Free University of Bolzano

Faculty of Computer Science

Thesis

Dycapo: On the creation of an open-source Server and a Protocol for Dynamic Carpooling

Daniel Graziotin

Submitted in partial fulfillment of the requirements for the degree of Bachelor in Applied Computer Science at the Free University of Bolzano/Bozen

Thesis Advisor: Paolo Massa, Ph.D.

October 2010

Abstract

Using a private car is a transportation system very common in industrialized countries. However, it causes different problems such as overuse of oil, traffic jams causing earth pollution, health problems and an inefficient use of personal time.

One possible solution to these problems is carpooling, i.e. sharing a trip on a private car of a driver with one or more passengers. Carpooling would reduce the number of cars on streets hence providing worldwide environmental, economical and social benefits. The matching of drivers and passengers can be facilitated by information and communication technologies. Typically, a driver inserts on a web-site the availability of empty seats on his/her car for a planned trip and potential passengers can search for trips and contact the drivers. This process is slow and can be appropriate for long trips planned days in advance. We call this static carpooling and we note it is not used frequently by people even if there are already many web-sites offering this service and in fact the only real open challenge is widespread adoption.

Dynamic carpooling, on the other hand, takes advantage of the recent and increasing adoption of Internet-connected geo-aware mobile devices for enabling impromptu trip opportunities. Passengers request trips directly on the street and can find a suitable ride in just few minutes. Currently there are no dynamic carpooling systems widely used. While adoption is still a key issue, dynamic carpooling offers many open basic challenges related to the implementation of the technological infrastructure.

This dissertation describes Dycapo, an open-source project for enabling dynamic carpooling services. After a review of the state of the art and a comparative analysis of dynamic carpooling issues, the two main components of the project are described, namely the protocol and the server. Dycapo Protocol is an open REST-based protocol for sharing trip information among dynamic carpooling clients and servers, taking inspiration from OpenTrip, a previously proposed data exchange format. Dycapo Server is a prototype providing web services for dynamic carpooling functionalities, implementing Dycapo Protocol. Our aim with the release of an open protocol and open source code is to provide a missing standard and platform that providers of dynamic carpooling services can adopt and extend.

Riassunto

Il veicolo privato è un sistema di trasporto molto diffuso nei Paesi industrializzati. Tuttavia questo utilizzo provoca diversi problemi, come ad esempio un uso eccessivo del petrolio e dei suoi derivati, un aumento della densità veicolare con conseguente incremento dell'inquinamento, problemi di salute e un uso inefficiente del proprio tempo.

Una possibile soluzione a questi problemi è il carpooling, ovvero la condivisione di un viaggio su una macchina privata tra un conducente ed uno o più passeggeri. Il carpooling ridurrebbe il numero di auto sulle strade, fornendo quindi benefici ambientali, economici e sociali a livello mondiale. L'incontro tra conducenti e passeggeri può essere facilitato dalle tecnologie di informazione e comunicazione. L'autista, al momento di pianificare un viaggio, inserisce su un sito web la disponibilità di posti vuoti della sua auto per quel tragitto, i potenziali passeggeri, in caso di disponibilità di posti, possono così contattare l'autista e condividere il viaggio. Questo processo, essendo lento, può essere adeguato per lunghi viaggi, pianificati con giorni di anticipo. Il servizio sopracitato viene definito carpooling statico e nonostante numerosi siti offrano già questo servizio, il fenomeno non è largamente diffuso. Di fatto, l'unica vera sfida aperta è un'adozione massificata di questi servizi.

Il carpooling dinamico si avvale della recente adozione di dispositivi mobili, collegati ad Internet e con funzionalità di geo-localizzazione, offrendo così l'opportunità di creare viaggi in apparenza improvvisati. Le richieste del potenziale passeggero avvengono direttamente "on the road", permettendogli di trovare un passaggio ideale in breve tempo. L'adozione del carpooling dinamico per l'attuazione di infrastrutture tecnologiche diventa quindi una sfida di importanza fondamentale.

Questa tesi descrive Dycapo, un progetto open-source ideato con lo scopo di abilitare servizi di carpooling dinamico. Dopo una rassegna dello stato dell'arte ed un'analisi comparativa delle problematiche legate al carpooling dinamico, descriviamo le due componenti principali del progetto, ovvero il protocollo ed il server. Dycapo Protocol è un protocollo aperto basato sull'architettura REST per la condivisione di informazioni di viaggio tra client e server, prendendo spunto dalla OpenTrip, un protocollo precedentemente proposto. Dycapo Server è invece un prototipo per fornire servizi di carpooling dinamico, che dimostra ed implementa il protocollo Dycapo Protocol. Il nostro obiettivo, con il rilascio di un protocollo aperto e codice open source, è di fornire una piattaforma standard mancante, che i fornitori di servizi di carpooling dinamico possano adottare ed estendere.

