffic congestion. The integration of different transport modes (known ad Multimodal transport, or combined transport) is even more required in urban areas. The particular vehicles used for the transport, the source of energy and the infrastructure used to implement the transport play a critical role for the evaluation of the social, environmental and climate impact [12]. Public transport infrastructure is for sure one of the best solutions to face the challenge of the vehicular traffic congestion, but is shows some drawbacks:

- generally a technical ("hard") intervention on a transportation structure in a city is exceptionally expensive
- public transport systems usually serve a fixed geographic route and a fixed schedule, i.e. there is no flexibility in a public transport mode

Therefore, many people prefer the use of the personal car (or other personal transportation means) over public transportation.

In the context of sustainable transport, carpooling solutions have gained more and more popularity in the last years. Carpooling (also known as ride-sharing), is the sharing of car journeys so that more than one person travels in a private car[4], therefore it combines the flexibility of private cars with the cheapness of public transport systems.

The sharing of a car journey can be mainly done trough two main different modes:

- A group of people that use to travel in the same path can share the seats in their own personal cars, interchanging the driver
- A driver can share his or her car's spare seats, and the riders can contribute to the cost of the fuel, that is they refund the driver's expanses

The first practice is common between user that use to travel within the same path (common origin and destination) and the same schedule, and then they spontaneously arrange the details of the shared ride. The second one is a mode that is also suitable for occasional commuters.

Carpooling clearly brings a wide range of positive impacts:

- for a driver, it means reducing travel costs
- for a rider, it represents a flexible - and generally cheaper - alternative to public transportation
- for the environment, it means reducing air pollution
- for cities, it means reducing the traffic congestion and the needing for parking spaces

The reallocation of the trip-related shared costs between drivers (which share their personal car) and passengers, should be made in a way that results in a cost reduction for both, but in a way that implies that the driver can not earn money, but he can only receive a reimbursement for the expenses. Carpooling is having a bootstrap in the last years, and this have been made possible thanks to the diffusion of web platforms that allows passengers and drivers to meet each other and to make an arrangement for a shared ride, for scheduled trips as well as for trips arranged on the fly. Such services have recently became popular in the form of smartphone and tablet applications, for the contemporary mobile devices. These devices have the peculiarity of allowing users to access the Internet wherever they happen to be; furthermore these devices generally rely on the availability of a set of sensors which provide important context-aware information.

A carpooling smartphone-based application is coordinated by a network mechanism which is able to connect together riders and driver that are willing to travel along similar paths.

Thanks to the information provided by such a network, ride offers or requests are able to reach a wider amount of users, that is the probability to find a travel companion is higher. The information provided by this type of applications to its users allow to decline the usual diffidence of the users to share their car with strangers. This situation used to represent one of the major stumbling blocks for the carpooling diffusion across the world.

However in reality, most of current carpooling systems or applications are not functioned well as the expected.

2.1 The major elements of carpooling

The matching between ride offers and requests has both temporal and positional factors. In the literature, Furuhata et al. [13] well classified the positional and temporal elements of carpooling. This classification can be summarized as follows:

1 an *identical ridesharing* happens when the departure and the destination pickup points of the rider coincide with the departure and the destination of the driver

2 an *inclusive ridesharing* happens when the the departure and the destination pickup points are on the original driver's route, but the shared ride covers only a part of the driver's original route

3 a *partial ridesharing* happens when the pickup and drop-off points are on the driver's original route, but they don't coincide with the departure or the destination of the rider

4 a *detour ridesharing* happens when the driver has to take a deviation in order to reach the rider's pickup points

As stated in [13], the partial ridesharing and detour ridesharing modes are currently not facilitated by ridesharing agencies. The partial ridesharing is particularly important for this thesis, and it will be taken into account in the implementation of the matching algorithm, which will be covered in depth in chapter 4.

The main temporal element of carpooling is the desired departure time of drivers and riders. This parameter plays a vital role in the determination of the matching between users, because the matching between offers and requests is not only related to the geographical distance between the users' routes, but it is also related with the scheduling of pick-up and drop-off times. Another important temporal parameter is the *time flexibility* [14] associated with departure times, which represents the amount of time that users are willing to wait (or to anticipate) the departure time.

For the identical ridesharing pattern is easy to find a schedule for the pick-up and drop-off times; but for the others ridesharing patterns the determination of the meeting times is more difficult because it requires the estimation of the arrival time to the pickup points. Therefore, another important temporal element to be taken into account while calculating the matching between riders and drivers on a temporal basis is the estimation of the travel duration.

Current carpooling agencies and operators have focused on two types of shared rides [13]:

- *long distance trips*, typically occasional trips. These trips are the target of the Static Carpooling, which will be covered in section 2.2

- *commute trips*, i.e. recurring trips between two locations, typically house and workplace

Another new segment is *ad-hoc* trips, whose details are typically arranged in an automated way, on the fly. This particular type of trips is the target of Dynamic Ridesharing, which will be covered in section 2.3

2.2 Static solutions

Carpooling applications allow users to register new accounts, specify the desired departure and destination locations, the desired departure time and, optionally, the desired time flexibility. Carpooling applications usually rely on third-party social networks (eg. *Facebook*) to improve the trustworthiness between users, and also to enable a sense of social participation. Furthermore, carpooling applications usually implement rating mechanisms that allows users to release a feedback about each other for a particular trip: this mechanism has the purpose of improving the trustworthiness between carpoolers. The sharing of the trip expenses can be made via cash or via automated payment mechanism. These mechanism allows riders to reimburse drivers with an amount of money related to the length of the shared ride.

2.2.1 Static carpooling Applications

Nowadays the most widespread implementations of ridesharing services rely on a ”static” approach: the users can post the requests and the offers several hours in advance for a future transportation need, and shared rides have to be arranged before the trip starts. Typically the static carpooling approach does not contemplates the possibility of unexpected changes of schedule or other details of the shared ride.

Such a system typically rely on matching lists, generally based on common origin-destination matching; drivers and riders are expected to spontaneously contact one another with the purpose of arranging the details of a shared ride and then to make an agreement. The details to be agreed include at least the specification of the pick-up and the drop-off points of a rider and the desired meeting time, as well as the the amount of the cost sharing, or the possibility to bring a luggage.

Static Carpooling performs well for long-distance scheduled trips, but it does not perform well in an urban context, when users typically have an improvised need. At the moment of writing, a famous example of a static carpooling application in europe is BlaBlacar [15].

2.2.2 A case study: BlaBlaCar

This section provides an explaination Static Carpooling process by explicating the process of BlaBlaCar [15], a software company that currently offers a static ridesharing application for Internet-enabled mobile phones and also for desktop browsers.

At the time of writing BlaBlaCar is the world's largest long-distance ridesharing community, with more than 25 million members across 22 countries. [16]

BlaBlaCar connects drivers that are willing to drive with passengers that need to find a ride. These users usually organize shared rides between two different cities and then share the cost of the journey. The users must register to the platform and then create their profile which includes user's interestes, profile accountability, related social network profiles as well as rating and reviews by other users. The service is accessible via web, mobile and also via apps for iOS and Android[16]. As others static ridesharing projects are, BlaBlacar is designed to be suitable for long-distance travel arrangements, for drivers that are looking to fill empty seats during long journeys that they have planned to take. Users can offer a ride as a driver or search for a ride as a rider. The application lets the users specify the departure and destination cities by typing its names or by selecting pre-defined locations such as home or work. With a GPS-enabled phone, users can set their current location as the departure or as the destination of the trip, and optionally they can add one or more waypoints. A driver's and a rider's journeys are matched if they have in common the same stop (origin, destination, or waypoints). The application lets user specify the desired departure times, and drivers can specify a desired amount for the reimbursement. Users can then, searching from a matching lists, choose the travel companion that best match is preferences: origin, destination and departure time If the shared ride is agreed, the driver picks up the rider at the agreed time and location. The costs are split between participants, and users can make the payment through their phones, via PayPal.

One drawbacks of a similar approach is that it does not allow a truly ad-hoc trip arrangement:

- users are matched via origin-destination pairs (or waypoints), therefore only the *identical ridesharing* mode is taken into account
- the shared rides has to be arranged in advance, preferably from some hours till a few days
- drivers cannot pick-up riders en-route during the trip: journeys are matched if they have in common the same stops

2.3 Dynamic Ridesharing

On the other hand, dynamic ridesharing is a relatively new type of carpooling which is more suitable in the urban context: it is a system where an automated process employed by a ride-share provider matches up drivers and riders on a very short notice [5], which can range from a few minutes to a few hours before the departure time. Dynamic Ridesharing is the most recent class of ridesharing. Dynamic ridesharing clearly brings several advantages over the static ridesharing approach. On the technical side, Information and communication technologies play a vital role in enabling the dynamic ridesharing.

2.3.1 Dynamic Ridesharing applications

The deployment of a Dynamic Ridesharing service typically relies on the following recent technologies [17] :

- *Mobile devices* (smartphones) for riders and drivers, which represents the user interface to the service and allow the arrangement of a shared ride in mobility
- *GPS devices* (usually integrated into the smartphones) to track the drivers' route and to determine the position of riders
- *Social networks* to improve the trustworthiness between riders and drivers
- *Constant Network Connectivity* The need of making an arrangement for a shared ride on a very short notice requires that users' smartphones have to be constantly connected to the Internet. [18]. Nowadays, many multinational telecommunications companies are facilitating constant mobile network connectivity.

The aforementioned technologies are the key enablers of the Dynamic Ridesharing. Early real-time ridesharing projects started several years ago, but they faced the lack of these means of communication.

These elements are coordinated through a network service, which can instantaneously handle the driver payments and match rides using opportune optimization algorithms. Agatz [14] listed the following most important features of the Dynamic Ridesharing:

- *Dynamic* i.e. the shared rides can be arranged on a short notice
- *Independent* drivers are not part of an organization, but they are autonomous entities
- *Cost-sharing* dynamic ridesharing leads to cost savings for both riders and drivers
- *Non-recurring trips* are the main target of the Dynamic Ridesharing. While traditional carpooling generally requires the shared rides to be committed several hours in advance, or to be arranged as recurring trips (for example between colleagues which

daily travel to work), dynamic carpooling generally is more suitable for on-demand, spontaneous, shared rides which can be arranged just a few minutes in advance.

- *Prearranged* the shared ride is arranged in advance, i.e. when drivers and riders are not on the same location
- *Automated matching* allows users to arrange the details of shared rides in an automatized way, so that it requires a minimal effort from the participants. Users are not required to manually search for suitable travel companions: a system matches up riders and drivers and then communicates the matches to the participants.

The idea of dynamic ridesharing is not very recent and several initiatives have been tried in the past in the field of business, for example by Flinc (www.flinc.org), Carma Carpooling (www.gocarma.com) and Commutr (www.getcommutr.com), Lyft (www.lyft.com).

2.3.2 A case study: Lyft

Lyft is an American Dynamic Ridesharing agency based in San Francisco, launched the 2012 by Logan Green and John Zimmer [19]. The system was launched as a service of Zimride as an on-demand ridesharing network for shorter trips within cities. Lyft now operates in a large number of cities, including San Francisco, Los Angeles, and New York City. To use the Lyft appication, users must download the Android or the iOS application from the store, and then they can sign up, enter a phone number and choose a method of payment[20].

Using Lyft, a passenger who wants to ride can search for a nearby driver from an interactive map.

Once the shared ride is arranged, the rider can consult the driver's data: name, rating, profile picture and the type of car. Riders are picked up by drivers at their current location. The costs of a shared ride are the split between participants, and the users can make the payment through their phones, via PayPal.

The tenet of Lyft's platform is establishing trust among its users [19]: drivers are rated by passengers and only the highest-rated drivers are allowed to share their spare seats. Moreover, all drivers have to follow a screening processes[20]: criminal background check, vehicle standards checks and zero-tolerance for drugs and alcohol.

One potential drawback of this platform is that riders can only be picked up by drivers at their current location, therefore the partial ridesharing mode is not taken into account by the matching algorithm. Moreover, riders can only requests rides to the nearby drivers, i.e. the driver's route seems not to be taken into account when evaluating the matching.

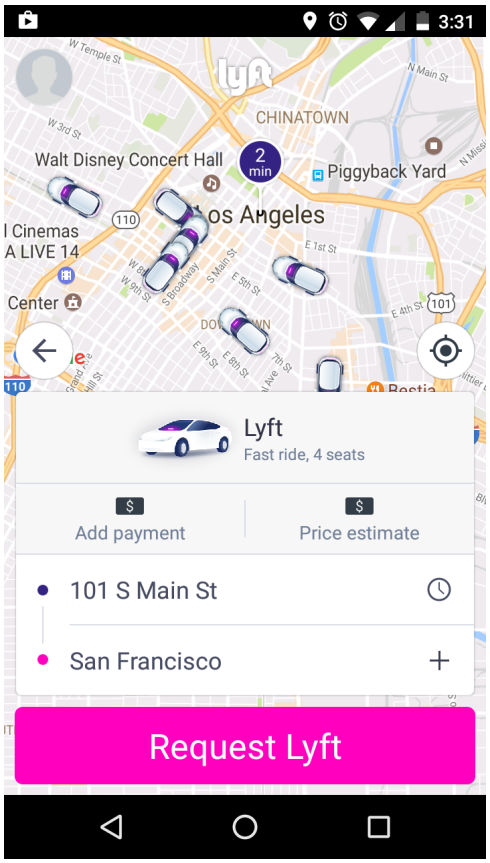

Figure 2.1: Lyft

2.3.3 Research efforts

Dynamic ridesharing clearly brings several advantages over the static ridesharing approach: because of its potential, also several research efforts have been done in the last few years. In spite of the amount of these efforts, the problem of matching ride requests and ride offers in large scale remains challenging. Several matching agencies tried different approaches, but what constitutes the best procedure is still a matter of debate [13].

Carpooling provides people a better way to make use of the spare seats on a personal car, and thus saving travel-related expenses by sharing the travel cost. Private Ridesharing agencies usually operates for profit, therefore, when these agencies implement an electronic payment mechanism, riders are able to reimburse the cost of the shared ride throughout this system. Therefore these agencies can take advantages by charging a fixed percentage of the reimbursement of the driver, and thus these agencies has the same objective of drivers: to save the largest amount of costs. Both private and public ridesharing agencies has also an

environmental objective, such as the reduction of pollution and congestion. Reducing travel costs is also the objective of riders.

Supposing that the most of ridesharing users would drive their own car if a matching have not been found by the ridesharing provider [21], it can been stated that the objectives of the objective of the service and the objective of the users are related. Given these objectives, the ridesharing matching problem in literature has been often modeled as an optimization problem [22], [13].

Commonly used objectives in literature are:

- *to minimize the total travel distance* traveled by users. When calculating the total distance, the travel distance shared by two or more users in the same car at the same time is considered only once
- *to maximize the total cost savings*, which is equivalent to the objective of minimizing the total distance traveled by users, or maximizing the total CO_2 saved.
- *to maximize the number of served passengers*, i.e. to maximize the success rate.
- *to minimize the users' waiting time*, i.e. the time that the users have to wait before finding a travel companion

Many studies in literature (but also many implementations of ridesharing providers) typically consider travel costs as proportional to shared route, which cost has to be shared between riders and drivers. Note that, while considering the cost savings, also the cost of the detour that a driver has to take in order to reach up the pickup points has should be taken into account. With this consideration, a shared ride leads to CO_2 savings only if the length of the driver's detour is less or equal to the length of the shared ride.

The main technical challenge is the complexity of the optimization problem and the matching process itself, along with the complexity of accurately modeling of the carpoolers behavior. On the practical side, one of the main challenges that the dynamic ridesharing has to face regards the *critical mass* issue [23], which is generally faced by the carpooling, and particularly by dynamic ridesharing in its startup phase, when typically it is very difficult to achieve a critical mass of users in order for the service to achieve an high value for the matching success rate. This challenge is also related to the quality of experience perceived by the users, which depends on factors such as safety, social discomfort, and time flexibility. In [24] a ride-sharing concept for short distance travel within metropolitan areas is designed as a multiagent system to handle spontaneous ridesharing requests of prospective passengers with transport opportunities available on short call. [25] illustrate WiSafeCa (Wireless Traffic Safety Network Between Cars), a Eureka/Celtic founded European project that consists in researching and prototyping efficient car-to-car and car-to-infrastructure networking mechanisms striving to reduce accidents and traffic congestion. In the scope of the project,

a dynamic ridesharing system was designed, in order to serve real time transport requests. In [26] is considered the problem of matching drivers and riders for a dynamic ridesharing scenario, presenting a simulation study based on travel demand data for the city of Atlanta. The matching problem is described as the minimization of the total system-wide vehicle miles incurred by users, and their individual travel costs. The simulation results indicated that the use of sophisticated optimization methods based on a rolling horizon approach substantially improve the performance of ride-sharing systems over a greedy matching algorithm. In the definition of their study, an important assumption is that a driver could make only one pickup and one delivery: this constraint makes the problem easier to solve, but it prevents the driver from serving some riders even if they are on his desired route. Another important assumption for the study was that a shared ride must be agreed before the starting of the driver's trip; moreover, the dynamics of the positions and the speeds of all the shared vehicles are omitted.In [27] is addressed the dynamic ridematching problem with time windows, optimizing a multicriteria objective function. Extending the work proposed by Agatz, they propose a genetic and insertion heuristic algorithm for solving the optimization problem, also considering the multiple ride problem (i.e., more than one rider for a single driver). The problem is represented using a maximum-weight bipartite matching model and the optimization software CPLEX is used to solve it. In [28] the proposed ridesharing system considers the interactions between drivers, riders and the system manager using a model based on mixed continuous-integer linear programming to maximize the performance of dynamic ridesharing systems. The dynamics of the positions and the speeds of all the shared vehicles are omitted for simplicity, and it assumed that users can meet only at a priori fixed delivery stations, such as near bus stops, intersections, the corners of squares. The performance of the proposed model has been analyzed through a simulation based on the modeling framework for discrete event systems (DES). In [29] the authors describes a former implementation of the agent-based travel demand model mobiTopp, with the aim of realizing a realistic model for ridesharing as an agent-based travel demand model. The model has the limitation that it currently supports only end-to-end ridesharing, i.e. only matching between origin-destination zones.

Sharing a ride can also lead to some side effects[30]: for drivers, making a detour to reach up the riders' pick-up and drop-off points could represent to a waste of time and money when these points are not close to the driver's route, since that behavior increases the total miles traveled by the driver. This drawback is generally minor compared to the total CO_2 savings related to the sharing of the ride, but it points out some fields of improvement for ridesharing systems. For example this side effect can be mitigated by a carpooling system that evaluates only the pick up point on the driver's route.

Recently the report *Commuting in America 2013* [31] shown that ridesharing has continuosly declined in the last years, reaching the 12.2% in 2000 and the 9.7% in 2010, while in 1980 the percentage was 19.7%[32]. This decreasing indicates that the recent improvements

brought by ridesharing matching agencies has not been successful in changing the users' inclination over the transport mode[13].

2.4 Open issues

The dynamic ridesharing clearly brings several advantages over the static ridesharing approach, but it has the drawback of requiring a critical mass of users in order for the system to work. Otherwise it would be difficult to reach adequate success rate in finding a travel mate, with a consequent desertion of the service by the users.

As it resulted from the previous review, many works in literature have focused on the optimization problem. Despite of the amount of the research efforts, current dynamic ridesharing agencies do not seem to adopt a trip planning policy that *globally* maximizes the number of shared rides [13]. This choice is likely due to the fact that most of the matching optimization studies in literature propose an optimization algorithm whose result have to be accepted by all the users in order for the system to work. In other words, it is presumed that each rider would accept to travel with a specific driver specified by the matching algorithm. This situation is not plausible in reality, and therefore it can not be applied to route matching algorithms adopted by real matching agencies. A lot of efforts have been made to study the optimization problem, but only few have worked on the modeling of the driver mobility to find better matches. Additionally, several works in literature proposed a simplified model for ridesharing to make the optimization problem easier to be addressed. One common assumption is that a shared ride must be agreed before the starting of the driver's trip. Nonetheless, the partial ridesharing mode is not currently facilitated by matching agencies [13] and, at the best of my knowledge, its benefits have not been investigated in literature yet.
Based on these considerations, the novel carpooling solution for dynamic ridesharing servicess proposed in this thesis and named CLACSOON has been implemented and deployed for real usage. The CLACSOON's route matching algorithm includes the partial ridesharing mode. In this way, the driver is avoided to take a detour whenever possible, therefore it leads to an increment in the total system-wide CO_2 savings. Clearly, it calls for the riders to walk to reach the driver along his/her route when driving to the destination. Additionally, by introducing the modeling of the position of the driver's vehicle, only the remaining part of the route that a driver has to travel is considered when evaluating the matching. Therefore this approach enables the possibility for shared rides to be agreed on the fly after the starting of the driver's trip, when a rider happens to be close to the remaining part of a driver's route. This approach leads to an increment in the amount of the number of total shared rides. To evaluate the impact on the performance of the system changing the population characteristics, an emulation system has been designed and deployed to generate increasing numbers of users that interact with the CLACSOON platform, and extensive trials have been imple-

mented to analyze some performance indicators varying the characteristics of the population in the city of Cagliari (Italy). In particular, the passenger success rate, the driver success rate and the total system-wide CO_2 saved have been evaluated with respect to the characteristics of the population. The results shows that introducing the aforementioned features in a route matching algorithm leads to a substantial performance improvement.

The proposed work is described in the next chapters.