Kurzfassung

Das Nützen eines privaten Transportmittels ist in den heutigen industrialisierten Staaten sehr üblich. Das ergibt aber zahlreiche Probleme wie z.B hoher Kraftstoffverbrauch, Verkehrsschlangen, Herzprobleme und eine ineffiziente Zeitnutzung.

Eine der möglichen Lösungen für diese Probleme ist „Carpooling“. Das bedeutet zum Beispiel das gemeinsame nutzen eines privaten Fahrzeuges mit ein oder mehrere Personen. Carpooling würde die Anzahl der Fahrzeuge drastisch reduzieren und somit weltweit soziale, ökonomische und umweltfreundliche Vorteile bringen. Die Anpassung und Paarung von Fahrern und Beifahrern kann mittels der neuen Informations- und Komunikationsaustauschtechnologien vereinfacht werden. Ein Fahrer kann womöglich die Anzahl der möglichen Beifahrer für seine geplante Reise eingeben und potenzielle Nutzer können diese somit finden und sich mit Ihm in Kontakt setzen. Dieser Prozess ist ziemlich lang und eignet sich für längere Reisen die von vornherein geplant sind. Diese Methode wird „statisches“ Carpooling genannt, hat jedoch trotz der zahlreichen Webseiten kein Erfolg.

„Dynamisches“ Carpooling jedoch basiert und erzielt Vorteil, von den immer mehr zunehmenden mit dem Internet verbundenen lokalisationsbewussten mobilen Geräten, die eine unmittelbare Reiseplanung ermöglichen. Beifahrer können Fahrziele direkt von überall abfragen und in nur einigen Minuten eine passende Mitfahrt finden. Zurzeit gibt es jedoch keine, im großen Ausmaß, benutzte dynamische Carpooling-Systeme. Die größte Problematik liegt bei der persönlichen Aufnahme des Produkts, jedoch gibt es auch Probleme die die Realisierung der Technologischen Struktur umfassen.

Dieses Dokument beschreibt Dycapo, ein open-source Projekt, welches dynamische carpooling Dienste ermöglicht. Nach einer Analyse und Studie der dynamischen carpooling Problematiken, werden die zwei Hauptteile des Projekts beschrieben, nämlich das Protokoll und der Server. Das Dycapo-Protokoll ist ein open REST-basierendes Protokoll für den Austausch von Reiseinformationen zwischen dynamischen carpooling Clients und Servern, welches Inspiration aus OpenTrip, ein vorheriges vorgestelltes Protokoll genommen hat. Der Dycapo-Server ist ein Prototyp, welcher Webdienste für dynamische Carpooling-Funktionalitäten, welche den Dycapo-Protokoll implementieren, zu Verfügung stellt.

Unser Ziel ist es mittels der Verbreitung eines offenen Protokolls und des open source Formats, einen zurzeit fehlenden Standard und Grundbaustein zu erstellen, welcher von anderen dynamischen carpooling Diensten angenommen und erweitert werden kann.

Acknowledgments

This project was carried out at the Fondazione Bruno Kessler, Trento, and would not have been possible without the support of my supervisor Paolo Massa, it has been a pleasure for me to work with him. I would like to thank in special all the people at the exploratory project SoNet for their support, with whom I had the opportunity to spend a great time, enjoy and learn a lot. Many thanks to Maurizio Napolitano for letting me know and enjoy this experience at FBK.

Another big thank goes to the Computer Science staff, in particular to our Administrative Staff, for their willingness, professionalism and friendliness.

Ringrazio di cuore la mia famiglia per avermi permesso di arrivare a questo obiettivo, per essermi sempre stata accanto, avermi supportato e sopportato durante le sessioni di esame (e non solo), senza farmi mai mancare niente nella vita.

Grazie infine a tutti gli innumerevoli amici (qualcuno più speciale) che mi hanno accompagnato in questo percorso, di cui non posso elencare i nomi perchè necessiterebbero un'entrata in Appendice. Farò un'unica eccezione per Riccardo Buttarelli, che ha accettato di accompagnarmi in questa avventura alla Fondazione Bruno Kessler, credere nella mia visione e sviluppare il suo progetto di tesi in sintonia con il mio. In bocca al lupo e grazie di tutto.

Contents

List of Figures

List of Tables

Part I
Introduction

Using a private car is a transportation system very common in industrialized countries. Between 2004 and 2009, the worldwide production of private vehicles has been of 295 millions of new units [1] and, as of 2004, there were 199 millions registered drivers in the U.S.A.[2]. Road transport is responsible for about 16% of man-made CO2 emissions[3].

Private car travelling is a common but wasteful transportation system. Most cars are occupied by just one or two people. Average car occupancy in the U.K. is reported to be 1.59 persons/car, in Germany only 1.05 [6]. Private car travelling creates a number of different problems and societal costs worldwide. Environmentally, it is responsible for a wasteful use of a scarce and finite resource, i.e. oil, and causes unnecessary earth pollution. The traffic caused by single occupancy vehicles causes traffic jams and hugely increases the amount of time spent by people in queues on streets. This is a unsavvy use of another scarce resource: time. Moreover, the additional pollution creates health problems to millions of individuals. Lastly, lone drivers in separate cars miss opportunities to meet and talk, incurring in a loss of potential social capital.

One possible solution to all these problems is carpooling, i.e. the act of sharing a trip on a private vehicle between one or more other passengers. The shared use of a single car by two or more people would reduce the number of cars on streets. Carpooling helps the environment by allowing to use oil wisely, to reduce earth pollution and consequent health problems. It reduces traffic and - consequently - time that people spend in their cars. Carpooling has also the potential of increasing social capital by letting people meet and know each other.

Carpooling is not a widespread practice. There are already many systems facilitating the match between drivers and passengers, most of them in form of bulletin board-like web-sites. The intention of offering empty seats of a vehicle is usually announced by a driver many days before the start of the trip. The coordination between a driver and the passengers who are candidating for sharing the trip with him/her is usually carried out by e-mails or private messages in the web-site.

Therefore, we may see carpooling as a *static* way of sharing a trip.

The availability of geo-aware, mobile devices connected to the Internet opens up possibilities for the formation of carpools in short notice, directly on streets. This phenomenon is called dyamic carpooling (also known as dynamic ridesharing, instant ridesharing and agile ridesharing). Dan Kirshner, researcher in this field and maintainer of http://dynamicridesharing.org website defines it as follows: "A system that facilitates the ability of drivers and passengers to make one-time ride matches close to their departure time, with sufficient convenience and flexibility to be used on a daily basis."[4].

Currently there are no dynamic carpooling systems widely used. In fact, there are many problematic issues related to the implementation and the adoption of dynamic carpooling systems. We analyze them critically in Part II of this dissertation. While we acknowledge all aspects are critical, we claim that the basic technological infrastructure is an important required and key building block. In fact we decide to focus on the creation of a solid, open and collaborative base framework for dynamic carpooling. The design and implementation

[1] (Accessed Sept. 9 2010) http://oica.net/

[2] (Accessed Sept. 9 2010) http://www.fhwa.dot.gov/ [U.S. Department of Transportation - Federal Highway Administration]

[3] (Accessed Sept. 9 2010) http://oica.net/ [Organisation Internationale des Constructeurs d'Automobile]

[4] Kirshner, D. (Accessed Sept 5th 2010) - http://dynamicridesharing.org

of an open protocol and an open-source server are presented in Part III.

Part II
State of the art

This part contains a summary of the state of the art regarding dynamic carpooling. It is divided in three sections. In the first section there is a summary of previously published papers, in order of publication. Then we introduce a brief analysis of the deployed systems. In the last section we present the outcomes of the analysis of the whole state of the art and how we decided to move in order to provide a significant contribute in solving the problem of adopting dynamic ridesharing services.

1 Review of published papers

During the research phase different papers were analyzed in order to obtain the state of the art. In this section we present a brief summary of each paper.

Sociotechnical support for Ride Sharing[11]

This paper lists barriers to dynamic carpooling adoption and possible actions to reduce them. It reports about High Occupancy Vehicles (HOV) lane - which are lanes dedicated for people doing carpooling - on streets of San Francisco and Oakland and complains that there should be no fees on bridges for HOVs. The author suggests conventions developed between drivers and passengers (e.g. pickup points near public transportation stops). Regarding security, the paper suggests to give priority to female passengers, to not leave them alone waiting for a ride. The paper reports that there are no stories about rape, kidnapping or murder and the most common reported problem is bad driving.

There are suggestions on needed research:

- Need of location-aware devices, because dynamic carpooling is actually limited to fixed pickups and drop-off locations.
- Simple user interfaces for passengers and drivers.
- Routing matching algorithms: short window of opportunity to match passenger and driver.
- Time-to-pickup algorithms: to help passenger decide whether to use carpooling or Public Transportation System.
- Safety and reputation system design: authenticate passenger and driver before making the match, monitor arrival at destination, feedback system.

The paper discusses about social capital impacts: there is the potential for creating new social connections and also matching drivers and passengers according to their profiles creates bridging across class, race and religious views.

Pilot Tests of Dynamic Ridesharing[8]

The author presents three pilot tests done in the USA, all of them failed. The reasons of failure are the following:

- Too complicated rules and user interface

- Too weak marketing effort
- Too few users. After 1 month, 1000 flyers distributed to the public and a proposed discount on parking, only 12 users were using the system.

The paper adds the idea of saving money when parking. It also enforces the idea of using social networks to allow car pooling on the fly. The author envisions using a web – and mobile service, also introducing some interesting user stories.

The smart Jitney: Rapid, Realistic Transport [10]

The paper focuses on environmental benefits of dynamic carpooling. It asserts that dynamic carpooling would lower greenhouse gas emissions in a better way than electric/hydrogen/hybrid cars would do. It introduces the idea of Smart Jitney: an unlicensed car driving on a defined route according to a schedule.

The author suggests the installation of Auto Event Recorders on cars, enforcing security. It complains that challenges are all focused in convincing the population to use the service, proposing a cooperative public development of the system.