Chapter 3

The CLACSOON Solution

The following sections provide a brief overview of CLACSOON, focusing on the architectural design, the development phase and the results of both the client side (Android and iOS mobile applications) and the backend side (with the Java EE platform) of the system. CLACSOON is a dynamic ridesharing system that automatizes the interaction process, avoiding the time-consuming anticipated trip planning which derives from the needing of manually establishing the pickup points and the details of a shared ride. Moreover, the service introduces the following two novel features:

- partial ridesharing, according to which the riders can walk to reach the driver along his/her route when driving to the destination
- possibility to arrange a shared ride when the driver has already started the ride by modeling the mobility to reach the driver destination. This feature enables drivers to pick up riders on the fly, if the pick-up point is close to the remaining route points.

The platform is currently working and it is available for the major mobile operating systems.

3.1 Features and requirements

The most important CLACSOON's features can be listed as the following:

1. ***registration and accounting*** to allow the user to access the service and manage his/her account. Each user has a profile in which various information is stored, such as name, age, type of car, received feedback.
2. ***offers and requests insertion*** through this feature each user can insert an offer or request a ride. Each ride is identified by a departure point, an arrival point and a search radius representing the maximum deviation from the scheduled trip.

3. ***route matching evaluation*** to match requests and offers on the basis of the geographical and temporal constraints. This feature is a key point for this work and it will be explained in depth in the chapter 4

4. ***matching notification*** to notify - in real time - the user when a matching is found. The details of this information will be explained in the next sections

5. ***travel cost estimation*** to estimate an amount for the cost sharing between users, based on the length of the shared ride

6. ***shared ride management***, to arrange the details of a shared ride, both on a short notice or several hours in advance,

7. ***a check-in and a check-out mechanism*** to automatize the reimbursement

8. ***rating mechanism (feedback)*** to establish trustworthiness between users

9. ***travel cost sharing*** through a credit mechanism

10. CO_2 ***savings estimation*** based on the length of the shared ride and the number of riders in a car

Additional nonfunctional requirements could be listed as:

- ***Scalability and Capacity*** the system should be capable of handling a growing amount of network traffic, in order to accommodate a growing amount of riders or drivers during the pick loads. These peak loads are surely related to the users habits as well as to climate, environmental and other factors.

- ***Usability*** The system must be easy to use both for riders and drivers. The application must have a good user experience for users who likely will be using the service in mobility.

A detailed description of the key features will be given in the following sections.

3.2 Architecture

This section provides a description of the whole architecture of the CLACSOON system. In typical urban real-time carpooling scenario it frequently happens that users have an improvised need and so the trip is not previously scheduled. Accordingly to this scenario, the system architecture needs to realize a system that simplifies and automatizes the provisioning of the carpooling processes. The automation of the system is a very important tile to improve the QoE (Quality of Experience) and to create an incisive user persuasion strategy.

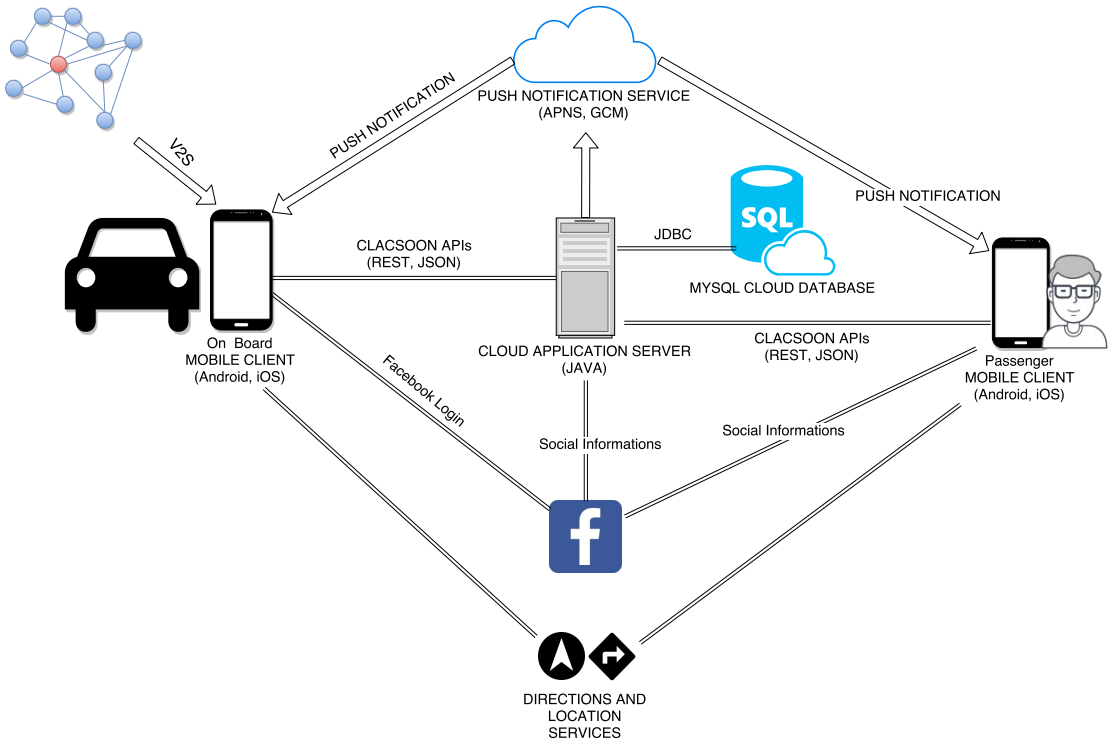

Figure 3.1: Functional blocks description

All the aforementioned features require an interaction between the CLACSOON mobile application (which has the role of the client) an the CLACSOON backend. A detailed description of the key features will be given in the following sections.

The figure 3.1 depicts the architecture of the CLACSOON system. Regarding the back-end side of the system, it is developed completely in the cloud, to offer good reliability considering the high number of expected connections and then to provide a good scalability feature [33]. Third party services (i.e. Facebook API, Direction API, etc.) are used to build the proposed service.

As already mentioned, the system follows the paradigm mobile-cloud. Fig. 3.1 shows the major components:

- The ***mobile client*** (Android and iOS) allows the user to access the carpooling service in mobility. Its sensors (e.g. GPS) are used to simplify the access of the service and to enhance the user experience [34]. For all communications toward the server, the JSON format is used.

- The ***cloud application server*** is the core of the system. It enables the access of users,

processes all requests and offers of rides and calculates the matching between requests and offers. In the implementation of CLACSOON the technology chosen is GoogleAppEngine and its tools for cloud solution.

- The ***database*** has the task of storing all the data useful for the service: user profile, ride offers, ride requests, etc.
- The ***Facebook APIs*** are used to simplify the process of registration by offering a quick and easy service to authenticate the user. Using the facebook social graph the aim is to increase the social participation of users.
- The ***direction and location services*** are used to evaluate the route between the two points (departure and arrival points) chosen by the user for its ride. These services are used also for the geocoding of address, that is the conversion of text in coordinates.
- The ***push notification services*** are used to enable the push notification toward smartphones. This feature is a milestone to obtain the real-time requirement [35].
- The ***sensors on board*** the information coming from these sensors can be processed and analyzed to improve user experience of the service, as well as to improve the efficiency and quality of the urban services in the context of a smart city

To further understand the operating principles of the system it is helpful to give a definition of the communications interfaces between the system's elements, referring to the figure 3.1

1. The interaction between the server and the mobile application is made via RESTful APIs, thus the services are represented by URIs
2. the server is connected to the database trough a JDBC connector
3. the mobile application and the server can interact with the *Facebook* social network trough REST APIs. This is done with the use of the Facebook SDK
4. The interaction between the mobile application and the Directions and Location services is made of RESTful APIs
5. The interaction between the Android application and the GCM service is made via the Google Play Services application within the Android device
6. The interaction between the iOS application and the APNS service is made via the Apple Notification Center Service (ANCS) within the iOS device
7. the server can send push notifications to Android and iOS devices trough GCM and APNS via RESTful APIs

The system has to be used by users in mobility, so the access of the system has to be guaranteed by mobile devices. Accordingly, the design of the system architecture considers this facility and the front-end layer is projected for a mobile device.

The driver offering a ride can use his/her mobile device connecting it with the car equipped with on board sensors. The connection between user's device and vehicle creates a vehicular sensor network (VSN) [36]. A VSN is a kind of network in which sensors equipped in the vehicle, providing widespread connectivity among mobile users [37]. Sensors in a vehicle can be of different types, as discussed in [38], which provides a classification based on the scope of each sensor. In most cases they are installed in the vehicles to support comfort in their usage and support the deployment of easy to use applications for drivers. The coming of IoT paradigm has transformed the car in a formidable sensor platform that absorbes information from the environment and feeds it to the drivers to assist him/her [39].

On the client side, the mobile application has been deployed both for the Google's Android and the Apple's iOS platiforms. On the other side, the server is responsible to handle all the operations necessary to enable the service: it's in charge of serving the requests from the client, it deals with the offers and requests to evaluate the matching, it manages and storage the user's data.

Therefore, for what regards the interaction between the client and the server, the HTTP protocol has been chosen: the client can send to the server HTTP requests as GET and POST requests. After the receiving of a request form the client, the server executes the request and returns a response to the client. These response have to be represented according to an encoding scheme that has to be conveniently chosen.

Also the JavaScript Object Notation (JSON) format has been chosen, which is a text-based object representation format, as the data-interchange format for all the communications. This format is a lightweight data-interchange format, which is language independent, "self-describing".

3.2.1 Functional levels

Fig. 3.2 shows the overall architecture of the platform through three main functional levels:

- the lower level is made up of the sensors which generate the flow of data used by the system
- the mid-level is the core level and it is totally deployed in the cloud, it interact directly with the data acquisition level and presentation layer
- the higher level is the application level in which user-oriented services are deployed.

The data acquisition level holds all sensors involved in the system. Each sensor can be directly connected on the internet to communicate acquired data or it can use a gateway (i.e. smartphone's user) to communicate its sensing data.

The core of the system is located in the cloud. This level is composed by various entities which have different assignments. The Data Handler may intervene when there is the need to process data from a source that is in another layer. For example, data coming from sensors could be strings, number or boolean: they have to be processed to extract interesting values that can be used by other entities inside the same layer. To maintain an open and interoperable architecture, also an entity that manages external services interfaces is inserted. By this interface the system can communicate with external services, for example Google cloud messaging to enable a push notification services or Facebook social graph to enable some social accounting services. The most important entity in this layer is the matching engine. This entity processes the data received by the data handlers and calculates if there is a matching between an offer and a request of a ride. The last entity in this layer is the accounting, which manages the access and the registration of the users.

At the top of the architecture, the last layer is the application level. This layer interfaces directly with the user to present the data generated by the system and to acquire data inserted by the user. In the case studied this level is implemented by a mobile app for smartphone.

Following the presented architecture, the resulting system is designed to scale, to interoperate with other external services and to be modular.

3.3 Adopted Technologies

This section describes the technologies adopted for the development of the system's mobile application and the backend. The backend side of the CLACSOON system has been deployed completely in the cloud to offer good reliability considering the high number of expected connections and then to provide a good scalability feature [33].

Also other third-party services (i.e. Facebook API, Direction API, etc.) are used to build the proposed service:

- ***Google Cloud Messaging*** to achieve the real time feature. A key requirement for the service is the capability of receiving real time updates: this is required for enabling the real-time matching notification. For example, when a rider requests a ride to a driver, the request message must be sent and delivered in a short time. In this context, it can be assumed that this requirement is satisfied when, after the request has been sent, a related message is received within a few seconds.

- ***Facebook API*** Allows the users to speed-up the sign-up process as well as to enable a sense of social participation

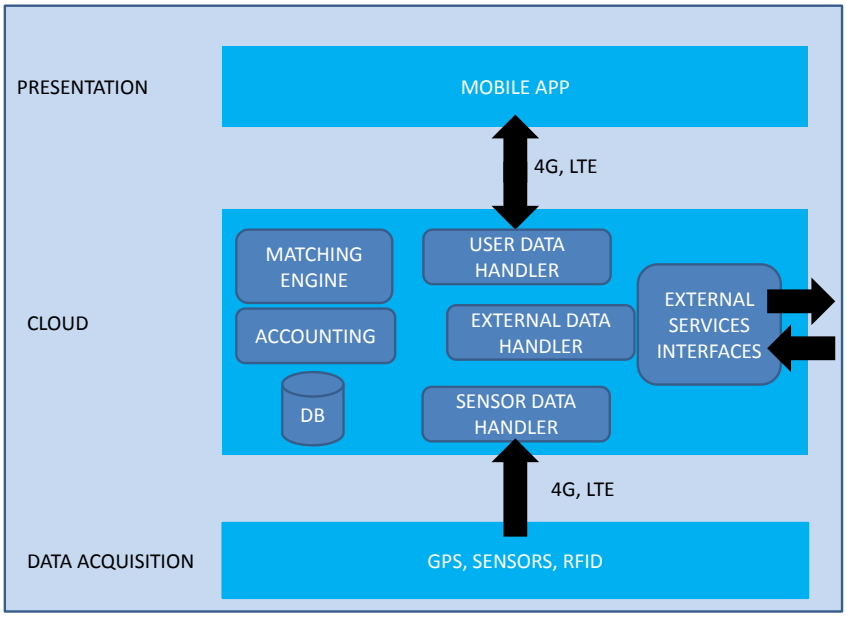

Figure 3.2: Functional levels of system architecture

- The ***Google Directions API*** is chosen to be the *Directions and locations provider*. This service is used to calculate the driver's path, on the basis of the rider's desired departure and destination points and a driver's desired travel path. In order to perform the calculation of the route matching, the carpooling platform relies on this information.

Furthermore, the following frameworks have been used during the development of the backend:

- ***Java Platform, Enterprise Edition (Java EE)*** is a widely used computing platform for enterprise software written in Java, which provides tools for deploying and running large-scale, multi-tiered, scalable, reliable, and secure network applications. The design of the CLACSOON's backend components has been made following the Model View Controller pattern (MVC).

- ***Spring framework*** At the time of writing Spring is one of the most popular application development frameworks for enterprise application development in Java. By using the Spring Framework, millions of developers around the world develop high performing, testable, and reusable Java code.

- ***Hibernate ORM (H8)*** is an object-relational mapping framework for the Java language. By mapping the application's business objects to the corresponding database's table, it therefore helps developers to work with databases
- ***Google App Engine for Java SDK*** is an essential tool for developing, deploying and managing Java applications within the Google App Engine infrastructure. Therefore Google App Engine is suitable for developers who wants to run large web applications without the needs of constantly take care of the traffic level and providing a way of scaling. It also allows companies to avoid upfront infrastructure costs (e.g., purchasing servers). As well, it enables organizations to focus on their core businesses instead of spending time and money on computer infrastructure. Thanks to the cloud computing, the CLACSOON backend can scale up as computing needs increase and then scale down again as demands decrease.
- ***Google Cloud SQL*** Google Cloud SQL is a database service within the Google's infrastructure that makes it easy to manage MySQL relational databases on the Google Cloud Platform.

3.4 The mobile application

This section depicts the main features of the current implementation of the CLACSOON Android mobile application. An analogous implementation has been made for the Apple iOS platform. A brief description of the application's workflow can be given trough the UML language, with the activity diagram depicted in figure 3.3. Each point will be described in detail in the following subsections.

3.4.1 Registration and Login

The first screen of the application is the Login form (Fig. 3.4.a). Through this screen the user can choose between the following options:

- ***register a new accont*** - the user is redirected to the screen in Fig. 3.5.b, to complete the registration by manually entering the information of his/her account. After the registration, it is possible to access to the application's home by making a sign-up with the user's credential or with the sign-in with a ***Facebook*** account
- ***sign-in with a CLACSOON account*** - This feature allows membership Signup and Login via the user's email and password
- ***sign-in with a Facebook account*** (Social sign-in) - This feature allows membership Signup and Login via Social Network Credentials.

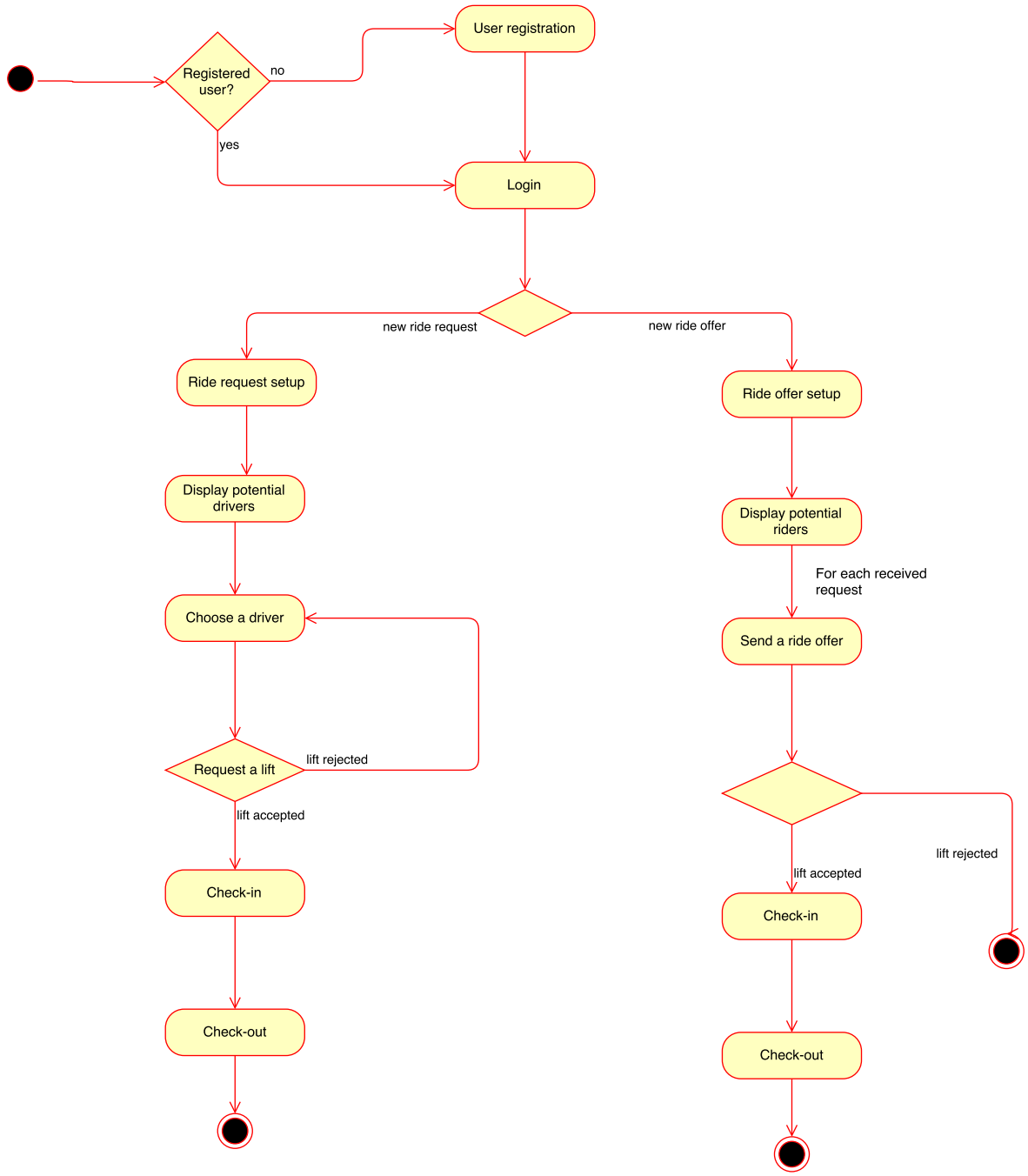

Figure 3.3: CLACSOON's activity diagram

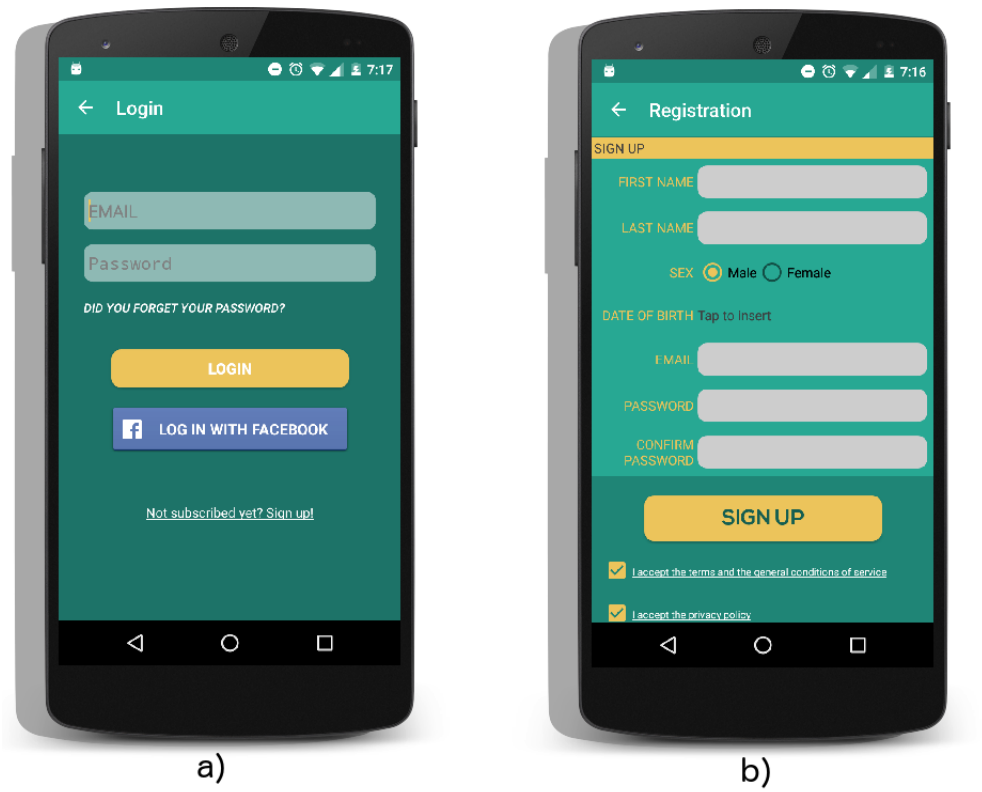

Figure 3.4: CLACSOON Android: Registration and Login

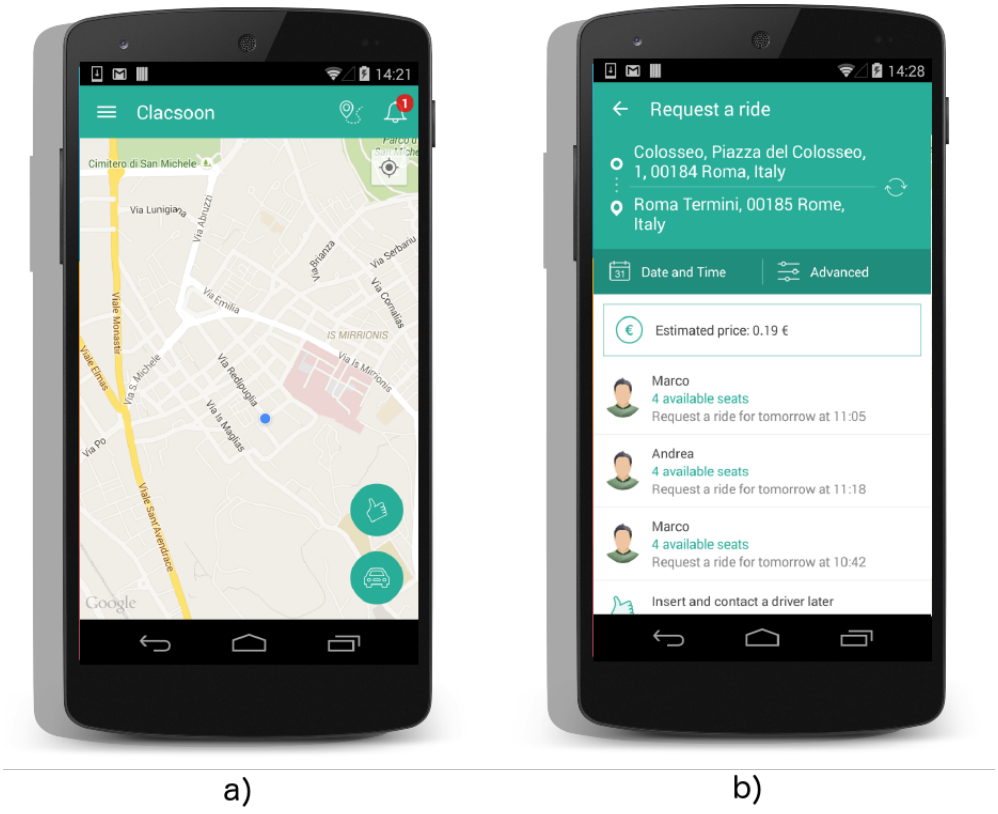

Figure 3.5: CLACSOON Android: Request and Offer insertion

3.4.2 Ride offers an ride requests pubblication

Once logged in, the user is redirected to the Home Screen (Fig. 3.5.a).

The home screen displays the recent users in the device's surroundings on an interactive map. The user can choose to offer or request a ride by clicking respectively on the ”thumb" icon or in the ”car" icon. This feature is implemented on the screen in (Fig. 3.5.a). The ride offers and requests publication feature is a core feature of the project. The users, by specifying their trip details, are able to indicate:

- the desired departure time: it could range from some minutes before the departure, to several days in advance. Users are also allowed to create recurring trips
- desired departure and arrival locations: they can be entered both textually (using the reverse geocoding feature provided by Google Maps) or by specifying the coordinates with the help of a dedicated map within the mobile application. In the case a user want to start moving form his current location, he can set the starting position with the help of the GPS or other geolocation providers.
- in the case the user is a driver, along with the departure and destination locations, he/she has to specify a route he wants to take by choosing it from a list of alternatives, which are supplied by the Directions and Locations Provider

In addiction, users can set the following advanced options:

- for riders, the maximum distance they could walk to reach the suggested pickup point
- for drivers, the maximum detour they are willing to take for reaching up a suggested pickup point
- for both drivers and riders, the maximum time flexibility

The role of these parameters will be made clear in the chapter 4. Once a new travel is submitted to the system, the route matching algorithm (chapter. 4) evaluates the matching between rides, according to the aforementioned options.

3.4.3 Matching Notification

After submitting the details of a ride offer or a ride request, the user is redirected to the screen 3.6.a).

The user can then search for suitable trip announcements which are compatible with his or her desired route. When a user searches for potential trip companions for a specified trip, the mobile application generates a matching request to the backend. The backend then executes the route matching algorithm which output results in a matching list containing all

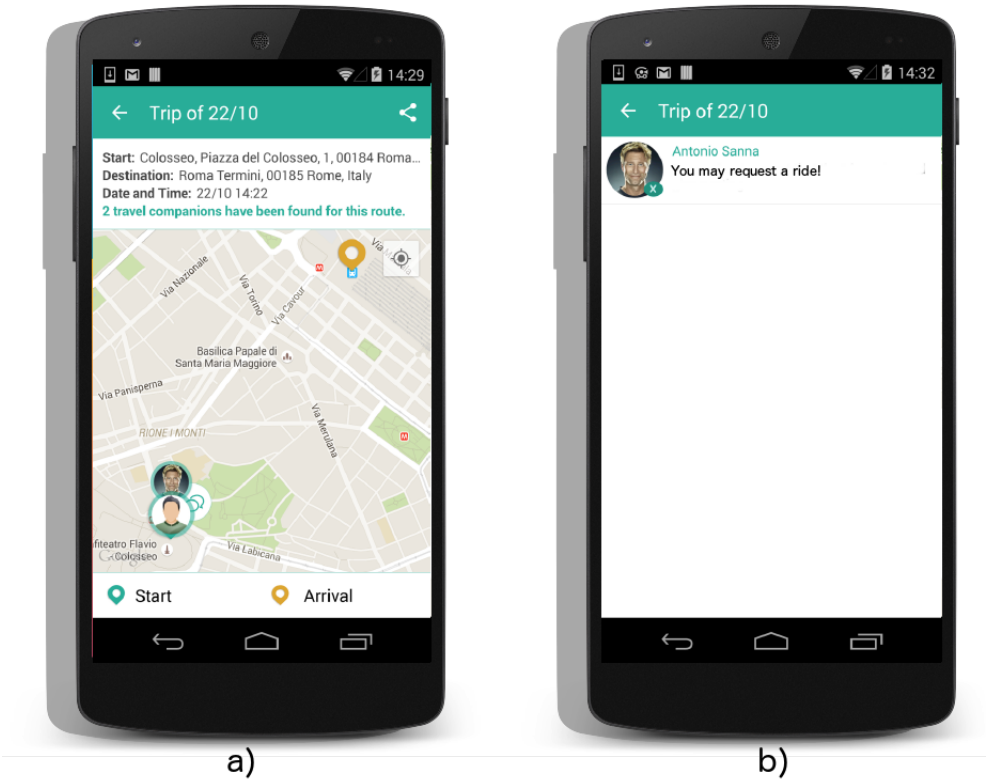

Figure 3.6: CLACSOON Android: Matching Notification

the trips (and their related users). All the items of this list have to satisfy the temporal and positional constraints that will be described in detail in the chapter 4. The information of the matching includes: the pick-up point (where the ride can start), the drop-off point (where the ride can finish) and the expected driver arrival time.

A list of the potential travel companions is ordered by the value of a cost function for the shared ride. Each user can accept or refuse the notification.

The information regarding the matching can be consulted on a map 3.6.a) or either in textual form with a list 3.6.b). The application updates the information about the matching after the following events:

1. scheduled update at a fixed interval of time
2. automatic update when a particular trigger happens
3. update after an explicit request from the user

The point 1) and 3) can be adopted in a situation in which the application is in foreground, so the user can force an update or the application can refresh the information at a fixed interval of time. The second point refers to a situation in which the application have been closed or it is running in background: that is useful when a certain happening has to be notified to the user even if he is not using the application. For example, that's the case when

an user *A* sends a ride request to an user *B* while the user B is not using the application: thus the request has to be notified to the user *B* with a *push notification*, and the matching information have to be updated.

3.4.4 Shared ride agreement

After the receiving of the information contained in the matching list, the user can consult the details of the potential travel companions (Fig. 3.7.a), and then he can choose a suitable travel companion which best matches his preferences (Fig. 3.7.b)).

When a user identifies a suitable travel companion, he can send him a ride request or a ride offer. The real time feature of CLACSOON requires that ride offers or ride request must be received within a short time; this feature is enabled by the *push notification*. On the other side, the receiver mobile application alerts the user that he just received a new offer or request for a ride. The information provided in this step also includes:

- the suggested pick-up and drop-off points, which have been calculated by the matching algorithm in the previous step
- the estimated meeting time, which is an estimation based on the travel speed estimation, provided by the Directions Provider
- public user-related information such as name, age, interests, mutual friends within social networks

The application also provides a messaging system, which allows the users to further interact and to refine the details of the shared ride before confirming the agreement. The receiver user, after he read this information, can accept or decline the offer or the request (Fig. 3.7.b).

3.4.5 Check-in and Check out

After a shared ride is agreed, users have to meet each other in the suggested pickup point. To notify the system that the shared ride has actually began, the users have to register the meeting with a check-in procedure. This operation has two main purposes:

- increase the safety of the users, because the meeting is certified
- automatize the reimbursement procedures between users. For sake of brevity, this feature is not further explained in this thesis

To certify the meeting, every user receives a secret code: before the beginning of the shared ride, the users have to mutually-exchange their codes, which have to by typed within

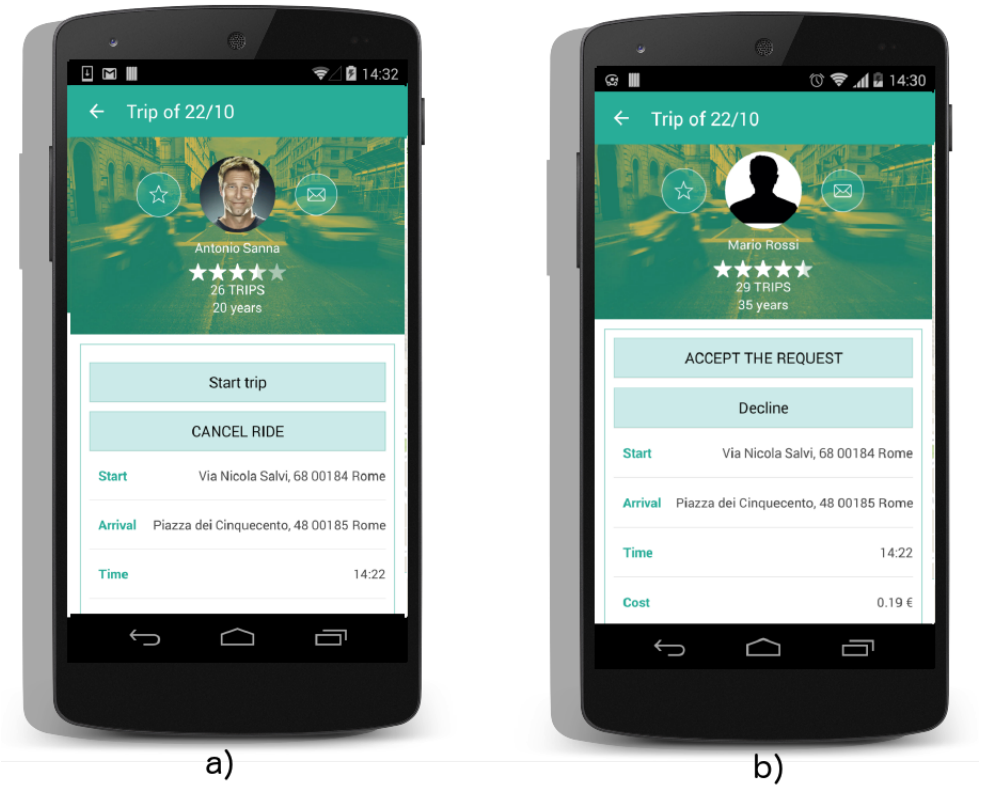

Figure 3.7: CLACSOON Android: Shared Ride Agreement

the check-in screen of the CLACSOON's mobile application. Also the conclusion of the ride has to be notified by the users with a similar check-out procedure. At the end of the shared ride, after the check-out has been done, the user are asked to give a feedback of their trip experience, i.e. a reciprocal evaluation between the two users which consists of a short message and an evaluation from a minimum of 1 to a maximum of 5 stars. Every feedback, along with the average of all feedbacks, is displayed in the public profile of the user who received the feedback.

Chapter 4

The Route Matching Algorithm

The previous chapter briefly described the CLACSOON platform, introducing the design choices that have been made in the design phase of both the client side and the backend side. The proposed solution is an application that automatizes the arrangement of the shared ride, automatically notifying the presence of suitable travel companions and suggesting the pick-up points and the meeting times. As stated in the previous chapter, the route matching algorithm has a key role for a dynamic ridesharing application. This chapter describes the implementation of the CLACSOON's route matching algorithm and its design choices. As it resulted from the section 2.4, many works in literature have focused on the optimization problem, but only few have worked on the modeling of the driver mobility to find better matches. Additionally, several works proposed a simplified model for ridesharing to make the optimization problem easier to be addressed. One common assumption in literature is that a shared ride must be agreed before the starting of the driver's trip. Furthermore, the partial ridesharing mode is not currently facilitated by matching agencies [13] and, at the best of my knowledge, its benefits have not been investigated in literature yet.

A former implementation of the CLACSOON's route matching algorithm didn't take into account the aforementioned considerations. This chapter describes the latest implementation of the route matching algorithm, that contemplates the partial ridesharing mode and proposes a method for estimating the varying position of the driver's vehicle in an urban context. The partial ridesharing mode avoids the driver to take a detour when possible, therefore it leads to an increment in the total system-wide CO_2 savings. By modeling the position of the driver's vehicle, only the remaining part of the route that a driver has to travel is considered when evaluating the matching: this approach enables the possibility for shared rides to be agreed on the fly after the starting of the driver's trip, when a rider happens to be close to the remaining part of a driver's route. This approach leads to an increment in the amount of total shared rides.

4.1 Scenario

During the design of the matching algorithm, we considered a scenario in which the back-end receives all the trip announcements for each participant. Recall that the service relies on the availability of a *Directions Provider*, which provides the information concerning the route between a departure and a destination location. This information includes travel directions, estimated path length, estimated travel time and the expected average travel speed, related to the type of roads, which may or may not depend on the historical average speed data over certain time periods. The proposed platform relies on the Google Maps Directions API[40] which is a service that provides directions between locations using HTTP requests. Nowadays such a service is provided by many agencies, and valid alternative is Bing Map [41], which calculates and display directions and routes on a Map with Direction API module or with Bing Map Rest Services. Several alternatives can also be used for those ride-sharing providers who opt for a self-hosted Direction Provider: a great example is The Open Source Routing Machine [42], which is a high performance routing engine written in C++ designed to run on OpenStreetMap data.

While specifying the details of the journey, as shown in section 3.4.2, a rider and driver request includes the desired departure and arrival locations. Each ride offer or request also includes a timeout which has to be intended as the maximum time the user is willing to wait before finding a mate. Furthermore, each announcement includes a search radius, which has to be intended as the maximum detour that the driver is willing to make from his/her original route or the maximum distance the rider is willing to walk to reach the pickup point. With this information, the route matching algorithm automatically establishes shared rides over time, matching potential drivers with riders.

For the purpose of describing the route matching algorithm, we assume that at a given time t:

- D is the set of drivers;
- P is the set of riders;
- $U = D \cup P$ is the whole population of the dynamic ridesharing system;
- t_d^{DEP}, t_p^{DEP} are the desired departure time for a driver d and a rider p, respectively;
- R_d is the search radius of the driver $d, \forall d \in D$, indicating the maximum detour from his/her scheduled trip that the driver is willing to travel;
- R_p is the search radius of the rider $p, \forall p \in P$, indicating the maximum distance the rider is willing to walk to reach a pickup point.

Furthermore, we assume that:

- $\vec{D}_d, \vec{D}_p$ are the coordinates of the desired departure location for the driver d and rider p, respectively;
- $\vec{A}_d, \vec{A}_p$ are the coordinates of the desired arrival location for the driver d and the rider p, respectively;
- $\vec{\alpha}_d$ is the desired route for a driver d, which connects $\vec{D}_d$ to $\vec{A}_d$. This information is provided by the Directions Provider and is encoded in a matrix of two columns where each row corresponds to a point in the path;
- τ_d is the estimated travel duration of $\vec{\alpha}_d$, provided by the Directions Provider;
- $\overline{V_d}$ is the average theoretical speed for $\vec{\alpha}_d$, provided by the Directions Provider;
- $s_d(t)$ is the number of spare seats for a driver d at the time t, and $s_d(0)$ is the initial vehicle capacity.

The problem of finding the matching between drivers and riders can be formulated as described in the following. The matching algorithm has to satisfy the following common constraints:

1. The total number of riders in a vehicle must not exceed the number of spare seats specified by the driver;
2. The entire commuting route must start at the departure and end at the destination locations specified by the driver;
3. Each rider must be picked up before he/she can be dropped off. This constraint seems obvious, but it must be made explicit in a carpool matching algorithm.
4. The maximum distance that a rider p has to walk for reaching the pickup point cannot exceed the search radius R_p;
5. The maximum detour that a driver d has to take respect to his/her route, for picking up a rider, cannot exceed the search radius R_d;
6. The rider and the driver can wait to find a matched mate for a shared ride at most the timeout T_p and T_d, respectively.

The constraints from 1 to 3 are usual for a commute process [43], while the constraints from 4 to 6 are specific for the proposed dynamic ridesharing system.

As mentioned previously, the potential route of a driver is encoded with a polyline $\vec{\alpha}_d$, which is a matrix with two columns where each row represents the coordinate of each point in the polyline. Accordingly

$$\vec{\alpha}_d = \cup_{i=0}^{n-1} \vec{\alpha}_{d_i} \tag{4.1}$$

where i indexes the points in the route and $\vec{\alpha}_{d_0} = \vec{D}_d$ and $\vec{\alpha}_{d_{n-1}} = \vec{A}_d$. The number of points in this matrix (n) is clearly variable and depends on the departure and arrival points, as well as on the route solution proposed by direction providers.

The proposed service is implemented in a way to require the minimal intervention from the users to maximize usability but at the same time giving him/her the freedom to chose among a possible list of mates. Therefore, this service finds all the matches and notifies the user with a list of suitable travel companions. The proposed matching algorithm works as follows: the algorithm first searches for one (or more) suitable matching and then, when the matching is found, the arrangement of the shared ride is proposed to the participants. The driver and the rider then can accept or refuse it. In most studies, the objective of the matching algorithm is the maximizing of the system-wide miles saved, the maximizing of the success rate (the percentage of satisfied drivers and riders), or the minimizing the waiting time of drivers and riders. Clearly, these objectives partially conflict each other. Depending on the policy of the ridesharing provider, one (or a combination) of the aforementioned objectives are selected for the implementation of the matching algorithm. In the proposed solution a weighting of the length of the shared trip and needed detour has been considered. The proposed Route Matching Algorithm relies on the following three sequential functions that are executed:

- *Temporal Matching*: for each new user (either a rider or driver), the system evaluates whether the time constraint is satisfied for each possible travel companions, given the timeout T, the driver's travel duration, and the current shared rides allocation, but without considering any geographical constraint;

- *Geographical Matching*: it is the evaluation of the matching between a driver and a rider on the basis of the distance from their paths. This step is performed for each pair (d, p) of drivers and riders that satisfied the previous matching. This step also takes into account the theoretical future position of the driver's vehicle, from the beginning till the end of his/her ride.

- *Cost Function Evaluation*: it evaluates the cost $C_{d,p}$ for a shared ride between each driver d and rider p that satisfied both the *Temporal Matching* and the *Geographical Matching* constraints.

The details of these steps will be explained in detail in the following sections.

The list of possible travel companions is then ordered by the value of the cost $C_{d,p}$. This result represents the output of the CLACSOON's matching algorithm. This list is then proposed to riders and drivers , as shown in section 3.4.4.

4.2 Temporal Matching

For the temporal matching it is necessary to consider the effect of the timeout (T_d and T_p), which is the maximum time the user is willing to wait to find a mate and after this amount of time the ride request is considered to be expired. For the drivers it is also important to consider the estimated travel duration τ_d, as after this amount of time the ride offer is considered to be over. In case the rider starts the ride after the driver, then the following two conditions must be verified:

$$t_d^{DEP} \leq t_p^{DEP} \leq (t_d^{DEP} + T_d) \tag{4.2}$$

$$t_d^{DEP} \leq t_p^{DEP} \leq (t_d^{DEP} + \tau_d) \tag{4.3}$$

which check that the rider arrives before the driver timeout and before the ending of his/her trip.

Differently, in case the driver starts the ride after the rider, the following condition must be verified:

$$t_p^{DEP} \leq (t_d^{DEP}) \leq (t_p^{DEP} + T_p) \tag{4.4}$$

which check that the driver arrives before the rider timeout.