Auction negotiation for mobile Rideshare service[1]

The paper proposes the use of agent-based systems powered auction mechanisms for driver-passenger matching.

Casual Carpooling - enhanced[7]

The author considers areas without HOV lanes and proposes the use of Radio Frequency IDentification (RFID) chips to quickly identify passengers and drivers. Readers should be installed at common pick-up points. The paper complains that it would cost less to pay passengers and drivers for using the service than to build a HOV lane.

Empty seats travelling[6]

This white paper by Nokia suggests to use the phone as a mean of transportation, creating a value in terms of a transport opportunity. It points out some factors limiting static carpooling, arranged via websites:

- Trip arrangements are not ad hoc
- It is impossible to arrange trips to head home from work or to drive shopping.

The paper notices that people are not widely encouraged to practice carpooling by local governments. It collects obstacles and success factors in terms of human sentences, and their solution. The authors say that the challenge is in the definition of a path leading from existing ride share services to a fully automated system.

Interactive systems for real time dynamic multi hop carpooling[5]

The author proposes a dynamic multi-hop system, by dividing a passenger route into smaller segments being part of other trips. The author claims that the problems of static carpooling are that matching drivers and passengers based on their destinations limits the number of possible rides, and with high waiting times. Carpooling is static and does not adapt itself well to ad hoc traveling. The paper asks governments to integrate carpooling

in laws and to push for its use. The author complains that the perceived quality of service is increased even driving the passenger away from destination: a driver and a passenger should not be matched only if they share the same or similar destination because perfect matching would require high waiting times.

The paper also addresses social aspects: in a single trip with 3 hops a passenger might meet 3 to 10 people, therefore passengers may be socially matched. It suggests to link the application with some social networks like Facebook, MySpace and use profile information to match drivers and passengers.

As security improvement, the paper suggests: the use of finger-prints, RFID, voice signature, display the location of vehicles on a map, using user pictures, assigning random numbers to be used as passwords.

Instant Social Ride Sharing[4]

The paper proposes matching methodologies based on both a minimization of detours and the maximization of social connections. It assumes the existence of a social network data source in which users are connected by means of groups, interests, etc. In such a network, the number of relatively short paths between a driver and a passenger indicates the strength of their social connection.

It provides algorithms and SQL queries. The authors assume that there is already a large scale of users, and no barriers to adoption are taken into account.

Combining Ridesharing & Social Networks[12]

The author envisions a mobile and web system that interacts with social networks profiles that should improve security and trust by users. Users can register to the system in a traditional way (e.g., by giving email, username, password), then complete their profiles by linking their accounts to multiple existing social networks account, to fill the remaining fields. Otherwise, they have to fill the fields manually and verify their identity in more classical ways. The paper proposes Opensocial[5] as connection interface. An own rating system is also complained, which keeps scores of persons. Amongst the criteria are factors like reliability, safety and friendliness.

It suggests the use of mobile systems, that should make use of GPS and creation of a match on the fly (real-time algorithms). The paper provides some results of surveys: people are willing to loose 23% more time to pickup a friend of their social network rather than a stranger (6%). It also provides a high-level description of the system and implementation details.

The author asks for extra research on psychological factors that increase trust and perceived safety.

SafeRide: Reducing Single Occupancy Vehicles [9]

The publication is about a project in the U.S.A. It reports that there is a market-formation problem: to achieve the system that attracts passengers, there will have to be many drivers available. But the drivers will emerge only when it appears profitable or otherwise desiderable, and that depends on there being many passengers, etc. The author complains that someone must discover a winning formula before anyone will invest.

The paper lists some interesting user stories, as well as algorithms and requirements.

[5]Google, MySpace et al. (Accessed Sept. 5th 2010) - http://www.opensocial.org/

Current Trends in Dynamic Ridesharing, identification of Bottleneck Problems and Propositions of Solutions [13]

This paper reports a state of the art of dynamic carpooling. The authors review several papers about the topic and some applications. We used this publication as a base for identifying problematics and to avoid duplication of work. The authors also identify barriers against the adoption of dynamic carpooling systems and propose solutions. In our comparative analysis, presented in Section 3, we report different proposals than those presented in this paper. Our proposals are complementary to those reported by this paper and are more focused in technological aspects.

2 Survey of deployed systems

After the theoretical research, also the existing systems were taken into consideration. The following list contains the existing dynamic carpooling applications and some static, web-based systems that are either innovative or well-known. All the reported websites were accessed on Sept. 5^{th}2010. Each text enclosed in double quotes is cited verbatim from the website of the application.

Carriva - https://www.carriva.org/MFC/app

It is a proprietary solution using phone calls as communication system and a fixed price of 0,10€ / km. Currently it has got 1118 active users.

Avego - http://www.avego.com

It is a proprietary application for Apple iPhone. It uses GPS technologies and presents a simple, intuitive user interface. It handles costs automatically. The passengers are not required to have an iPhone. It will offer information about public transports. The application relies on a proprietary service called Futurefleet, on which no implementation details are given. On October, 10^{th} 2009 the service offered 5310 empty seats.