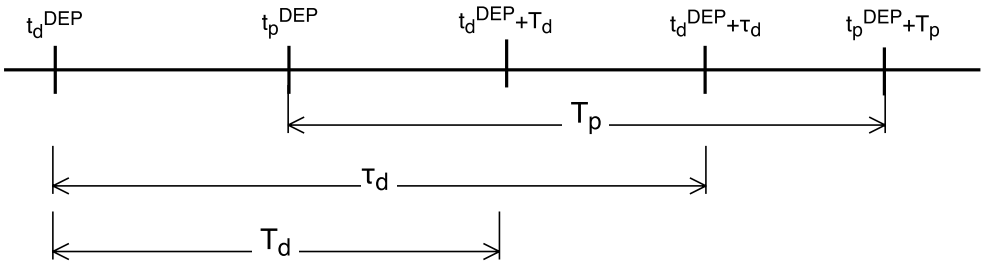

Figure 4.1: Temporal matching example

Each pair (d, r) that satisfies this step is then evaluated in cascade by the *Geographical Matching Algorithm*.

4.3 Geographical Matching

Each pair (d, p) that satisfies the Temporal Matching constraints is evaluated by the *Geographical Matching Algorithm*. For this purpose, we propose a method for estimating the driver's position at the time t on the basis of his/her destination. Modeling the position of the driver's vehicle enables for sharing rides even after the starting of the driver's trip. As

mentioned before, $\vec{\alpha}_d$ is the desired route for a driver d, which connects the point $\vec{D}_d = \vec{\alpha}_{d_0}$ with the point $\vec{A}_d = \vec{\alpha}_{d_{n-1}}$ (section 4.2).

Recall that n is the number of segments in the polyline. Assuming that τ_{d_i} is the travel duration between the point i and the point $i+1$, the total travel duration τ_d can be decomposed as the sum of the travel duration of each single segment of the route:

$$\tau_d = \sum_{i=0}^{n-1} \tau_{d_i} \tag{4.5}$$

For simplicity it can be assumed that

$$\tau_{d_i} = \frac{\tau_d}{n}, \forall i \in (0, ..., n-1) \tag{4.6}$$

We then choose to estimate the driver position at the time $t = t_d^{DEP} + \Delta t$ as the point of the route with index $K_d(t)$

$$K_d(t) = \lfloor \frac{\Delta t}{\tau_d} \rfloor \cdot (n-1) \qquad \text{if } t_d^{DEP} \leq t \leq (t_d^{DEP} + \tau_d) \tag{4.7}$$

Given that the constraints discussed in the section 4.2 have just been satisfied, note that this equation has to be considered in the range $t_d^{DEP} \leq t \leq (t_d^{DEP} + \tau_d)$, i.e. the position of the driver is considered to be undefined before the beginning of the ride and after the ride is over.

The remaining part of the path that a driver d has to travel at a time t can then be expressed as

$$\vec{\beta}_d(t) = \cup_{i=K_d(t)}^{n-1} \vec{\alpha}_{d_i} \tag{4.8}$$

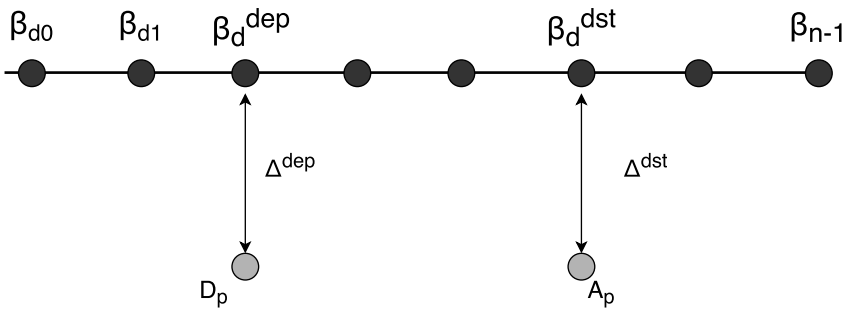

Figure 4.2: Geographical matching: graphical representation

Accordingly, $\vec{\beta}_d(t)$ is a subset of $\vec{\alpha}_d$ which does not contain the points with index $i < K_d(t)$ that the driver d should has passed by at time t. In other words, $\vec{\beta}_d(t)$ represents the part of the route that theoretically the driver has to travel after the time t. When evaluating the matching at the time t between the offer d and a request p only the remaining route

points $\vec{\beta}_d(t)$ are considered, against the departure $\vec{D}_p$ and destination $\vec{A}_p$ points of the rider: this feature enables drivers to pick up riders on the fly, if the pickup point is close to the remaining route points in $\vec{\beta}_d(t)$. This situation is depicted in Fig. 4.2. For the purpose of describing the *Geographical Matching Constraints*, assume that:

- $\Delta dep_{d,p}(t)$ is the minimum distance between the set of route points in $\vec{\beta}_d(t)$ and the rider's departure $\vec{A}_p$;
- $\Delta dst_{d,p}(t)$ is the minimum distance between the set of route points in $\vec{\beta}_d(t)$ and the rider's destination $\vec{D}_p$;
- $\beta_d^{\vec{dep}}(t)$ and $\beta_d^{\vec{dst}}(t)$ are the two points on the driver's route with the minimum distance from $\vec{D}_p$ and $\vec{A}_p$, respectively. These points represent the pick-up points on the driver's route.

A first constraint for the matching to be found is that the index of the point $\beta_d^{\vec{dst}}(t)$ has to be greater than the index of $\beta_d^{\vec{dep}}(t)$, so that the rider must be picked up before he/she can be dropped off.

To take the partial ridesharing mode into account, when the search radius R_p specified by a rider allows him/her to reach a departure pickup point on the driver's route, the system places the pickup point on the point $\beta_d^{\vec{dep}}(t)$. A similar method is used for the evaluation of the destination pickup point $\beta_d^{\vec{dst}}(t)$. In this way, this setting avoids the driver to take a detour when possible, and thus it is expected that it leads to an increasing of the total system-wide CO_2 savings. Depending on the values of the rider's and the driver's search radius, the matching algorithm assigns the departure pickup point $P_{\vec{dep}}(t)$ and the destination drop-off point $P_{\vec{dest}}(t)$ with the following method:

$$P_{d,p}^{\vec{dep}}(t) = \begin{cases} \beta_d^{\vec{dep}}(t) & 0 \le \Delta dep_{d,p}(t) \le R_p \\ \vec{D}_p & R_p < \Delta dep_{d,p}(t) < R_d \\ \uparrow & \Delta dep_{d,p}(t) > R_d \end{cases} \tag{4.9}$$

$$P_{d,p}^{\vec{dst}}(t) = \begin{cases} \beta_d^{\vec{dest}}(t) & 0 \le \Delta dst_{d,p}(t) \le R_p \\ \vec{A}_p & R_p < \Delta dst_{d,p}(t) < R_d \\ \uparrow & \Delta dst_{d,p}(t) > R_d \end{cases} \tag{4.10}$$

The three conditions in both the previous equations have been derived from the following motivations:

Condition 1) if the search radius of the rider R_p is greater or equal to the distance $\Delta dep_{d,p}(t)$ between his/her departure and the driver's route, the matching algorithm specifies the location $\beta_d^{\vec{dep}}(t)$ to be the departure pickup point.

Condition 2) if the search radius of the rider R_p is lower than the distance $\Delta dep_{d,p}(t)$, but the search radius of the driver R_d is greater or equal to the distance $\Delta dep_{d,p}(t)$ the pickup point is assigned to be on the rider's departure point D_p

Condition 3) if both Conditions 1) and 2) are not satisfied, a pickup point does not exist, an then a matching between p and d does not exist.

An analogous approach is followed in calculating the drop-off point.

The total driver's deviation[1] from his original path can be expressed as

$$dev_{p,d}(t) = w^D_{d,p} \cdot \Delta dep_{d,p}(t) + w^A_{d,p} \cdot \Delta dst_{d,p}(t) \tag{4.11}$$

where $w^D_{d,p}$ is a binary variable set to one if $P_{dep} \equiv D_p$, i.e. if a detour from the driver's original path is needed. Likewise, $w^A_{d,p}$ is a binary variable set to one if $P_{dest} \equiv A_p$ and set to zero otherwise. The last constraint to be satisfied is that the total detour that a driver should take in order to reach the pick-up and drop-off points can not be higher than the distance $\Delta km_{p,d}$ covered by the shared route, i.e. the shared ride provides positive cost savings. If this constraint is not satisfied, there wouldn't be a benefit for the driver to take the detour in order to share the ride.

$$dev_{p,d}(t) \leq \Delta km_{p,d} \tag{4.12}$$

If both $P_{dest}(t)$ and $P_{dep}(t)$ are defined, the matching is assumed to be found.

4.4 Cost Function

For the pairs of offers and requests (d, p) which satisfies the temporal and geographical constraints, the value of a cost function $C_{d,p}(t)$ for a shared ride is evaluated. It takes into account the following two elements: $\Delta km_{p,d}$, which is the length of the shared ride; $dev_{p,d}$, which is the length of the needed detour.

The cost function is defined as:

$$C_{d,p}(t) = f(dev_{p,d}(t), \Delta km_{p,d}) = \Theta \cdot dev_{p,d}(t) - \Psi \cdot \Delta km_{p,d} \tag{4.13}$$

Where Θ, Ψ are tuning parameters that respectively determines the importance of the detour from the original path and with respect to the travel share. Accordingly, a list of suitable travel companions is ordered, which are associated with:

- the departure pickup point $P_{dep}(t)$
- the destination pickup point $P_{dest}(t)$
- the cost $C_{d,p}(t)$

[1] for simplicity, the deviation has been considered to be in a straight line

The list of suitable travel companions for a user represents the output of the CLACSOON's matching algorithm. The user is then left with the option to select the best mates by using the mobile applications, according to his/her personal interests (section 3.4.4). In the next chapter on the performance evaluation, it will be assumed that the user always select the mate corresponding to the lowest cost function.

Chapter 5

Case study in the Cagliari urban area

The previous part of the thesis faced the challenges related to the design and the development of a Dynamic Carpooling platform. In particular, the implementation of the CLACSOON platform, along with its route matching algorithm have been described.

This chapter presents the case study that has been conducted in order to analyze the performance of the proposed matching algorithm in urban areas, along with the Quality of Experience (QoE) provided to the users. For this purpose, an emulation system has been developed, as the current population of CLACSOON users is limited and because we were interested in analyzing the performance with different population characteristics, which cannot be controlled in real scenarios.

A real area have been considered to emulate the mobility patterns in real urban conditions, including real roads in the city and real paths between any departure and destination (e.g. pedestrian zone, one-way roads, limited traffic zones). Three Key Performance Indicators have been analyzed: the number of shared rides, the waiting time to find a ride and the average total system-wide CO_2 savings.

In a preliminary work [7] the system with the major design choices has been presented. The performances have been evaluated in terms of passenger success rate and driver success rate, by varying the time distribution of the service requests.

In the paper [8] the study is more in-depth and in addition to the aforementioned KPIs, also the passenger waiting time and the total system-wide CO_2 savings saved have been evaluated, against the time distribution of trips (within a fixed time window), and the timeout (T) which is the maximum time that a user can wait before he decides to retire his/her offer of a ride or his/her request for a ride.

The paper [9] presented the improved version of the route matching algorithm, introducing the features described in chapter 4, and analogous experiments have been made to

evaluate the performances of the system service. For sake of clarity, in the following sections only the results from the latest study are presented.

The following sections describe the emulation system, present the experimental setup, analyze the achieved performance results, and provide a comparison with alternative approaches.

5.1 Emulation Setup

This section provides an accurate description of the experimental setup and the emulation system. The place selected to be the emulation scenario is the city of Cagliari, which is an Italian municipality with nearly 150.000 inhabitants, with a metropolitan area (including the surrounding 15 municipalities) of more than 420,000 inhabitants [44]. Considering a real area it is possible to emulate the mobility patterns in real urban conditions, including real roads in the city and real paths between any departure and destination (e.g. pedestrian zone, one-way roads, limited traffic zones). An agent-based emulator has been implemented to generate the ride offers and requests on behalf of real users, evaluate the matching between them and emulate the sharing of rides. The emulator is implemented in Java and it is based on the core of the CLACSOON platform (indeed the matching is exactly the service in production but executed in the emulation environment). In the experiments several *scenarios* have been run, each one characterized by a combination of parameter settings as explained in the following. Three Key Performance Indicators have been analyzed: the number of shared rides, the waiting time to find a ride and the average total system-wide CO_2 savings. In the following, we describe the processes we followed for the *Configuration*, *Setup*, *Run* and *Evaluation* phases in our experiments.

Configuration

During the *Configuration* step, a list of scenarios is generated: each scenario represents the configuration of a population. During the experiments some parameters of the population have been changed to evaluate the effects on the KPIs; these parameters are listed in Table 5.1. The performed experiments have been conducted by selecting an area of interest in the city of Cagliari (39.23,9.14) and with an area (A) of about 64 kmq, which is where the users can operate. This area is of interest for this study since the majority of the CLACSOON'S users mainly operates inside this boundary. Furthermore, this area is representative of medium-small cities with numerous residential areas, commercial sites, factories and historic neighborhoods within its metropolitan boundaries. Fig. 5.1 shows the area selected for this case study where the area of interest is delimited by a black line. Each run lasts for S hours during which a total of N users act as either passenger or driver. When evaluating the performance of the system with respect to the spatial $clacsooners$ density, both the population density

(N_k) and the ratio between the number of drivers and the number of passengers (L_d/L_p) have been varied. As shown in the table, N ranges from 600 to 2500, which correspond to a different population density given the size of the reference geographical area, and the ratio L_d/L_p ranges from 1/8 to 8. In the performed emulations, we also refer to the timeout T, that ranges from 1 to 30 minutes. For simplicity, we assume the same timeout T for each rider and each driver.

System parameters		
Time window	S	4 hours
Total users	N	from 600 to 2500
Population percentage	N_k	from 10 to 40 users/kmq
Number of drivers	L_d	L_d/L_p from 1/8 to 8
Number of passengers	L_p	
Temporal rate of ride offers	f_d	from 10 to 600 users/hour
Temporal rate of ride requests	f_p	
Timeout	T	From 1 min to 30 min
Search radius of passengers	R_p	300 m
Search radius of drivers	R_d	1/10 of the travel length
Cost function tuning parameters	Ψ, Θ	$\Psi/\Theta = 1$

Table 5.1: Values of parameters varied during experiment

Each scenario represents a combination of the parameters listed in Table 5.1. To perform the simulations proposed in this case study, the KPIs have been evaluated for 3 sizes of the population N, 8 levels for the ratio L_p/L_d and 8 values for the timeout T, for a total of 192 scenarios.

5.1.1 Setup

The *Setup* step consists of the generation of each member of the population for a single *Run*. During this step, the total population N is divided into L_d drivers and L_p passengers. The emulator assigns each user a departure and a destination locations chosen randomly and uniformly within the selected area, with the following two constraints:

- both locations falls into a street
- it is actually possible to travel from departure to destination, i.e. a path exists between the departure and the destination locations.

At each user it is also assigned the shortest path between the departure and destination points, which is calculated by the Directions Provider.

Generating random paths within this area leads to an average travel duration of approximately 13 minutes with a standard deviation of approximately 6 minutes (fig. 5.2). The

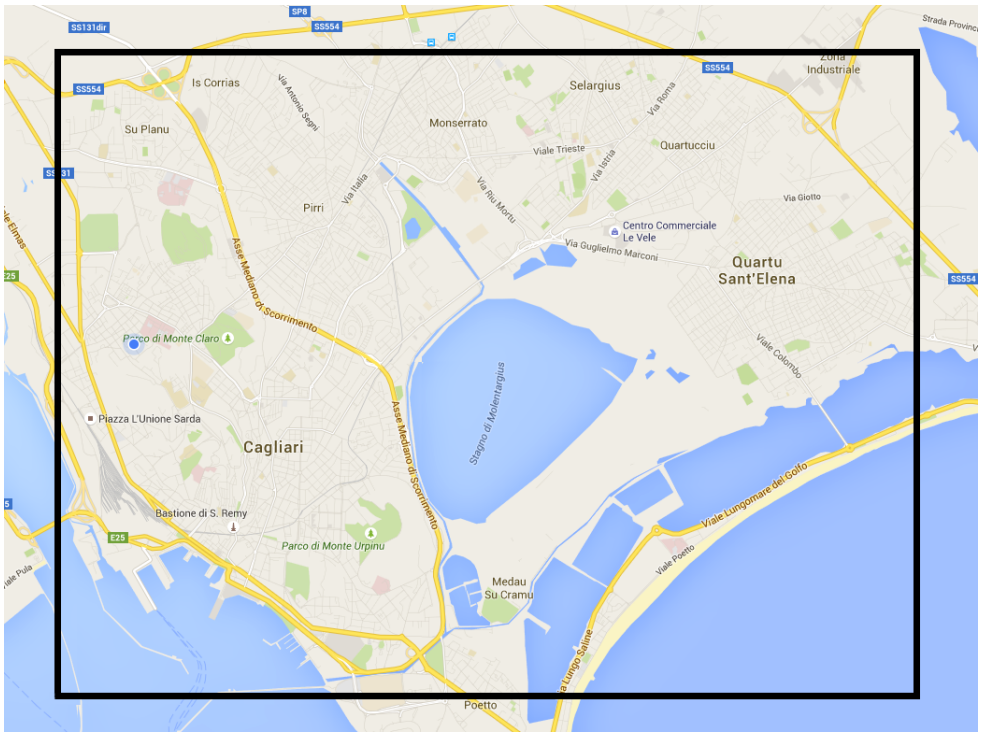

Figure 5.1: The area for the case study

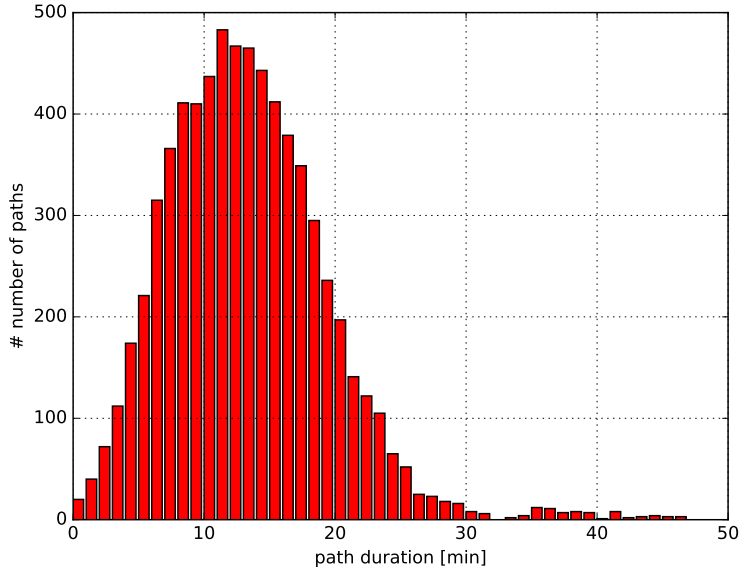

Figure 5.2: Average travel duration within the selected area

emulator also assigns, for each driver and each rider, the desired departure time t_u^{DEP}. The time interval between two successive departure times is set to have an exponential distribution within the time window S. Each run lasts for a fixed time window S. If we consider this period and a given number of drivers L_d and a number of riders L_p, we obtain an expected time interval between offers μ_d and an expected time interval between requests μ_p:

$$\begin{cases} \mu_d = \dfrac{S}{L_d} \\ \mu_p = \dfrac{S}{L_p} \end{cases} \tag{5.1}$$

5.1.2 Run

Once the population's details have been set, the *Run* step is executed. Each *Run* for a given scenario has to be repeated for 20 cycles, in order to reduce the width of the confidence interval. In particular, we checked the 95% confidence interval for one of the most important KPI, i.e., the passenger success rate whose results are shown in Fig. 5.3, and we checked that it was very small for so that we almost had no overlaps among the curves. Specifically, it was lower than 0.01, which was very low. The emulator models a situation in which an user u joins to the population at his desired departure time t_u^{dep}: this step simulates the publication of a trip offer or request trough the CLACSOON mobile application. Strictly after the user u joins the population, the matching algorithm is evaluated between the user u and the set of complementary users (if the user is a driver the matching is evaluated against a set of riders, and vice versa). If no matching is found, the user is given a time of τ_u to be contacted by another user. When a new rider joins to the population, a set of offers that satisfy the constraints specified by the *Temporal Matching* is retrieved from the database. These offers are then evaluated by the *Geographical Matching* algorithm. If one or more offers satisfy the *Geographical Matching*, the value of the *Cost Function* is evaluated for these offers, and the ride is agreed with the offer which leads to the less cost. Moreover,the number of empty seats for the offer is reduced by one and the ride request is marked as *busy*, i.e. it won't be possible for other drivers to give a lift for this request. On the other side, when a new driver joins to the population, the list of the existing ride requests is retrieved from the database. Those rides have to satisfy the constraint of not being *busy*, i.e. the ride request is not yet committed to another driver. If the aforementioned constraint is satisfied, then the matching between the offer and the request is evaluated by the *Temporal Matching* and the *Geographical Matching* algorithm. If a matching is found, the shared ride is considered to be agreed, the ride request is marked as busy and the spare seats for the ride offer are decremented by one.

5.1.3 Evaluation

After the end of a *Run*, the following Key Performance Indicators are computed:

- **Passenger waiting time** i.e. the average of the time that the riders that found a matching had to wait before finding that matching
- **Passenger success rate** the percentage of riders that found a ride
- **Driver success rate** the percentage of drivers that shared a ride
- **Total System-wide CO_2 saved** the sum of the estimation of the CO_2 saved for each shared ride, based on the distance that the riders should have traveled alone if the shared ride was not agreed

The results of the emulations from this case study have been computed as an average of the KPIs obtained from 20 runs for each scenario. As mentioned before in the section 5.1 a total numbero of 192 scenarios has been executed, leading to a total number of about 4000 runs.

5.2 Experimental Results

The performances, in relation with the time distribution of the service utilization, have been evaluated varying the rate of ride offers (f_d) and the rate of ride requests(f_p). The performance has also been evaluated in relation with the maximum waiting time of the users T.

Figs. from 5.3 to 5.6 shows the results according to the mentioned KPIs varying the parameters with the values indicated in Table 5.1.

5.2.1 Passenger success rate

Fig. 5.3 shows the passenger success rate, which is the percentage of passengers that find a ride. This chart shows the trend of this indicator in relation with the timeout, for three levels of the ratio L_d/L_p and three levels of the population N.

The first highlighted trend is that the success rate increases with the population, which is something expected since if the spatial density of users is low, the probability to have a matching is small as well. The growth in relation with the timeout is significant until the value of T is around 15 minutes; after this value any further increase in the timeout doesn't have a big impact. This is due to the fact that the random paths generated in the selected area results in an average travel length of 13 minutes. This value of T is comparable to the bus transit frequency within the city of Cagliari, then it can be states that this amount of time is likely to be acceptable for riders. Considering a threshold in the success rate of 80%, it can be seen that this threshold can be achieved with a balance between the numbers of drivers

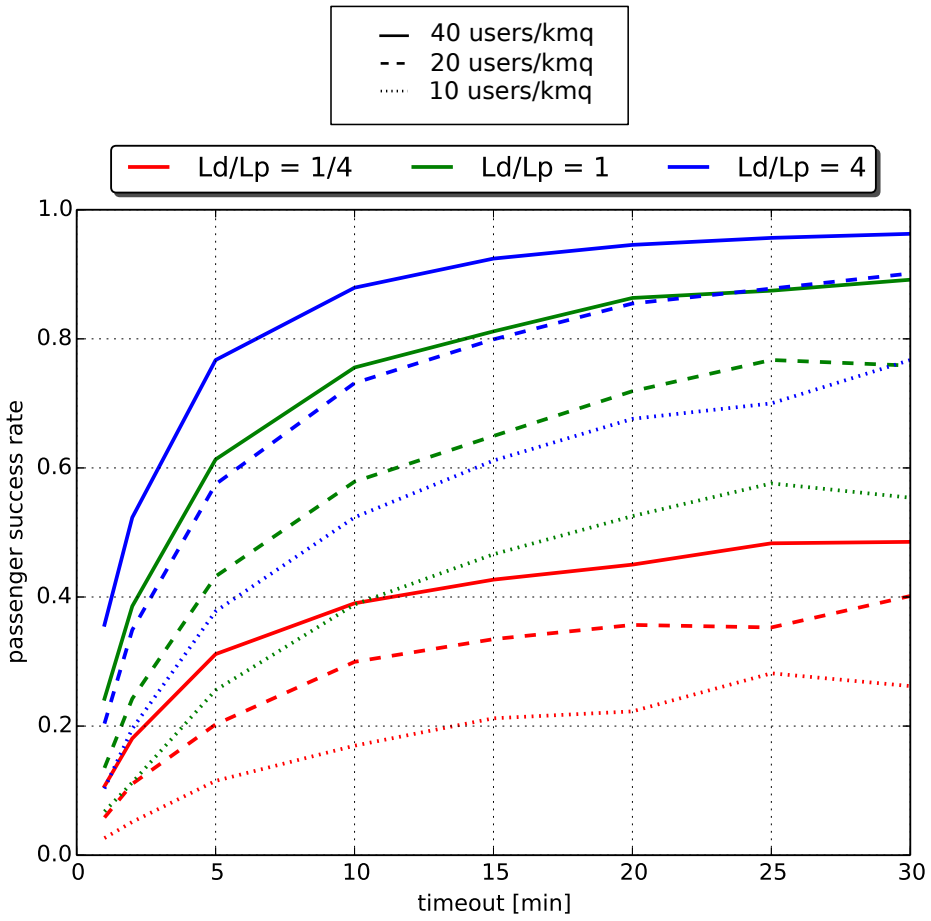

Figure 5.3: Passenger success rate

and of passengers (i.e., $L_d/L_p = 1$) if the later have the patience to wait for up to 13min in case the $clacsooner$ density is of 40 users/kmq. Otherwise, if the percentage of drivers is high to have a ratio $L_d/L_p = 4$, then the passengers would have to wait only 6 min. This result tells us that depending on the patience of the customers a different marketing actions should be followed to reach the needed percentage of drivers in the $clacsooner$ population.

5.2.2 Driver success rate

Fig. 5.4 shows the driver success rate. The first highlighted trend is that the success rate increases when the population increase and when the ratio L_d/L_p decreases (i.e., the longer the number of riders request a ride, the higher the driver success rate).

The Figs. 5.3 and 5.4 show that, with the same population level, the rider success rate is higher than the driver success rate. This difference is related to the different nature of these two agents: a single driver could give a ride to more than one passenger. This situation is more likely to happen if the number of drivers is higher than the number of riders and when the number of users is high.Moreover, an increase in the timeout always leads to an increase in the success rate for riders, but for drivers this effect is limited: the driver success rate in-

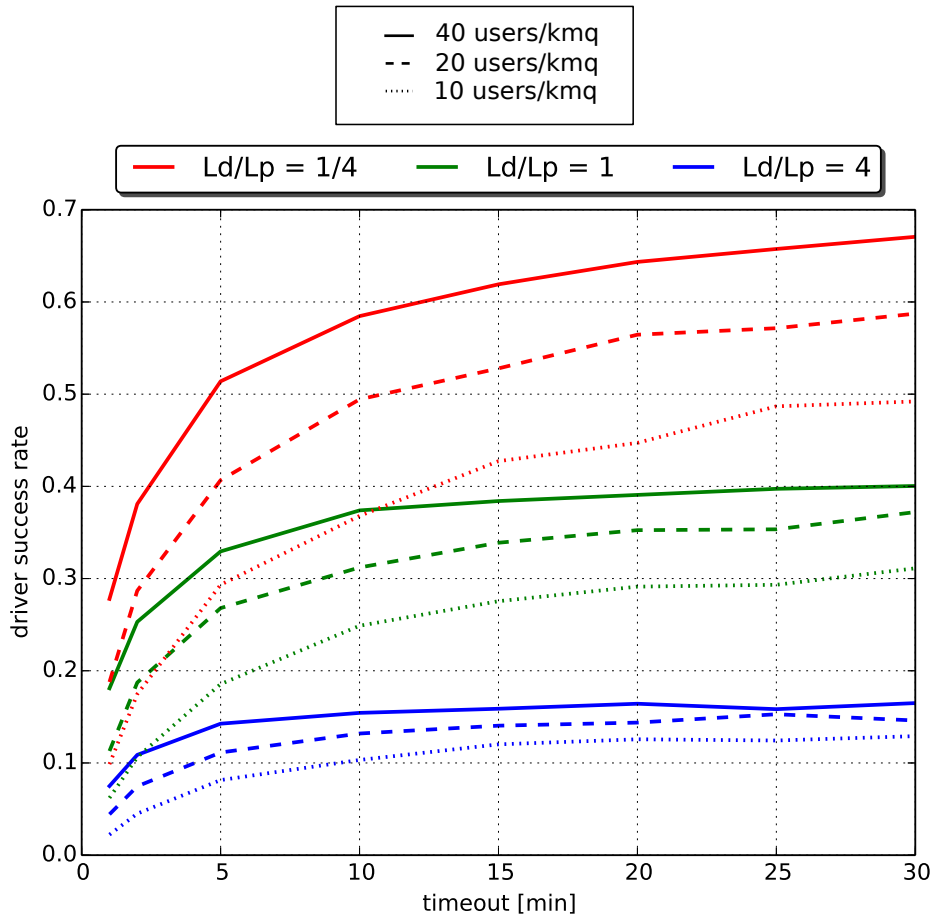

Figure 5.4: Driver success rate

creases slowly after 15 minutes. This is due to the fact that the random paths generated in the selected area results in an average travel length of 13 minutes. The emulator models a situation in which riders insert their trip when they arrive in the desired pickup point and then they can wait for a matching at most until their waiting time reaches the timeout. Drivers, as opposite to riders, insert their trip when they are ready to start the trip and they could obtain a matching at most until their trip is over. So, a further increase in the timeout doesn't lead to a big benefit for drivers. Moreover, when L_d becomes higher than L_p and for a high level of population, there is another important trend. The system results unbalanced in favour of the riders, so most of them can find a ride. The remaining drivers have low probability to find a passenger, because the majority of passengers have just agreed to take a ride from a driver.

5.2.3 Passenger waiting time

Fig. 5.5 shows the rider's average waiting time needed to find a ride. The trend is linear and decreases when the number of drivers increases. If the ratio L_d/L_p is high (i.e., more drivers than riders are in the system) the waiting time is low and vice versa. In case there are more

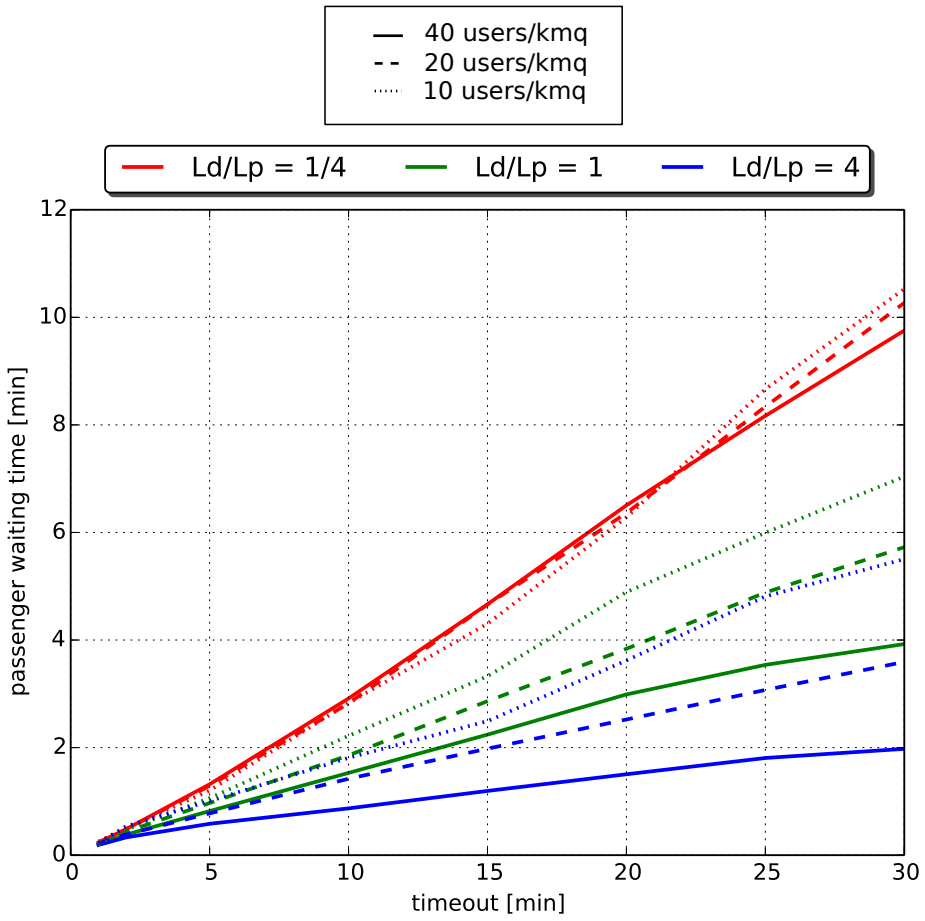

Figure 5.5: Passenger waiting time

drivers than riders and then there are many offers, the probability to find a ride becomes quickly high . So, if the number of driver is higher or equal than the number of passengers, the waiting time increases slowly.

This figure doesn't consider the waiting time of rides that haven't found a ride (in fact, a waiting time for these rides is undefined), so this figure has to be considered in conjunction with the fig. 5.3.

5.2.4 Total system wide CO_2 saved

The average success rate and the waiting time represent the performance from the single trip point of view. By collecting the travel length of each shared ride, the global system CO_2 saved has been computed. The result is shown in Fig. 5.6. This chart shows the trend of this indicator in relation with the the ratio L_d/L_p, for three levels of the population N and for three levels of the timeout T. It's important to note that the CO_2 saved is a KPI that describes the performance of the whole system.

The CO_2 saved is estimated as the product between the total travel shared (evaluated in km) and the average CO_2 emitted by a car (140 grams of CO_2 per km) [45] . Since this pa-

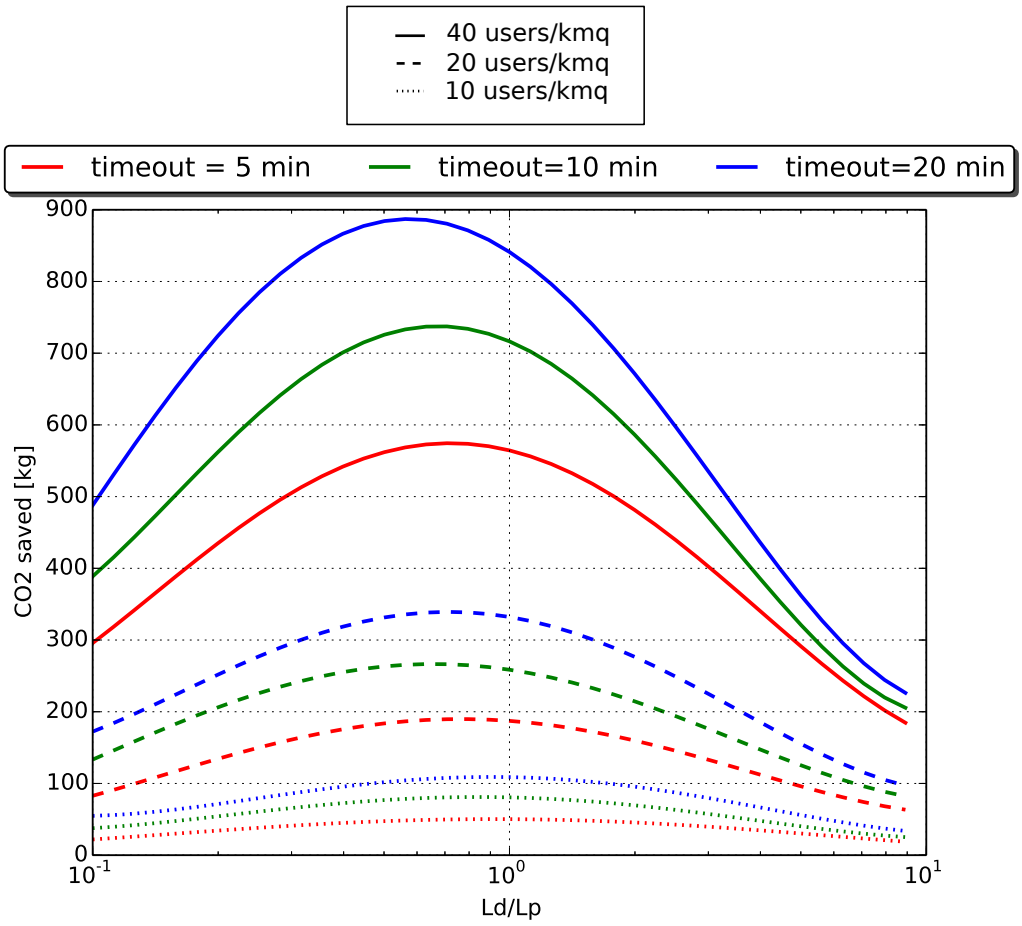

Figure 5.6: Total System-wide CO_2 savings

rameter is assumed to be proportional to the shared travel length, it is also representative of the total cost savings generated by the carpooling system. The curve reaches the maximum value when the number of riders is close to the number of drivers, that is when the system is balanced. It increases when the timeout increases as well as when the population increases.

Assuming the aforementioned value of CO_2 emitted per km, the value of emission savings can be computed in the time window S used in the simulation. It is notable that with a timeout value of 10 minutes and for $L_p \approx L_d$ the emission savings in this scenario are about 80 kg for 10 users/kmq, 250 kg for 20 users/kmq and 720 kg for 40 users/kmq. It is clear that the trend is not linear but follows an exponential increase with respect to the population.

5.3 Performance Comparison

Following the approach done in [22], to assess the value of the CLACSOON's matching algorithm, in this section its performance have been compared with those of an alternative matching algorithm I have developed (following named "DUMMY"), which presents the two following simplifications with respect to the CLACSOON's matching algorithm:

- when a matching is found the pickup and the drop-off points are assigned to be the rider's desired departure ($\vec{D}_p$) and destination ($\vec{A}_p$) points, respectively
- a driver cannot accept requests after his/her trip started: each shared ride must be agreed before the starting of the driver's trip, and the dynamics of the positions and the speeds of the vehicles are not taken into account

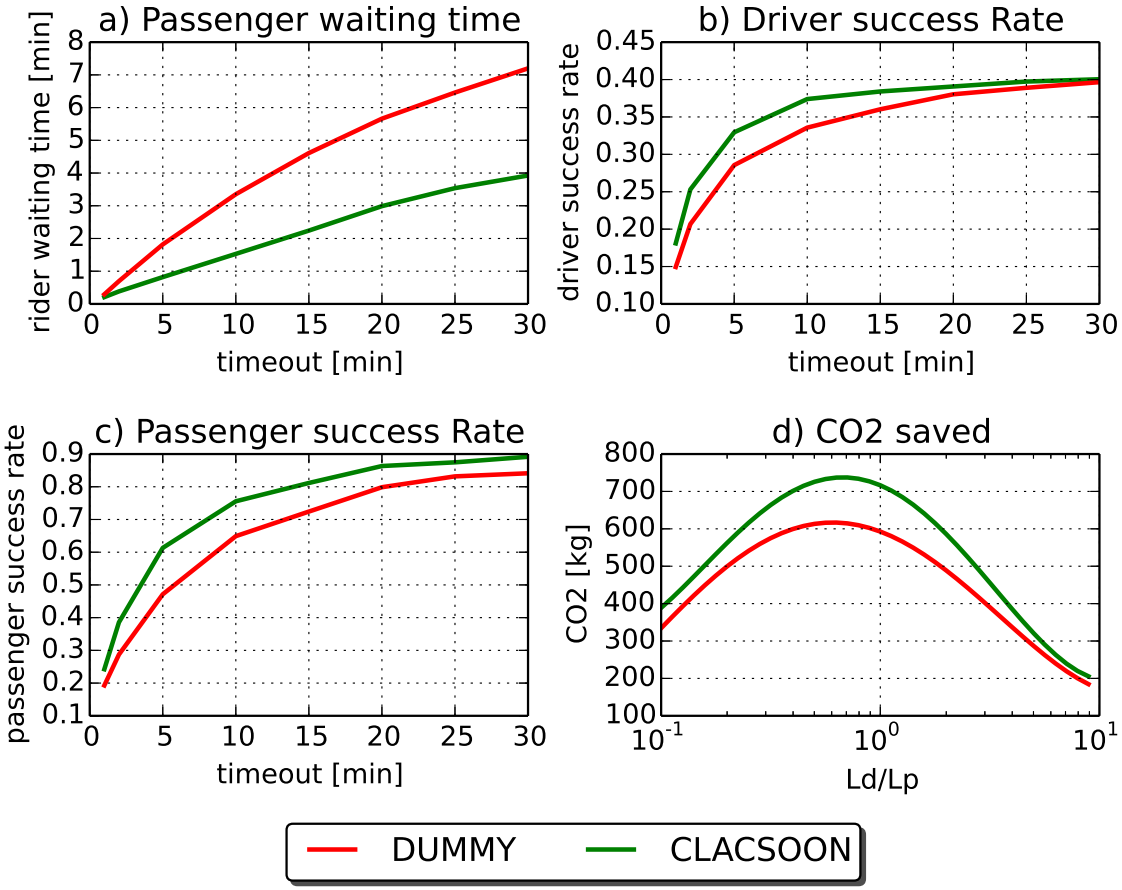

Figure 5.7: Comparison with the "DUMMY" matching algorithm

Note that the DUMMY algorithm does not only contemplate the Identical Ridesharing (i.e. when the departure have to be the same for riders and drivers). Indeed, the DUMMY matching algorithm also contemplates the presence of intermediate meeting points that can: i) be on the original route of the driver, ii) be not on the way of an original route of the driver, so that a detour would be needed to reach up the pick-up and drop-off points. The difference with the CLACSOON algorithm is that, when rider's desired departure and destination locations are not on the way of the driver's original route, the driver should have to take a detour to reach up the rider's departure and destination points. Therefore the DUMMY

matching algorithm does not contemplate the partial ridesharing mode and prevents drivers from picking up riders on the fly, even if the pickup points result on the driver's route.

This comparison is intended to specifically evaluate advantages of the proposed algorithm.