Carticipate - http://www.carticipate.com

Carticipate is a proprietary iPhone application that integrates with Facebook, defined as "a location based mobile social network for ride sharing, ride combining, and car pooling". It has a very simple interface looking like Google Maps mobile. According to the website, it is available on 59 countries.

Piggyback - http://www.piggybackmobile.com/

It is an Android application using a step-by-step approach (maximum one user input at each application screen) and makes wide use of graphical representations instead of text. It offers the possibility to bookmark addresses. The map screen is proprietary. When a driver and passengers are matched their compatibility is showed, represented with stars (0 to 5) and categorized as friendliness, reliability, driving skills and car. The trip cost is also showed. After the ride, the feedback system lets the user set the points for the aspects listed above. The application lets also plan rides using a static carpooling approach.

Aktalita - http://www.aktalita.com/

It is an under development application, supposed to be proprietary.

"Aktalita combines the Web, a geospatially enabled database, and a Java enabled cellphone to provide real-time dynamic carpooling between drivers and passengers"

RideGrid - http://www.highregardsoftware.com/ridegrid-dynamic-ridesharing.html

Ridegrid is another proprietary, not yet in production system. "RideGrid is a service that uses mobile Internet and location technology to enable individuals to obtain rides to and from any location, spontaneously. [..] RideGrid works by dynamically combining routes. We evaluate the change required in a driver's route such that it passes through the desired source and destination of a compatible rider, and broker the agreement. We have proprietary means to calculate the routes, monetize the transactions, and introduce people to others they trust. "

It uses an internal credit system. The client has an outdated classical Java Micro Edition interface.

Project Carpool - https://launchpad.net/carpool

Carpool was the only open-source project, using PHP and Javascript. The development was stuck at the research time. The project is now closed.

GoLoco - http://goloco.org/

GoLoco is a proprietary web application that also relies on Facebook. It uses a private payment system.

Ecolane DRT - http://www.ecolane.com/

Ir is a proprietary solution, web-based, focused on security. It provides a customized Nokia touchscreen device. Among the features, they declare that the device is capable of real-time data communication, reports of arrivals and departures with time information, device locking mechanisms, GPS location and direction, mileage tracking, detailed trip information.

Divide The Ride - http://www.dividetheride.com/

The project is a static, web-based solution organized around children activities. Families invite other trusted families to join their group. Groups get notifications when a ride is needed.

iCarpool - http://www.icarpool.com

This application is a static, web-based system that does not require payments. They declare to use advanced proprietary algorithms for ride matching. "High precision trip matching. helps you find the best carpool match. Find co-workers, neighbors and friends for carpool. Use for daily commute, recurring trips, long distance trips and events Plan ahead or use on-demand". Matching criteria includes social relationships, but no details are given.

Hover - http://www.hoverport.org/

It is a casual carpooling system using RFID technologies and an own credit system. The members are approved after human verification tests. Participants must meet at a location called "Hover Park" and are identified by the RFID system. On exiting the Hover Park, the system recognizes driver and passengers and distributes credit points. There are several destination points available, that register the arrival in the same way. It also offer a guaranteed back-to-home system, by using taxis.

Flinc - http://www.flinc.mobi/

Flinc is a dynamic carpooling system using smartphones (Android and Apple iPhone). "Flinc connects navigation systems and mobile phones and arranges available seats within a few seconds - directly in the car and on the pavement. Flinc combines GPS and location-based capabilities with social networking to offer a dynamic and automated method of getting from one place to another. The service can be used on smartphones or on the PC or Mac, helping users create rides within a few seconds via an entirely automated process.". The system is currently under active development.

3 Comparative Analysis of Dynamic Carpooling Issues

The analysis of the state of the art brought some issues related to adopting dynamic carpooling systems, many of them recognized also by [13]. We categorized the issues gathered from the state of the art and their proposals in the following categories:

- Interface Design - all issues related to graphical implementation of clients and ease of use
- Algorithms - the instructions regarding driver/passengers matching problems
- Coordination - the aspects related on how to let people meet, authenticate and coordinate.
- Trustiness - the problems related on raising user confidence on dynamic carpooling systems
- Safety - the issues regarding ensuring protection of users
- Social Aspects - all the issues related to create social connections and raising social capital in dynamic carpooling systems
- Reaching Critical Mass - the problems on reaching a sufficient amount of persons using the system that would attract more other people
- Incentives - all the political, motivational and economical issues related to dynamic ridesharing systems
- System Suggestions - everything else that we consider relevant for building dynamic carpooling systems

We attach the comparative analysis summarized in tables, in Appendix A, Table 2 up to table 6, for each category (columns), we list the suggestions and interesting points made in the different research papers (rows). Tables 2 to 6 in Appendix A is our contribution

rationalizing the many problematic issues involved in the creation and deployment of dynamic carpooling systems and in summarizing best practices and suggestions in how to deal with them.