The following results shows the comparison of the indicators for a population density of *40 users/kmq*. In the following, if not explicitly stated otherwise, the simulation environment parameters are equal to those listed in Table 5.1. Fig. 5.7 clearly demonstrates that the CLACSOON matching algorithm performed better than the DUMMY matching algorithm in terms of all the Key Performances Indicators that have been computed. Fig. 1.a) to 1.c) shows the computed results in relation with the the timeout T and for $L_d/L_p = 1$, i.e. the number of riders is set to be equal to the number of drivers. For instance, for a Timeout of 10 minutes, the CLACSOON matching algorithm leads to a decrease in the waiting time of around 55% and an increase in the driver and passenger success rates of around 4% and 10%, respectively. However, note that the relative advantage for the success rate, decreases with the timeout for both drivers and riders. Fig. 5.7.d) shows the total system-wide CO_2 saved for a timeout of 10 minutes, in relation with the ratio L_d/L_p. It is notable that, for $L_p = L_d$ the CLACSOON algorithm leads to an increase (+21%) of the CO_2 saved, which corresponds to 120kg of CO_2 emission savings over the value computed for the DUMMY matching algorithm. For the selected scenario, the overall CO_2 that riders would have emitted if each one had driven his/her car (estimated on the total length of their desired route) is estimated to be approximately 1150kg, so a participation rate of *40 users/kmq* leads to the 64% of emission savings for the CLACSOON matching algorithm and the 54% of emission savings for the DUMMY algorithm.

Therefore the achieved results demonstrate that the CLACSOON matching algorithm performs better than the DUMMY matching algorithm in terms of all the Key Performances Indicators that have been computed: the success rate, the waiting time to find a travel companion and the CO_2 emission savings.

Chapter 6

Dynamic Involvement of Real World Objects in the IoT

In the previous chapters we faced with the issues of a dynamic ridesharing service in an urban scenario. This type of service is considered as one of the approved ways, in the field of smart mobility, to mitigate the problems generated by the population increase, while improving the quality of life in the smart city context. Thanks to the coming of the IoT paradigm the *carpoolers'* vehicles, which may or may not be involved in a shared ride, can represent a good sensor platform (section 3.2). Since one of the aspects of smart cities is the optimal use of the resources, the information coming from sensors integrated with real-time monitoring systems can be processed and analyzed to make certain adjustments as per the need, to improve the efficiency, quality, performances and resource consumption of the urban services. A great example for a smart city is Santander: this Spanish city is one of the smartest cities in the world, thanks to the use of the Internet of Things (IoT) [46] . The city is embedded with more than 12,000 sensors [47], installed and deployed both at static locations (streetlights, facades, bus stops) as well as on-board of mobile public vehicles (including buses, taxis and police cars [48]), to retrieve environmental parameters associated to determined parts of the city, such as the air pollution levels and traffic conditions. These technologies also provide an opportunity to collect the information about urban mobility, in the form of participatory sensing [49] [50].

Within the carpooling scenario presented before, the sensing devices could be installed on-board of the driver's vehicles to acquire information from the surrounding ambient and to cooperate with other devices to achieve a common application goal. This is the case for instance of different cars that are moving in a given urban area that is affected by different congestion points and that can share the knowledge about the status of the roads so that they can better find the route that minimizes the driver objects, typically expressed in terms of expected time to reach the destination. They can also share information about the park-

ing lots occupancy so as to reduce the time needed to park the car. Another situation is the one in which devices that are located in the same geographical area share the knowledge about the temperature so that the IoT applications can benefit from a more accurate view of this physical magnitude. In this context, Machine to Machine (M2M) and Device to Device (D2D) communications play an important role. To design a service of this type, the problem of the limited resources for the IoT nodes must be taken into account. Following this considerations, in [10] we presented a study of a distributed algorithm for the task allocation and assignment, which can be executed by a group of real IoT devices. It is based on a consensus approach, with the aim of maximize the lifetime of groups of nodes involved and ensures the fulfillment of the requested Quality of Information (QoI) requirements. In the next sections an overview of the basic concepts of the IoT will be given, and then the proposal is explained in detail.

6.1 The role of Virtual Objects in the IoT

The last few years have been involved by the technological revolution represented by the Internet of Things (IoT) [46]. The IoT paradigm relies on the interconnection of devices with different capabilities such as sensors, actuators, Radio Frequency Identification (RFID) tags, smart objects (e.g. smartphones), and servers, within the same heterogeneous network. The aim is to enable the network objects to dynamically cooperate and make their resources available, in order to execute complex applications and services. Nowadays, this high attention on the IoT topic stimulates industry and research to invest a lot of resources in this emerging field, consequently IoT is became a hot research topic, as demonstrated by the increasing attention and the large worldwide investments devoted to it.

In the Internet of Things (IoT) vision, even the most common and simple object is expected to acquire information from the surrounding ambient and to cooperate with other objects to achieve a common application goal, fulfilling the expected quality requirements.

This is also fostered by the widespread adoption of cloud computing technologies to augment capabilities of simple and cheap devices to take part to the deployment of complex applications [57], especially through the introduction of the Virtual Object (VO) [58] concept, which is the digital counterpart of a physical entity[59]. According to [60], the physical components of an object can be abstracted and made available as virtual resources, inheriting all their functionalities, characteristics and acquired information. Virtualisation allows the higher layers of the IoT architecture to:

i) interface with devices;

i) provide devices with the required commands, adapted to their native communication protocol;

i) monitor their activities and connection capabilities.

In the depicted scenario, it may happen that a group of devices have in common the capability to perform the same tasks (e.g., sensing the quality of the air in a given geographical area), which entails for a procedure to decide about their involvement when an application requires the execution of this task. This procedure is typically implemented by the VOs, following either a centralized or decentralized approach. According to the former, the procedure runs in the cloud, which needs to be constantly updated about the status of the objects. According to the latter, the devices directly interact each other and agree on the best solution without the involvement of the central platform.

In the past few years, many well-known IoT middleware architectures based on virtualisation of real objects have been proposed [58]. The management and resource allocation of these objects is usually committed to the power of cloud computing, which ensures high reliability, scalability and autonomy to provide ubiquitous access

However, reaching the cloud to manage network nodes is not always a good solution, especially for real-time applications, nor is it a convenient solution in terms of energy consumption.

This issue is partially solved in [61], where fog computing is used to virtualise real world object characteristics and resources, and to allocate application tasks to them, forming a decentralized computing infrastructure in which the resources are distributed between the data sources and the cloud.

Resource allocation has been extensively studied in Wireless Sensor Networks (WSNs), particularly with reference to network lifetime. In [62] a distributed task allocation that focuses on the reduction of the overall energy consumption and task execution time into a heterogeneous WSN is proposed, with attention to nodes' residual energy. A similar approach is studied in [63], where a distributed algorithm based on particle swarm optimization is proposed. Since the main criticality of wireless networks is their lifetime, all these algorithms mainly focus on maximizing this resource. Nevertheless, IoT nodes have more heterogeneous characteristics and capabilities, included residual memory, processing capacity and throughput.

As far as IoT networks are concerned, distributed resource allocation is an open issue. Most of the existing studies on resource allocation for IoT are focused on IoT service provisioning, such as in [64] and [65]. In these studies, the aim is to allocate the resources that enable service execution. However, they do not focus on finding the best configuration that corresponds to an optimal resource allocation. None of the works found in the literature tries to find the optimal resource allocation associated to the lowest impact of the application assigned to the network. Additionally, QoI is not taken into account [66].

6.2 Reference Scenario and Problem Statement

The following sections focus on the decentralized approach and propose a consensus-based sensing allocation algorithm, which has a twofold objective:

i) considering Quality of Information (QoI) constraints in the process of allocating tasks to the IoT objects, so that the fulfillment of application requirements is ensured;

i) optimizing the use of resources of the underlying IoT system by maximizing the lifetime of the group of devices involved.

In the following we refer to the Cloud-based IoT model that relies on virtualization technologies [67] and includes three levels, as shown in Figure 6.1. A Real World Object (RWO) is a device that has the ability to observe the real world phenomena and to perform measurements or operate on other objects. The Virtual Object [58] is its digital representation and guides its involvement in the implementation of the deployed IoT applications, by providing a description of the RWO, with also semantic enrichment. It also supports discovery and mash up of services, improving the objects energy management efficiency, as well as addressing heterogeneity and scalability issues. Each VO is instantiated with a template that should match the type of RWO it is associated to (e.g., smartphone model, embedded device type, temperature sensor). Depending on the RWO capabilities and use-cases, the VO processes are run in the cloud, gateway or RWO physical devices. Scenarios where the VO functionalities are distributed among these locations are also possible. The Service Level receives user application requests and maps them dynamically to the appropriate VOs, which take in charge their accomplishment by involving the relevant RWOs.

These deployment processes need to take into consideration the applications' QoI. QoI is the characterisation, in terms of some salient attributes represented in the form of metadata, of the goodness of the data collected, processed and flowing through a network [68]. QoI concerns the information that meets a specific user's need at a specific time, place, physical location, and social setting. Some examples of QoI requirements are data sampling rate, precision, and provenance [69].

An important component of this architecture is the information model, which is implemented by the VO template and encodes all the information that is used for their appropriate involvement in the IoT application deployment and delivery. A great effort in the definition of the information model has been done by the iCore FP7 project [70].

However, in this work there is the need to extend this model in order to take into account the mobility of objects, their temporal features and their characteristics of QoI. This enhancement is meant to improve the VO search, discovery and selection processes that enable the tasks assignment to the most appropriate VOs, with a QoI-oriented perspective. Figure 6.2 shows the new elements in dashed border boxes:

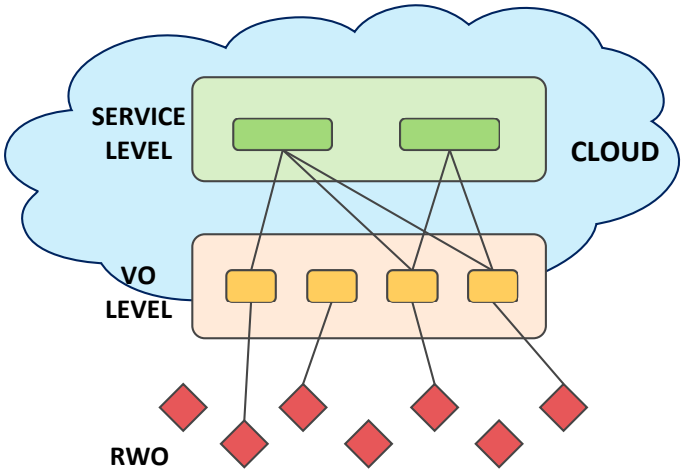

Figure 6.1: The reference IoT cloud architecture

- **Temporal features**: The use of the temporal features, both in terms of date and time range, allows to know the activity phases of a device associated with its VO. Knowing the date and time in which a mobile device is located in a given place, helps the association process among ICT and non- ICT object. It also ensures the ability to know in advance when a particular resource is available, when it is possible to refer to it, and how long it has not been updated.

- **QoI Parameters**: The information model, on which the selection processes are based, includes a field dedicated solely to the QoI parameters. The values in this field are named uniquely based on their characteristics. In addition, it introduces their descriptive aspects, that allow their identification. The parameters stored in this field will therefore be examined in the selection phase and allow an optimised choice of the resources to use.

- **Indoor location**: It is particularly useful in cases of closed environments. This could be an element that enhances the scalability of the system. It permits to the model to be used not only in large-scale distributed environments (metropolitan areas or neighborhoods), but also in small size environments and internal locations (such as buildings or structures in which a geo-localization of the nodes is not enough).

When a new application has to be deployed, the service layer sends a request to the VO level to search, among the available VO instances, those that are able to perform the relevant tasks based on the appropriate templates and other parameters (e.g., position, ownership). To this the information model becomes vital to implement an effective search function. As a result, for each requested task k, a group of VOs capable of performing it are identified. At this point there is the need to decide how they should contribute to the execution of this task while considering the required QoI level. The decentralized approach which is followed in this proposal has the advantage of being able to better follow the changes of RWOs status.

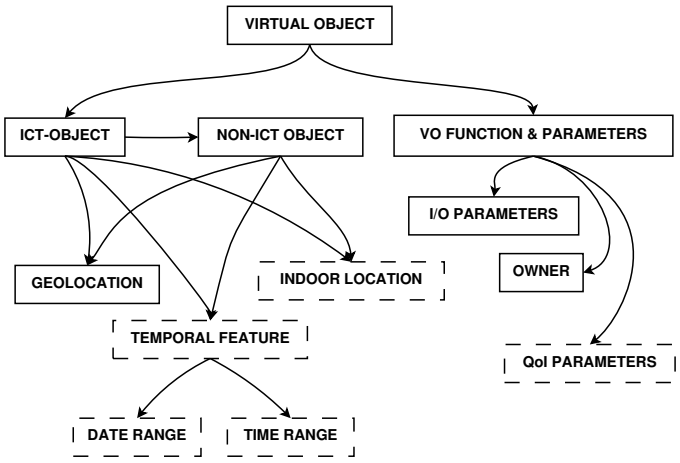

Figure 6.2: VO Information Model used. Solid border boxes correspond to elements included in the iCore VO Information Model. Dashed border boxes are new elements introduced by the proposed architecture

This is possible when the VO task allocation functionalities are implemented in the RWO or gateways, in case they do not have the sufficient computing power and the fog/edge computing technologies are used [71]. This also requires the RWOs in the group for task k and the gateway (if needed) to form a connected group relying on short-range communication technologies. In the following we consider that the target QoI level is a required execution frequency F_k^{ref}. However, the proposed solution can be generalized to other QoI requirements.

6.3 The Resource Allocation Model

6.3.1 Resource Model

This section provides a description on the main resources that represent an issue for IoT systems: object lifetime, storage capacity, processor and data throughput.

Lifetime

As defined in [72], the lifetime of a node is defined as the time until it depletes its battery. Applying this to this case, the lifetime of the node associated to VO i at time t is

$$\tau_i^{lftm}(t) = \frac{E_i^{res}(t)}{\sum_k E_{ik}^c \cdot f_{ik}(t)} \tag{6.1}$$

where $E_i^{res}(t)$ is its residual energy, E_{ik}^c is the energy consumed by the RWO associated to VO i to perform task k, and $f_{ik}(t)$ is the frequency at which VO i performs task k. This means that the lifetime of a node depends on the frequency at which the tasks assigned to it are performed.

Storage capacity

The storage capacity of a node decreases according to the frequency at which data are stored in it, and to the amount of data stored. Analogously to the definition of node lifetime, the storage capacity depletion time of the node associated to VO i is defined as

$$\tau_i^{stor}(t) = \frac{M_i^{res}(t)}{\sum_k D_k \cdot f_{ik}} \tag{6.2}$$

with $M_i^{res}(t)$ residual memory expressed in bits, and D_k amount of data to be stored for task k. Note that residual memory can change over time, not only because of its usage, but also because its stored data can be moved to another location.

Processor

The time needed to perform a task is in inverse proportion to the processing speed of the node that is performing it, and in direct proportion to the number of instructions required by the task. Calling t_{ik}^{exec} the time needed by the node associated to VO i to perform task k, it can be stated that

$$t_{ik}^{exec} = \frac{N_k^{instr}}{S_i^{proc}} \tag{6.3}$$

where N_k^{instr} is the number of instructions that need to be processed to perform task k, and S_i^{proc} is the processing speed for the node associated to VO i. If task k is performed at a frequency $f_{ik}(t)$, this means that the processor of the node associated to VO i will be busy for a ratio of time equivalent to

$$\theta_{ik}^{proc}(t) = t_{ik}^{exec} \cdot f_{ik}(t) \tag{6.4}$$

Generalising for all the tasks performed by i, the total processor occupancy is defined as

$$\Theta_i^{proc}(t) = \sum_k t_{ik}^{exec} \cdot f_{ik}(t) \tag{6.5}$$

which is the ratio of time for which the processor is busy, considering all the tasks.

Bandwidth

Analogously to the analysis made for the processor occupancy, and considering that the bandwidth needed by the node associated to VO i to transmit the output data for task k is proportional to the D_k bits of data to transmit and to the frequency $f_{ik}(t)$ at which they are transmitted, the bandwidth occupancy is defined as

$$\Theta_i^{BW}(t) = \frac{\sum_k D_k \cdot f_{ik}(t)}{B_i^{tot}} \tag{6.6}$$

where B_i^{tot} is the available bandwidth for i.

6.3.2 Consensus-Based Resource Allocation Optimisation

The resource optimisation strategy proposed in this section relies on a consensus-based algorithm where VOs decide the amount of resources to allocate to a task, according to the constraints requested by the higher layers.

The Equation that describes the use of a resource by VO i can be generalized as

$$\Theta_i(t) = \sum_k \alpha_{ik} \cdot f_{ik}(t), \quad \text{with } \alpha_{ik}(t) = \begin{cases} E^c_{ik}/E^{res}_i(t) & if \Theta_i(t) = 1/\tau^{lftm}_i(t) \\ D_k/M^{res}_i(t) & if \Theta_i(t) = 1/\tau^{stor}_i(t) \\ t^{exec}_{ik} & if \Theta_i(t) = \Theta^{proc}_i(t) \\ D_k/B^{tot}_i & if \Theta_i(t) = \Theta^{BW}_i(t) \end{cases} \tag{6.7}$$

From the analysis carried out in the previous Section, it is evident that optimising the use of the resources belonging to the nodes involved in the system entails adjusting the use that VOs make of them, so that nodes are not overloaded. In other words, the frequency at which each node performs the tasks assigned to it needs to be adjusted so that the effort put by each node to contribute to the execution of tasks needed by the system is equally shared among all of them. This means that, taken two VOs i and j that received an activation request for task k, at time t_c when the algorithm converges, $\Theta_i(t_c) = \Theta_j(t_c)$. Therefore

$$\sum_k \alpha_{ik}(t_c) f_{ik}(t_c) = \sum_k \alpha_{jk}(t_c) f_{jk}(t_c) \tag{6.8}$$

Defining the total amount of resource usage contributions with the exception of task k as $\delta_{ik}(t) = \sum_{l \neq k} \alpha_{il}(t) \cdot f_{il}(t)$, it follows that

$$f_{jk}(t_c) = \frac{\alpha_{ik}(t_c)}{\alpha_{jk}(t_c)} \cdot f_{ik}(t_c) + \frac{\delta_{ik}(t_c) - \delta_{jk}(t_c)}{\alpha_{jk}(t_c)} \tag{6.9}$$

According to accuracy constraints provided by the higher layers, the collaborative completion of a task is required to be performed at a reference frequency $F^{ref}_k = \sum_j f_{jk}(t_c)$. Using Equation 6.9 in this identity, after some simple computations and multiplying and dividing by the number N_k of VOs involved in task k, we obtain

$$\alpha_{ik}(t_c) \cdot f_{ik}(t_c) = \frac{\bar{\varphi}_k}{\bar{\beta}_k(t_c)} + \frac{\bar{\gamma}_k(t_c)}{\bar{\beta}_k(t_c)} - \delta_{ik}(t_c) \tag{6.10}$$

with

$$\bar{\varphi}_k = \frac{F^{ref}_k}{N_k}$$

$$\bar{\beta}_k(t_c) = \frac{1}{N_k} \cdot \sum_j \frac{1}{\alpha_{jk}(t_c)}$$

$$\bar{\gamma}_k(t_c) = \frac{1}{N_k} \cdot \sum_j \frac{\delta_{jk}(t_c)}{\alpha_{jk}(t_c)}$$

It is easy to notice that they represent mean values evaluated over all the VOs that are able to perform task k. This fact, along with the consideration that nodes that are assigned to the same task are usually located close to each other, and thus they can communicate directly without passing through the cloud, leads to the conclusion that their value can be estimated in a distributed way using an average consensus algorithm.

It is supposed to have a system where nodes may not be connected during the whole convergence process. For this reason, in this section the consensus algorithm proposed in [73], which is robust against topology changes, is used. Since variations of α and δ are negligible over the time needed by the algorithm to converge (as it will be clarified in the experiments), in the following they are considered to be constant and their dependence from time is omitted. Nevertheless, if substantial variations of them are experienced, the algorithm needs to start again.

6.3.3 Resource Allocation Optimisation Algorithm

As soon as VO i receives an activation request for task k from the VO layer, it verifies if it is able to satisfy the minimum level of QoI required by the higher levels. If it is not, it sets f_{ik} to 0 and informs the VO layer about it. Otherwise, it initialises its local values $\varphi_{ik} = \varphi^0_{ik}$, $\beta_{ik} = \beta^0_{ik}$ and $\gamma_{ik} = \gamma^0_{ik}$. As far as φ_{ik} is concerned, only one VO receives the reference frequency F^{ref}_k from the VO layer, and sets φ^0_{ik} to it. The other VOs set it to 0. The initial local values are set as follows:

$$\varphi^0_{ik} = \begin{cases} F^{ref}_k & \text{if } F^{ref}_k \text{ is given} \\ 0 & \text{otherwise} \end{cases} \qquad \beta^0_{ik} = \frac{1}{\alpha_{ik}} \qquad \gamma^0_{ik} = \frac{\delta_{ik}}{\alpha_{ik}} \tag{6.11}$$

and starts the consensus with its neighbours. Whenever VO i receives an update from one of its neighbours j, it computes the following updates:

$$\varphi^+_{ik} = \varphi_{ik} - \lambda^\varphi_1 \sum_j (\varphi_{ik} - \varphi_{jk}) - \lambda^\varphi_2 \sum_j \operatorname{sgn}(\varphi_{ik} - \varphi_{jk}) \tag{6.12a}$$

$$\beta^+_{ik} = \beta_{ik} - \lambda^\beta_1 \sum_j (\beta_{ik} - \beta_{jk}) - \lambda^\beta_2 \sum_j \operatorname{sgn}(\beta_{ik} - \beta_{jk}) \tag{6.12b}$$

$$\gamma^+_{ik} = \gamma_{ik} - \lambda^\gamma_1 \sum_j (\gamma_{ik} - \gamma_{jk}) - \lambda^\gamma_2 \sum_j \operatorname{sgn}(\gamma_{ik} - \gamma_{jk}) \tag{6.12c}$$

$$\Theta^+_i = \frac{\varphi^+_{ik} + \gamma^+_{ik}}{\beta^+_{ik}} \qquad f^+_{ik} = \frac{1}{\alpha_{ik}} \cdot \left(\Theta^+_i - \delta_{ik}\right) \tag{6.12d}$$

where λ^φ_1, λ^β_1, λ^γ_1, λ^φ_2, λ^β_2, and λ^γ_2 are tuning parameters that affect the convergence time and steady-state accuracy [73], and that will be better explained in the following Subsection. If $f^+_{ik} > 0$ and if its value has changed after the update, the VO sends the updated value of φ^+_{ik}, β^+_{ik} and γ^+_{ik} to its neighbours. It may happen that $f^+_{ik} \leq 0$. In this case, the VO cannot

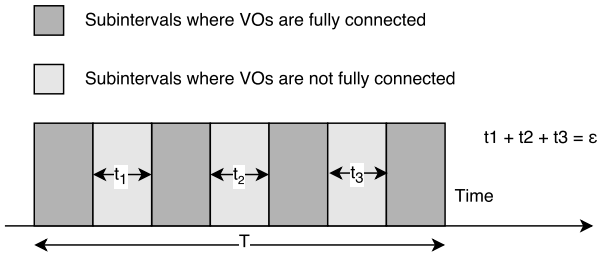

Figure 6.3: .

participate into executing task k. Therefore, it sets f_{ik} to 0 and informs its neighbours, which restart the consensus process. The algorithm can be considered converged when f_{ik} does not change after the updates.