Our rationalization of dynamic carpooling issues and possible solutions shows how dynamic carpooling systems still have many important open issues to be addressed and solved. This fact explains the current absence of any dynamic carpooling system deployed and used for real. We decided to address the overall challenge from a very core point of view and to focus on technical aspects. Among them, we observed that the source code of the projects and the prototypes produced was not freely accessible by the general public. There are no information regarding the servers, that are all proprietary and obscured. Another issue seems related to a missing standardization of the protocols used. Therefore, every project started from zero, "reinventing the wheel". While, in order to overcome the "reaching critical mass" issue, we believe that it is important that providers of dynamic carpooling services can exchange their data easily so that cross provider matching are possible.

Based on this analysis, we decided to create two basic technological building blocks:

- An open, discussable protocol for communication between dynamic (and static) carpooling systems
- An open-source server prototype implementing the protocol

We decided to release the protocol under Creative Commons License and to release the source code as open-source because our goal is to fill a void and provide a basic infrastructure that future providers of dynamic carpooling services can adopt, extend and in general build on.

Therefore, the Dycapo project was born.

Part III
Dynamic Carpooling Project: Dycapo

4 Overview

Dycapo is the name given to a project aiming to share knowledge outcomes and technological products on dynamic carpooling. The information is created using a fully open and collaborative system. The website, http://dycapo.org, is the start point and the container of each project component. It is a Wiki, freely accessible and discussable. The Wiki content and the source code are available under permissive and open licenses. We used a blog and social networks to update in real-time about the status of the researches and the development. The website was opened on Oct. 11 2009. As of Sept. 11 2010 we had 1617 visits and many e-mail contacts, about proposals of collaboration and general interest. This approach is part of the strategy we chose to become a world standard for dynamic carpooling systems.

Dycapo Project is composed of four main parts:

- Research - this part is needed in order to understand the state of the art about dynamic carpooling.
- Protocol - a formal specification of the communication protocol we propose as a standard for the exchange of information among dynamic carpooling clients and services.
- Server - the prototype implementing dynamic carpooling functionalities and the protocol
- Client - will act as a start point for the clients implementing the protocol

The research part has been covered in Part II of this dissertation. A client is under development by a university colleague and will be object of another Bachelor thesis. Our analysis of the state of the art guided our decision to focus on technological aspects, precisely server and protocol.

In this section we first introduce some terminology related to our domain of interest. For the same reason, we then present two user stories on system functionalities. In the following sections we present a general architecture of Dycapo, the open protocol proposed as standard and the server prototype implementing the protocol and dynamic carpooling functionalities.

4.1 Terminology

In Table 1 we introduce some key concepts used in both the protocol and the server for defining entities operating in the system. The remaining concepts will be explained in section 5.1.

Term	Definition
Person	A user registered in the system, with login and password
Trip	The Driver can create Trips in the system. A Trip is the information about the availability of seats in a car going from a certain location to a certain destination, driven by a Driver on a certain date
Driver	The role of a Person when he/she offers to share some seats on his/her car for a specific trip
Passenger	The role of a Person when he/she accepts to occupy a seat on a car of a Driver
Participation	The act of taking part into a Trip. The Driver participates by default in a Trip he/she created. A Passenger can participate in Trips created by a Driver. Participation can be just requested by the passenger or already confirmed by the driver.
Location	A geographical location.

Table 1: Key Terms of Dycapo

Formal relationships such as the fact that a Trip can have more related participations, are summarized in the section about the internal working of the server, in Figure 12.

4.2 User Stories

The following user stories are for introducing the reader in the domain of interest related to our dynamic carpooling protocol and server. For each user story, we specify in parentheses the appearance of terms introduced in Table 1.

A simple trip

Paul (a Person) is in Via Roma, Bolzano, Italy (a Location). He wants to move to the cemetery (a Location). He takes out his internet-enabled smart-phone with GPS activated, opens his Dycapo client and gives the desired destination, therefore searching for an available trip (and becoming a Passenger). Angela (a Driver) is travelling with her car along Corso Libertà, Bolzano (a Location). The destination of her previously stored trip is Laives, Italy, southern of the cimitery. She receives a notification from her Dycapo client running on her smartphone, located in a car docking station. The notification contains Paul's position and she accepts his participation (Participation) request. The client displays the directions for reaching Paul, using the GPS-chip . Paul (a Passenger) and Angela meet and share the trip.

A planned concert

Anna lives in Trento, Italy (a Location). In two days she will attend a concert in Milano, Italy (a Location). She takes out her internet-enabled smart-phone running a Dycapo client and inputs her travel intentions, therefore creating a trip. Two days after she starts her trip. Just after her start, she discovers that Mary is in Rovereto, Italy (a Location) and would like to have a ride (a Participation) to Verona, Italy. The clients make them meet and share part of Anna trip. While driving Mary to Verona, Anna receives another ride request (a Participation) from John, who is in Brescia center, Italy and wants to

travel to Milano, too. Anna realizes that the deviation from her planned route is too much and simply refuses the request (a Participation). Anna arrives to Milano and enjoys the concert.