6.3.4 Convergence Time and Steady-State Accuracy

The proposed consensus protocol represents a discrete-time application of the finite-time discontinuous average-based consensus algorithm discussed respectively in [74] for a network of connected continuous time integrators and in [75] for networks of perturbed, and possibly switching, spanned–tree topologies. It follows that, as long as the stability of the linear part of the problem in (6.12) is preserved, the convergent properties discussed in [74] and [75] are in force. Thus, from Lemma 3 of [76] and Theorem 1 of [75], it is straightforward to derive that

$$0 \leq \lambda_1^{\varphi}, \lambda_1^{\beta}, \lambda_1^{\gamma} \leq \frac{1}{\max_i |\mathcal{N}_i|} \tag{6.13}$$

where $|\mathcal{N}_i|$ denotes the number of neighbours of node i. Note that Equation (6.13) derives straightforward from considerations on discrete-time consensus and Perron matrices, which however go beyond the scope of this research. Further details can be found in [76][77]. If condition (6.13) holds, then the results of Theorem 1 of [75] are directly applicable for the characterization of the convergence properties of the discrete-time collective multi-agent dynamic in Equation (6.12).

Thus, following Assumption 1 of [74], let ε and T, with $\varepsilon \leq T$ be two positive constants, where T defines the length of a receding horizon time interval $I(t) = (t, t+T)$, and, ε is the total length of the subinterval $S(t) \leq I(t)$ given by the union of the subintervals during which the network is connected (see Figure 6.3 for a graphical explanation of the interval I(t) and S(t)), it results that, for the problem in (6.12), consensus will be reached in finite time if

$$\lambda_2^{\varphi}, \lambda_2^{\beta}, \lambda_2^{\gamma} \geq 2 \cdot \frac{T}{\varepsilon} + \mu^2 \tag{6.14}$$

with $\mu \neq 0$. If condition (6.14) holds, the convergence is reached with accuracy Γ after, at most, a transient time t_r that is proportional to the maximum deviation of the agents' states at the start-up (i.e., when t=0) of the algorithms

$$t_r \leq \left(\frac{T}{\varepsilon\mu^2}\right) \cdot \max_{i,j} |x_i^0 - x_j^0|$$
$$\Gamma = 2 \cdot (T - \varepsilon) + \xi \tag{6.15}$$

where $\xi > 0$ is an arbitrary infinitesimally small parameter, and x_i^0, x_j^0 are the initial values for VOs i and j of the generic consensus variables, that in our case are those specified by Equation (6.11).

Supposing that $T = \varepsilon$, i.e. the VOs are always connected during the consensus process:

$$t_r \leq \left(\frac{1}{\mu^2}\right) \cdot \max_{i,j} |x_i^0 - x_j^0|$$
$$\Gamma = \xi \tag{6.16}$$

6.4 The Proposed IoT System

In this section the whole IoT resource allocation system proposed in this chapter is described. Algorithm 1 provides the pseudo-code for the whole process. As soon as the service level receives a request for a task, it translates it into computer language, generating the query $\mathcal{Q}_k$, which is sent to the VO level. Based on $\mathcal{Q}_k$, the VO level finds the VO Information Model that best fits the characteristics required by $\mathcal{Q}_k$. The VO level than starts a search for the set $\mathcal{S}_k$ of VO instances that correspond to the required VO Information Model, i.e. the set of VOs that can respond to the query. Then, the VO level selects one of the VOs to whom forwarding the request, i.e. the candidate VO VO_0. Since the candidate VO has to perform some additional operations with respect to the other VOs, the VO level tries to choose the one that is likely to have more resources. For this reason, if in $\mathcal{S}_k$ there is at least one VO that is located in the cloud, the candidate VO is chosen randomly among them; otherwise, if there is at least one VO that is located in an intermediate gateway, the candidate VO is chosen to be the one located in the closest gateway; if all the VOs are located remotely, the candidate VO is chosen to be the closest one. The VO level sends to the candidate VO a message $\mathcal{M}_k$, including the reference frequency, the set of VOs, the resource to be optimally allocated and the time interval T_k during which the task has to be continuously performed: $\mathcal{M}_k = \{F_k^{ref}, \mathcal{S}_k, resource, T_k\}$.

After receiving the request from the VO level, the candidate VO has to choose whether or not the consensus algorithm is convenient to be started. Indeed, since the consensus process requires a certain amount of resources, before proceeding with it, it is important to evaluate if it is convenient to the system, i.e. if the amount of resources saved thanks to consensus is higher than the amount of resources needed to reach a consensus. It is trivial

[1] Note that it is not necessary that nodes in $\mathcal{S}_k$ are directly connected: it is sufficient that their VOs are connected (either physically or logically) by a limited number of hops

Algorithm 1 Resource allocation process: pseudo-code

The service level receives a request for task k
The service level translates the task request into query $\mathcal{Q}_k$
The service level sends $\mathcal{Q}_k$ to the VO level
The VO level finds the appropriate VO Information Model to respond to $\mathcal{Q}_k$
The VO level finds the set $\mathcal{S}_k$ of VOs corresponding to the required VO Information Model
if at least one VO $\in \mathcal{S}_k$ is in the cloud **then**
    Set it as VO_0
else if at least one VO $\in \mathcal{S}_k$ is in an intermediate gateway **then**
    Set the VO in the closest intermediate gateway as VO_0
else
    Set the closest VO as VO_0
end if
The VO level sends message $\mathcal{M}_k$ to VO_0
VO_0 evaluates Equation (6.18)
if Equation (6.18) is **false then**
    VO_0 assigns $f_{ik} = F_k^{ref}/|\mathcal{S}_k|, \forall i \in \mathcal{S}_k$
else
    VO_0 sends the activation request and initialization message for task k to the nodes in $\mathcal{S}_k$[1]
    for each $i \in \mathcal{S}_k$ **do**
        if An initialization message is received **then**
            Initialize φ_{ik}, β_{ik} and γ_{ik} values according to Equation (6.11)
        end if
        if An update message is received **then**
            Compute φ_{ik}^+, β_{ik}^+, γ_{ik}^+ and f_{ik}^+ values according to Equation (6.12)
            if f_{ik}^+>0 **then**
                if $f_{ik}^+ \neq f_{ik}$ **then**
                    i sends φ_{ik}^+, β_{ik}^+ and γ_{ik}^+ values to all $j \in \mathcal{N}_i$
                end if
            else
                i sets $f_{ik} = 0$ and sends an initialization message to all $j \in \mathcal{N}_i$
            end if
        end if
    end for
end if

to demonstrate that, if $T_k = \infty$, i.e. task k's duration is not specified by the request, the consensus execution is always convenient. If T_k is limited, the candidate VO has to evaluate how much the consensus algorithm costs in terms of resources, with respect to the requested task.

We call α_i^{cons} the amount of resource consumed to perform a single step of the consensus algorithm, i.e. the value of α_{ik} computed according to Equation (6.7) not considering a single execution of task k, but a single execution of a step of the consensus algorithm. Let $\overline{N}^{step}$ be the average number of steps required by consensus to converge. Performing consensus is convenient if the following condition is satisfied:

$$\alpha_i^{cons} \cdot \overline{N}^{step} \ll \alpha_{ik} \cdot f_{ik} \cdot T_k \tag{6.17}$$

Approximating f_{ik} with $F_k^{ref}/|\mathcal{S}_k|$, where $|\mathcal{S}_k|$ is the number of VOs in $\mathcal{S}_k$, it is possible to approximate the condition above as follows

$$\alpha_i^{cons} < \alpha_{ik} \cdot \frac{F_k^{ref}}{\mathcal{S}_k} \cdot \frac{T_k}{\Lambda \cdot \overline{N}^{step}} \tag{6.18}$$

where Λ is an arbitrarily high design parameter. Considering, for example, 10 VOs, $\alpha_i^{cons} = \alpha_{ik}$, $F_k^{ref} = 0.1$ Hz, $\Lambda = 20$ and $\overline{N}^{step} = 7$ (which, as shown in Section 6.5, is a reasonable value), the condition in (6.18) is met for $T_k > 3.8$ hour. If the amount of saved resources is not expected to be sufficient, the process is not started at all, and frequencies are assigned to the nodes in $\mathcal{S}_k$ according to a static assignment, e.g. they are set to $F_k^{ref}/|\mathcal{S}_k|$ for each node. Otherwise, the algorithm described in Section 6.3.3 is started.

6.5 Experiments

The proposed algorithm has been implemented to run in the Arduino Mega 2560 [78] device, whose microcontroller is an ATmega 2560. The local network was created through XBee S1 802.15.4 modules, by Digi International [79]. These modules use the IEEE 802.15.4 networking protocol for fast point-to-multipoint or peer-to-peer networking. The XBee modules are ideal for low-power and low-cost applications. The XBee modules have been connected to Arduino via serial port, using Xbee USB serial adapters by DF Robot [80]. Tests were performed considering up to 10 real devices participating in the optimisation process for the allocation of up to 10 tasks.

Tasks are supposed to have different complexities are assigned to nodes one at a time. Nodes have a residual energy ranging from 2 to 3 kJ. It has also been supposed to know the energy consumption value associated to each task at each node. According to it, energy consumption values for a single execution of each task are assigned randomly to the nodes in the ranges defined in Table 6.1. As a term for comparison, typical energy consumption values to transmit data using XBee modules are ~ 0.3 mJ/byte [79][81], while approximately

Task ID	Task 1	Task 2	Task 3	Task 4	Task 5
E^c_{ik} **value [mJ]**	6.82 ÷ 12.27	7.50 ÷ 13.49	9.70 ÷ 17.46		5.11 ÷ 9.20
Task ID	Task 6	Task 7	Task 8	Task 9	Task 10
E^c_{ik} **value [mJ]**	6.51 ÷ 11.71	8.49 ÷ 15.28	9.13 ÷ 16.43	5.68 ÷ 10.23	9.07 ÷ 16.33

Table 6.1: Energy consumption values per task's single execution

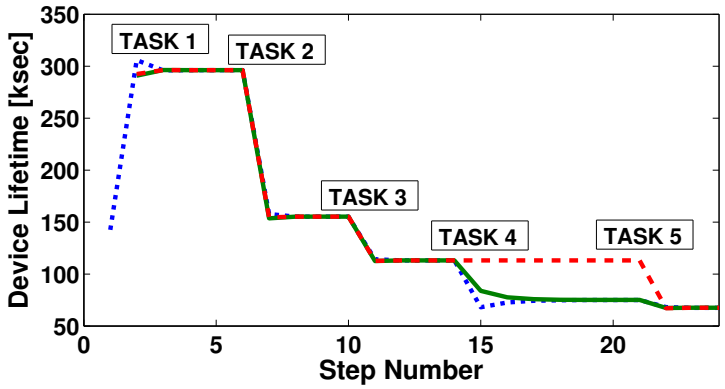

Figure 6.4: Example plot for algorithm convergence

7 μJ are needed, on a typical board, to execute a simple application such as the average of five numbers [82].

Figure 6.4 shows in an explanatory example how 3 devices reach consensus for 5 different tasks. Each line style is associated to a different device. Whenever a new task is activated, the devices that can perform that task initiate the consensus process. The initialisation instants correspond to the peaks in the Figures, and are marked by the respective label. It is possible to see how, for each task, the convergence is reached in just a few steps. On average, the algorithm takes only less than 7 steps per task to converge. For each task activation, the lifetime values of the 3 devices converge, as the frequency of execution is distributed in an optimised manner to reach the reference frequency. In the example, task 4 can be performed by only 2 devices out of 3. Thus, only 2 devices take charge of the workload related to task 4, and their lifetime value converges toward a lower value than that of the other device. After the algorithm has run for task 4, it could be run again for the tasks whose frequency has already been assigned, in order for the devices to equally redistribute the workload and reach the same lifetime value again.

Nevertheless, we believe that the benefit introduced by this process would not be enough, especially considering that the following tasks will have the same result of making the devices converge to the same lifetime. Indeed, in the example, once the fifth task is activated, frequencies are divided one more time and devices reach the same lifetime value again.

To evaluate the performance of the algorithm, the three different approaches have been compared:

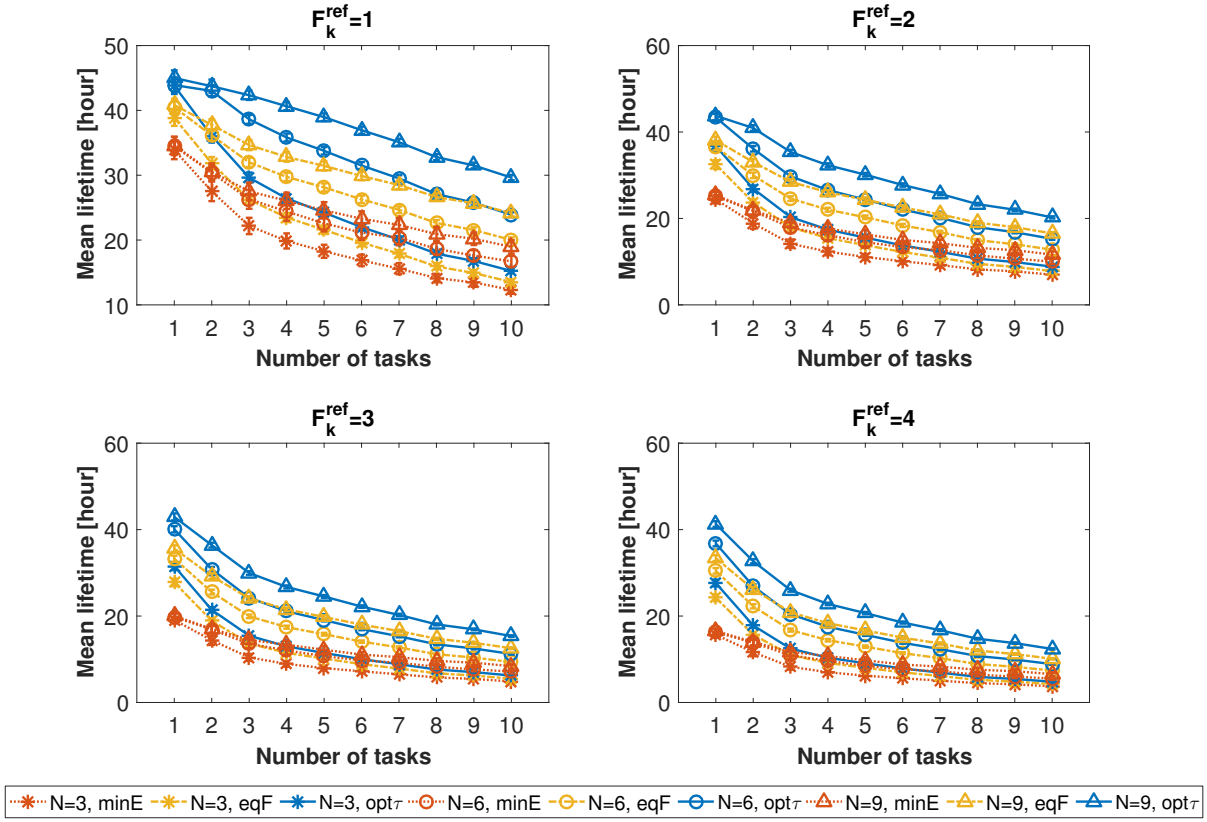

Figure 6.5: Average values of network lifetime when the number of tasks increases, for a number of available nodes equal to 3 (star marker), 6 (circle marker) and 9 (triangle marker). Results are shown for different reference frequency values for each task

- network lifetime achieved using the proposed algorithm (indicated with label *opt*τ);
- network lifetime when each task is entirely assigned to the node with the lowest energy consumption value related to that task (label *minE*);
- network lifetime when the task's reference frequency F_k^{ref} equally divided by the number of devices available to run it (label *eqF*).

Figures 6.5 and 6.6 show the average network lifetime and related confidence interval, using the three different approaches, for different numbers of assigned tasks and nodes (indicated respectively with labels K and N). The graphs show that the optimal resource allocation algorithm always outperforms the other approaches, especially with respect to *minE*. The gap is particularly evident when the amount of available resources is higher than that of required resources, i.e. when the number of nodes is high, or when the number of assigned tasks and reference frequency are low. This is motivated by the fact that, with the non-optimized solutions, if the number of tasks is lower than the number of involved nodes, the probability to have an unfair distribution of energy among nodes is higher with respect to that of a high number of tasks. Therefore, the higher the amount of available resources,

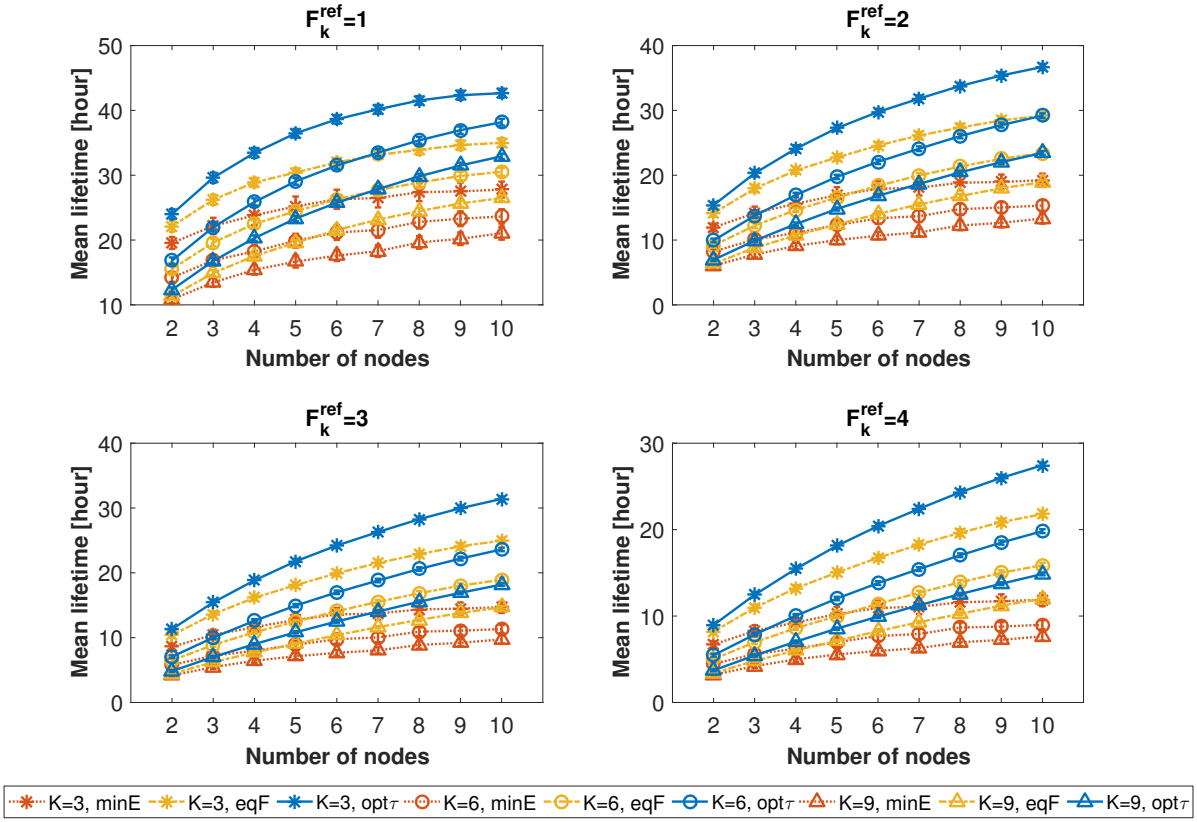

Figure 6.6: Average values of network lifetime when the number of nodes increases, for a number of assigned tasks equal to 3 (star marker), 6 (circle marker) and 9 (triangle marker). Results are shown for different reference frequency values for each task

the better the behaviour of the resource allocation algorithm. The lifetime improvement of the optimal resource allocation algorithm goes from 12% to 60.3% for the *minE* approach, and from 6.5% to 20.8% for the *eqF* approach.