4.3 High-level architecture of the system

Dycapo system follows the client-server model, in which a server receives messages from clients (running on geo-aware mobile phones) using a protocol, processes them and returns them using the protocol again. In Figure 1 we can see three smartphones that, having a client software, communicate with Dycapo servers using the protocol. The server system operates the data received from the clients (e.g. performs matching between a Driver's trip and a Passenger search of a trip) and returns the results using the same protocol.

If the open protocol becomes a de-facto standard, the same client will be able to communicate with different servers, possibily operated by different operations. In this future scenario, server-to-server exchange of information is also possible, thanks to the protocol.

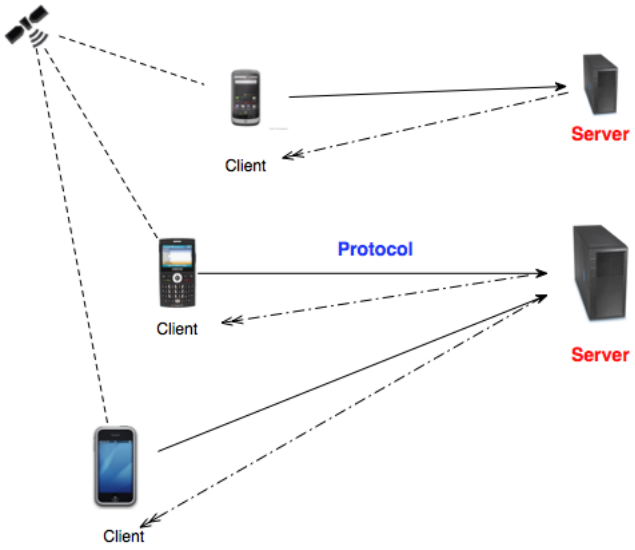

Figure 1: High level view of Dycapo components architecture

5 Protocol

Dycapo Protocol (also known as DycapoP) is an open application-level protocol for enabling communication between dynamic (and static) carpooling servers and clients, using HTTP for operations and JSON (JavaScript Object Notation) as data exchange format. The protocol models are inspired by OpenTrip Core[6], a former proposal of data exchange format for carpooling and dynamic carpooling presented during the MIT "Real-Time" Rideshare Research workshop[7]. DycapoP is released under Creative Commons Attribution-ShareAlike 3.0 Unported license[8] and documented at http://dycapo.org/Protocol.

The two main components of the protocol are: elements and operations. *Elements* are the constructs to represent the entities involved in the protocol, for example a Person (as

[6] http://opentrip.info/wiki/OpenTrip_Core
[7] http://ridesharechoices.scripts.mit.edu/home/workshop/
[8] http://creativecommons.org/licenses/by-sa/3.0/legalcode

are Paul and Angela in the first user story) and a Location (as it is "Via Roma, Bolzano, Italy" in the first user story). *Operations* are the actions that clients can perform on the elements to obtain desired results. As example, the "searching a trip" action performed by Paul in the first user story and "Anna [..] refuses the request" are translated as protocol operations.

5.1 Elements

The elements defined by OpenTrip Core (using Atom Syndication[9] format) have been extracted and represented using UML 2.1 Class Diagrams. While proceeding and discussing the development of the server, those elements were extended and adapted. The elements are represented in the Protocol using descriptive tables and real world code snippets. The Dycapo Protocol draft version of September 27, 2010 is included in the Appendix B. Each element has a *href* attribute, that holds the unique URL for doing operations on the resource. It will be further explained in sub-section 5.2.

The following are the elements of Dycapo Protocol:

Location A Location (Figure 2) represents a geographical position, using human understandable attributes (like *street*, *town*, *postcode*, *region*, *subregion* and *country*) or geolocation values such as *georss_point*[10].

The *leaves* attribute is responsible for holding time-based information. It is used for indicating the left of a location when starting a trip or a ride. It is also used for holding the timestamp of the location of a Person and the predicted arrival at a place.

Person A Person (Figure 3) represents a user of the system and contains useful information for building social connections or search preferences.

Figure 2: Location Element

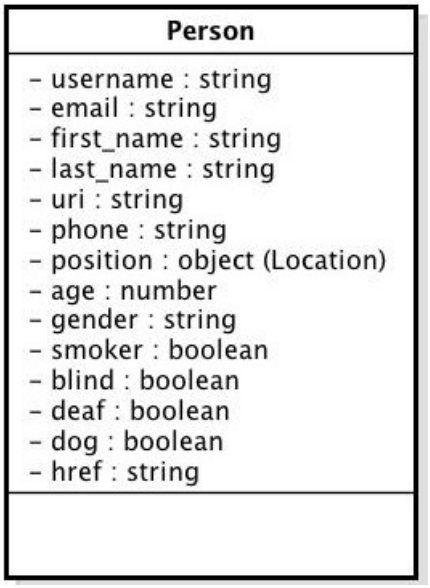

Figure 3: Person Element

[9] http://www.ietf.org/rfc/rfc4287.txt
[10] http://www.georss.org/simple#Point

Modality A Modality (Figure 4) element is a description of the mode of transportation being used by the Person offering a Trip and a representation for the cost of the Trip.