The behaviour of the algorithm was also evaluated from the time performance point of view. The convergence times measured during the testing phase and related confidence interval are shown in Figure 6.7 as a function of the number of tasks to be assigned. It goes from 440 msec when only 2 tasks are assigned to 2.14 sec when 10 tasks are assigned, with an average convergence time of 214 msec per task.

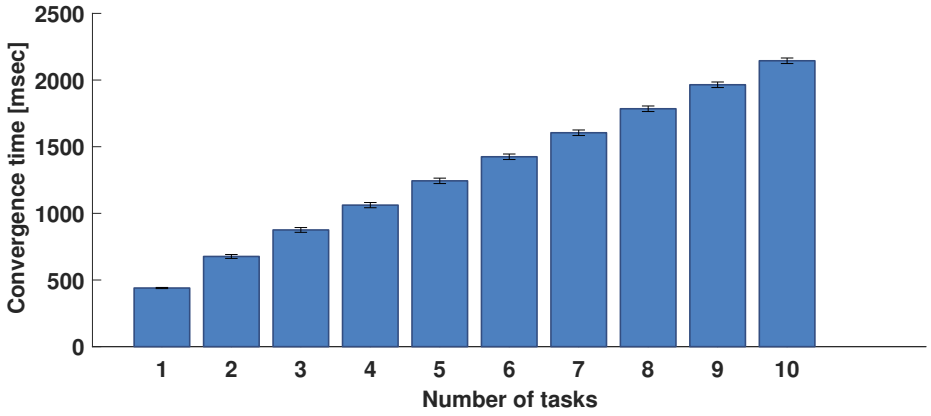

Figure 6.7: Average values of convergence time

Chapter 7

Conclusion and future works

This thesis focused on ICT technologies for urban mobility, with particular attention to the study, design and the development of applications for dynamic carpooling services in smart cities.

In the chapter 3 it has been provided an overview of the design and the implementation of the CLACSOON project: the service implements a carpooling application that automatizes the arrangement of the shared ride, automatically notifying the presence of suitable travel companions and suggesting the pick-up and drop-off points and the meeting times.

A first beta version of the CLACSOON's Android application has been deployed in the 2014. The mobile application was released to a restricted group of users, the betatesters, and the results we obtained allowed us to improve the application from the user-experience point of view, with an user-centered design lifecycle. Nowadays the service has been designed and implemented and it is publicly available for the Android and the iOS platforms.

In the chapter 4, we have proposed a novel implementation of a route matching algorithm. This algorithm contemplates the partial ridesharing mode: the implementation avoids the driver to take a detour when possible, resulting in an higher value for the total system-wide CO_2 savings. Moreover, the matching algorithm includes a method for modeling off-line the position of the driver's vehicle in an urban context, enabling the possibility for shared rides to be agreed after the starting of the driver's trip.

An emulation system (chapter 5) has been implemented to analyze the performances of the matching algorithm and to investigate the Quality of Experience provided to the users by the service, respect to the characteristics of the population. Experiments have been conducted considering a real urban area, to emulate the mobility patterns in urban conditions taking into account the real paths between any departure and destination (e.g. pedestrian zone, one-way roads, limited traffic zones).

The results allowed to identify the relationships between all the KPIs (the passenger success rate, the driver success rate, the passenger waiting time and the total system-wide CO2

saved) and the characteristics of the population (spatial distribution of users, percentage of drivers and passengers, temporal distribution of requests and offers).

The achieved results show that introducing the aforementioned features in the CLAC-SOON matching algorithm leads to a substantial performance improvement in terms of all the KPIs, leading to an increase (+21% for a population of $40 users/km^2$) of the CO_2 saved . Another relevant result is that, depending on the ratio between drivers and riders, the service achieves the higher level of performances when the number of drivers is close or equal to the number of riders. The trend of the KPIs suggest that, depending on the characteristics of the population, different campaigns or strategies could be followed to achieve the desired level of performances. The results presented in this thesis could be considered when designing a strategy to build a successful carpooling service in a smart city.

Starting from the vision that *carpoolers'* cars can be seen as a formidable sensor platform, in the chapter 6 the problem of task allocation in a typical IoT scenario has been analysed. The use of Virtual Objects has been proposed to control and manage the heterogeneous resource-constrained objects that characterize the IoT, and a consensus-based algorithm has been proposed to distribute the workload between these objects in a fair way. Experiments have been conducted with real devices, and the results shown that the use of the consensus approach lead to an improvement of devices' lifetime of more than 20% using up to 10 devices, with respect to a uniform distribution of tasks. The results from this work, in the field of infomobility and support of the urban mobility, could be run in real devices on the vehicles, to perform tasks such as the sensing of traffic conditions, noise or the quality of the roads.

The study done so far has led to the acquisition of the expertise required to widen the research about the applications for dynamic carpooling services and, more in general, about the ICT technologies for urban mobility. In this respect, future works could be focused on the use of even more real scenarios to study the performance of the proposed carpooling service: for example, the generation of ride offers and ride requests according to the real data of the urban mobility in the city of Cagliari. The proposed service could also be tested in real situation involving a community of volunteers to validate the synthetic results. Future works could also be focused on the development of new algorithms to manage the resources of the IoT objects, on the basis of the quality and the trustworthiness of the acquired data, in order to achieve better quality of information.

Bibliography

[1] Hafedh Chourabi et al. “Understanding smart cities: An integrative framework”. In: *System Science (HICSS), 2012 45th Hawaii International Conference on*. IEEE. 2012, pp. 2289–2297.

[2] Wikipedia. “Sustainable transport”. In: *Wikipedia* (Aug. 2016). URL: https://en.wikipedia.org/wiki/Sustainable_transport.

[3] EPA U. “US Transportation Sector Greenhouse Gas Emissions: 1990–2011”. In: *Office of Transportation and Air Quality EPA-420-F-13-033a* (2013).

[4] Wikipedia. “Carpooling”. In: *Wikipedia* (Aug. 2016). URL: https://en.wikipedia.org/wiki/Carpool.

[5] N Agatz et al. *The Value of Optimization in Dynamic Ride-Sharing: a Simulation Study in Metro Atlanta, Research paper, Erasmus Research Institute of Management (ERIM), Report No.* Tech. rep. ERS-2010-034-LIS. 2010, Retrieved from: http://hdl. handle. net/1765/20456.

[6] Stiglic Mitja and Agatz. “The benefits of meeting points in ride-sharing systems”. In: *Transportation Research Part B: Methodological* 82 (2015), pp. 36–53.

[7] Matteo Mallus et al. “Carpooling in Urban Areas: A Real-Time Service Case-Study”. In: *Internet of Things. IoT Infrastructures: Second International Summit, IoT 360° 2015, Rome, Italy, October 27-29, 2015. Revised Selected Papers, Part I*. Cham: Springer International Publishing, 2016, pp. 157–166. ISBN: 978-3-319-47063-4. DOI: 10.1007/978-3-319-47063-4_14. URL: http://dx.doi.org/10.1007/978-3-319-47063-4_14.

[8] Matteo Mallus et al. “A persuasive real-time carpooling service in a smart city: a case-study to measure the advantages in urban area”. Manuscript accepted for publication. 2016.

[9] Matteo Mallus et al. “Dynamic Carpooling in Urban Areas: Design and Experimentation with a Multi-Objective Route Matching Algorithm”. In: *Sustainability* 9.2 (2017), p. 254.

[10] Virginia Pilloni, Luigi Atzori, and Matteo Mallus. “Dynamic Involvement of Real World Objects in the IoT: A Consensus-Based Cooperation Approach”. In: *Sensors* 17.3 (2017), p. 484.

[11] EU Commission et al. “Green paper, towards a new culture for urban mobility”. In: *European Union, Brussels* (2007).

[12] Mihyeon Jeon, Christy, Amekudzi, et al. “Addressing sustainability in transportation systems: definitions, indicators, and metrics”. In: *Journal of infrastructure systems* 11.1 (2005), pp. 31–50.

[13] Masabumi Furuhata et al. “Ridesharing: The state-of-the-art and future directions”. In: *Transportation Research Part B: Methodological* 57 (2013), pp. 28–46.

[14] Niels Agatz, Alan L. Ererab, et al. “Dynamic Ride-Sharing: a Simulation Study in Metro Atlanta”. In: *19th International Symposium on Transportation and Traffic Theory* (2011), pp. 532–550.

[15] Blablacar. *Blablacar ridesharing*. [Online; accessed 30-July-2015]. URL: https://www.blablacar.it/.

[16] Wikipedia. *Blablacar*. [Online; accessed 01-Sept-2016]. URL: https://en.wikipedia.org/wiki/BlaBlaCar.

[17] B Akshay et al. “Carpool „up-Real Time Carpooling using GPS”. In: *Proceedings of National Conference on New Horizons IT (NCNHIT)*. 2013, pp. 126–128.

[18] Andrew Amey, John Attanucci, and Rabi Mishalani. “Real-time ridesharing: opportunities and challenges in using mobile phone technology to improve rideshare services”. In: *Transportation Research Record: Journal of the Transportation Research Board* 2217 (2011), pp. 103–110.

[19] Wikipedia. *Lyft — Wikipedia, The Free Encyclopedia*. [Online; accessed 13-December-2016]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Lyft&oldid=754506168.

[20] Lyft.com. *Lyft - safety*. [Online; accessed 13-December-2016]. 2016. URL: https://www.lyft.com/safety.

[21] Niels Agatz et al. “Sustainable passenger transportation: Dynamic ride-sharing”. In: (2010).

[22] Niels Agatz et al. “Optimization for dynamic ride-sharing: A review”. In: *European Journal of Operational Research* 223.2 (2012), pp. 295–303.

[23] DynamicRidesharing.org. *Dynamic Ridesharing - Critical mass*. [Online; accessed 13-December-2016]. 2016. URL: http://dynamicridesharing.org/critical_mass.php.

[24] Xin Xing et al. “Smize: a spontaneous ride-sharing system for individual urban transit”. In: (2009), pp. 165–176.

[25] Gérald Arnould et al. “A transport based clearing system for dynamic carpooling business services”. In: *ITS Telecommunications (ITST), 2011 11th International Conference on*. IEEE. 2011, pp. 527–533.

[26] Niels AH Agatz et al. “Dynamic ride-sharing: A simulation study in metro Atlanta”. In: *Transportation Research Part B: Methodological* 45.9 (2011), pp. 1450–1464.

[27] Wesam Mohamed Herbawi and Michael Weber. “A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows”. In: *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM. 2012, pp. 385–392.

[28] A Di Febbraro, E Gattorna, and N Sacco. “Optimization of dynamic ridesharing systems”. In: *Transportation Research Record: Journal of the Transportation Research Board* 2359 (2013), pp. 44–50.

[29] Nicolai Mallig and Peter Vortisch. “Modeling Car Passenger Trips in mobiTopp”. In: *Procedia Computer Science* 52 (2015), pp. 938–943.

[30] Sisinnio Concas and Philip Winters. “Impact of carpooling on trip-chaining behavior and emission reductions”. In: *Transportation Research Record: Journal of the Transportation Research Board* (2007).

[31] AASHTO. “Commuting in America 2013: The National Report on Commuting Patterns and Trends”. In: *American Association of State Highway and Transportation Officials (AASHTO)* (2014).

[32] Erik Ferguson. “The rise and fall of the American carpool: 1970–1990”. In: *Transportation* 24.4 (1997), pp. 349–376. ISSN: 1572-9435. DOI: 10.1023/A:1004928012320. URL: http://dx.doi.org/10.1023/A:1004928012320.

[33] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. “Mobile cloud computing: A survey”. In: *Future Generation Computer Systems* 29.1 (2013), pp. 84–106.

[34] Guanling Chen, David Kotz, et al. *A survey of context-aware mobile computing research*. Tech. rep. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[35] Ivana Podnar, Manfred Hauswirth, and Mehdi Jazayeri. “Mobile push: Delivering content to mobile users”. In: *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*. IEEE. 2002, pp. 563–568.

[36] Deyun Gao et al. “Wireless vehicular sensor and ad hoc networks 2015”. In: *International Journal of Distributed Sensor Networks* 2015 (2015).

[37] Xiaolan Tang et al. “Integrated extensible simulation platform for vehicular sensor networks in smart cities”. In: *International Journal of Distributed Sensor Networks* 2012 (2012).

[38] Sherin Abdelhamid, Hossam S Hassanein, and Glen Takahara. “Vehicle as a mobile sensor”. In: *Procedia Computer Science* 34 (2014), pp. 286–295.

[39] Mario Gerla et al. “Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds”. In: *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE. 2014, pp. 241–246.

[40] Google Developers. *Google Directions API*. [Online; accessed 30-November-2016]. 2016. URL: https://developers.google.com/maps/documentation/directions/.

[41] Microsoft. *Bing Maps for Enterprise*. [Online; accessed 30-November-2016]. 2016. URL: https://www.microsoft.com/maps/choose-your-bing-maps-API.aspx.

[42] Dennis Luxen and Christian Vetter. “Real-time routing with OpenStreetMap data”. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS ’11. Chicago, Illinois: ACM, 2011, pp. 513–516. ISBN: 978-1-4503-1031-4. DOI: 10.1145/2093973.2094062. URL: http://doi.acm.org/10.1145/2093973.2094062.

[43] Xia Jizhe et al. “A New Model for a Carpool Matching Service”. In: *PLoS ONE 10(6): e0129257. doi:10.1371/journal. pone.0129257* (6) (2015).

[44] Wikipedia. *Cagliari — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-July-2015]. 2015. URL: https://en.wikipedia.org/w/index.php?title=Cagliari&oldid=671950070.

[45] Feng An and Amanda Sauer. “Comparison of passenger vehicle fuel economy and greenhouse gas emission standards around the world”. In: *Pew Center on Global Climate Change* 25 (2004).

[46] Luigi Atzori, Antonio Iera, and Giacomo Morabito. “The internet of things: A survey”. In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[47] Governing. *Santander: The Smartest Smart City*. [Online; accessed 13-December-2016]. 2016. URL: http://www.governing.com/topics/urban/gov-santander-spain-smart-city.html.

[48] Santander. *Smart Santander*. [Online; accessed 13-December-2016]. 2016. URL: http://www.smartsantander.eu/index.php/testbeds/item/132-santander-summary.

[49] Jeffrey A Burke et al. “Participatory sensing”. In: *Center for Embedded Network Sensing* (2006).

[50] Xiao-Feng Xie and Zun-Jing Wang. "An empirical study of combining participatory and physical sensing to better understand and improve urban mobility networks". In: *Transportation Research Board 94th Annual Meeting*. 15-3238. 2015.

[51] Amirhossein Ghanbari, Óscar Álvarez, Jan Markendahl, et al. "Internet of Things: re-definition of Business Models for the next generation of Telecom services". In: *26th European Regional ITS Conference, Madrid 2015*. 127142. International Telecommunications Society (ITS). 2015.

[52] Geng Wu et al. "M2M: From mobile to embedded internet". In: *Communications Magazine, IEEE* 49.4 (2011), pp. 36–43.

[53] Michele Nitti et al. "On adding the social dimension to the internet of vehicles: Friendship and middleware". In: *Communications and Networking (BlackSeaCom), 2014 IEEE International Black Sea Conference on*. IEEE. 2014, pp. 134–138.

[54] Vicente Milanés et al. "An intelligent V2I-based traffic management system". In: *Intelligent Transportation Systems, IEEE Transactions on* 13.1 (2012), pp. 49–58.

[55] Jaume Barceló et al. "Microscopic traffic simulation: A tool for the design, analysis and evaluation of intelligent transport systems". In: *Journal of Intelligent and Robotic Systems* 41.2-3 (2005), pp. 173–203.

[56] Urban ITS Expert Group. *Intelligent Transport Systems for Urban Areas*. [Online; accessed 30-July-2015]. 2015. URL: http://ec.europa.eu/transport/themes/its/road/action_plan/its_for_urban_areas_en.

[57] Jayavardhana Gubbi et al. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future Generation Computer Systems* 29.7 (2013), pp. 1645–1660.

[58] Michele Nitti et al. "The Virtual Object as a Major Element of the Internet of Things: a Survey". In: *IEEE Communications Surveys & Tutorials* 18.2 (2015), pp. 1228–1240.

[59] J Pacual-Espada et al. "Virtual objects on the Internet of things". In: *International Journal of Interactive Multimedia and Artificial Intelligence* 1.4 (2011).

[60] Vivek Kumar Sehgal, Anubhav Patrick, and Lucky Rajpoot. "A comparative study of cyber physical cloud, cloud of sensors and internet of things: Their ideology, similarities and differences". In: *Advance Computing Conference (IACC), 2014 IEEE International*. IEEE. 2014, pp. 708–716.

[61] I. Farris et al. "Federated edge-assisted mobile clouds for service provisioning in heterogeneous IoT environments". In: 2015, pp. 591–596.

[62] Virginia Pilloni et al. "TAN: a distributed algorithm for dynamic task assignment in WSNs". In: *Sensors Journal, IEEE* 14.4 (2014), pp. 1266–1279.

[63] Yan Shen and Hui Ju. “Energy-Efficient Task Assignment Based on Entropy Theory and Particle Swarm Optimization Algorithm for Wireless Sensor Networks”. In: *Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on*. IEEE. 2011, pp. 120 –123.

[64] Dominique Guinard et al. “From the internet of things to the web of things: Resource-oriented architecture and best practices”. In: *Architecting the Internet of Things*. Springer, 2011, pp. 97–129.

[65] Bilhanan Silverajan and Jarmo Harju. “Developing network software and communications protocols towards the internet of things”. In: *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*. ACM. 2009, p. 9.

[66] Giuseppe Colistra, Virginia Pilloni, and Luigi Atzori. “The problem of task allocation in the Internet of Things and the consensus-based approach”. In: *Computer Networks* 73 (2014), pp. 98–111.

[67] Panagiotis Vlacheas et al. “Enabling smart cities through a cognitive management framework for the internet of things”. In: *Communications Magazine, IEEE* 51.6 (2013), pp. 102–111.

[68] Chatschik Bisdikian, Lance M Kaplan, and Mani B Srivastava. “On the quality and value of information in sensor networks”. In: *ACM Transactions on Sensor Networks (TOSN)* 9.4 (2013), p. 48.

[69] Wei Wang et al. “A comprehensive ontology for knowledge representation in the internet of things”. In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*. IEEE. 2012, pp. 1793–1798.

[70] iCore Project. *iCore: Empowering IoT through Cognitive Tecnologies*. 2015. URL: http://www.iot-icore.eu/about-icore.

[71] Flavio Bonomi et al. “Fog computing and its role in the internet of things”. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, pp. 13–16.

[72] YoungSang Yun et al. “Distributed algorithm for lifetime maximization in a delay-tolerant wireless sensor network with a mobile sink”. In: *Mobile Computing, IEEE Transactions on* 12.10 (2013), pp. 1920–1930.

[73] Alessandro Pilloni et al. “Recent advances in sliding-mode based consensus strategies”. In: *Variable Structure Systems (VSS), 2014 13th International Workshop on*. IEEE. 2014, pp. 1–6.

[74] Mauro Franceschelli et al. “Finite-time consensus with disturbance attenuation for directed switching network topologies by discontinuous local interactions”. In: *52nd IEEE Conference on Decision and Control*. IEEE. 2013, pp. 2611–2616.

[75] Fangcui Jiang and Long Wang. “Finite-time weighted average consensus with respect to a monotonic function and its application”. In: *Systems & Control Letters* 60.9 (2011), pp. 718–725.

[76] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. “Consensus and cooperation in networked multi-agent systems”. In: *Proceedings of the IEEE* 95.1 (2007), pp. 215–233.

[77] Chris Godsil and Gordon F Royle. *Algebraic graph theory*. Vol. 207. Springer Science & Business Media, 2013.

[78] Arduino. *Arduino Mega 2560*. 2015. URL: https://www.arduino.cc/en/Main/ArduinoBoardMega2560.

[79] Digi International® Inc. *Xbee S1*. 2015. URL: http://www.digi.com/products/xbee-rf-solutions/modules/xbee-series1-module.

[80] DF Robot. *Xbee S1*. 2015. URL: http://www.dfrobot.com/index.php?route=product/product&product_id=72.

[81] Ming Liu et al. “An energy-aware routing protocol in wireless sensor networks”. In: *Sensors* 9.1 (2009), pp. 445–462.

[82] Antônio Dâmaso et al. “Evaluating the power consumption of wireless sensor network applications using models”. In: *Sensors* 13.3 (2013), pp. 3473–3500.

List of Publications Related to the Thesis

Published papers

Journal papers

- Mallus M, Colistra G, Atzori L, Murroni M, Pilloni V. “Dynamic Carpooling in Urban Areas: Design and Experimentation with a Multi-Objective Route Matching Algorithm”. Sustainability. 2017; 9(2):254 (Relation to Chapter 4 and 5)

- Pilloni V, Atzori L, Mallus M. “Dynamic Involvement of Real World Objects in the IoT: A Consensus-Based Cooperation Approach”. Sensors. 2017; 17(3):484 (Relation to Chapter 6)

Conference papers

- Matteo Mallus et al. “Carpooling in Urban Areas: A Real-Time Service Case-Study”. In: Internet of Things. IoT Infrastructures: Second International Summit, IoT 360° 2015, Rome, Italy, October 27-29, 2015. Revised Selected Papers, Part I. Cham: Springer International Publishing, 2016, pp. 157–166. ISBN: 978-3-319-47063-4 (Relation to Chapter 3)

- Matteo Mallus et al. “A persuasive real-time carpooling service in a smart city: a case-study to measure the advantages in urban area”. In: ICIN 2017 - Innovations in Clouds, Internet and Networks 7-9 March 2017 - Manuscript accepted for publication. 2016 (Relation to Chapter 3 and 5)