Preferences A Preferences (Figure 5) element represents the preferences of the Driver when performing a Trip, such as the age range of the Persons he will accept as passengers, whether they can smoke during the Trip, etc.

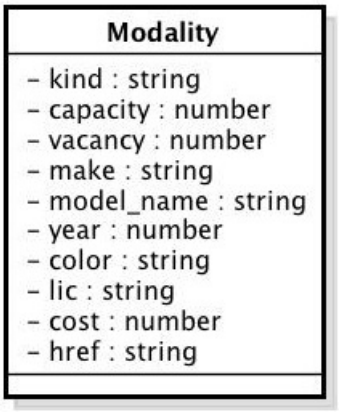

Figure 4: Modality Element

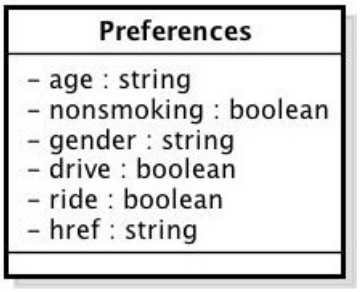

Figure 5: Preferences Element

Participation A Participation (Figure 6) element represents the fact of taking part in a Trip. The attribute *status* represents the status of a Participation, being it a request, a refuse, a start etc. This element is a central part of DycapoP, as it is used since the beginning to end of a ride of a Passenger. The majority of the operations of the protocol are built around a Participation element.

Trip A Trip (Figure 7) represents a single course of travel taken as part of one's duty, work, etc, offered by a Person. It is the most complex object of the Dycapo Protocol. It contains all the information needed for doing operations with Trips. Following the philosophy behind OpenTrip's Trip element, it contains datestamp information such as *published*, *updated* and *expires*, respectively representing the creation, the last edit and the expiration of the trip. A Trip contains other Dycapo Protocol elements such as the *modality* and the *preferences*, explained above. The *locations* element is an array of Location objects, and holds the start, the end and all the intermediate geographical points of the trip.

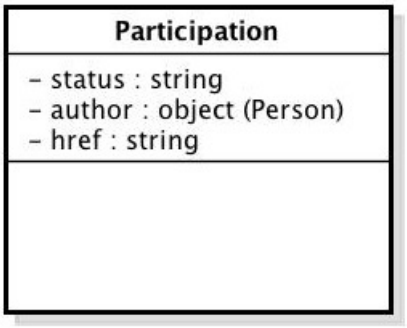

Figure 6: Participation Element

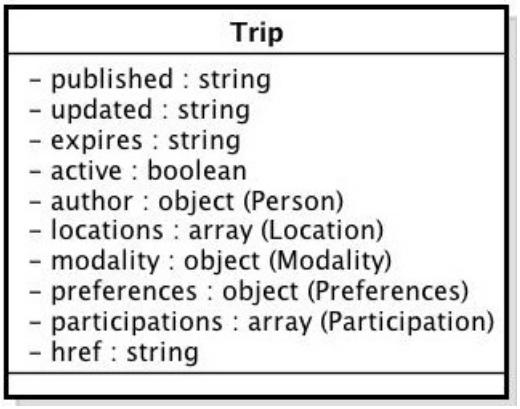

Figure 7: Trip Element

Search A query to the server when a passenger searches an available trip is represented by a Search element (Figure 8).

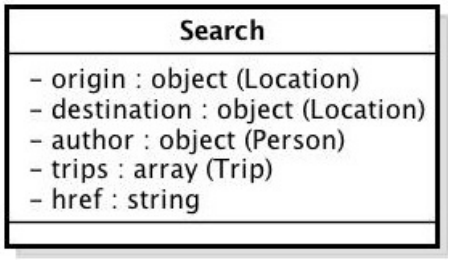

Figure 8: Search Element

Attributes of elements can be mandatory or optional. The specific settings for each entity are described in the complete protocol at Appendix B.

5.2 Operations

DycapoP is implemented using a resource oriented architecture, i.e. the definition of a full REST-based [3] application-level protocol. That is, all DycapoP elements are represented as web resources encoded in JSON exchange format. Each element is referenced through a unique URL, stored in their *href* attribute. All the operations of the protocol are implemented and performed using HTTP methods, i.e. GET, POST, PUT, DELETE of resources. As

For example, to retrieve a particular trip, a client must perform the following HTTP request:

```
GET /api/trips/3/ HTTP/1.1
Authorization: Basic cmlkZXIxOnBhc3N3b3Jk
Host: test.dycapo.org
```

It is self-descriptive: a GET against the collection namespace of trips, specifying in the URL that we want to retrieve the Trip with id 3, using HTTP version 1.1. The authorization

line is present because in our implementation of the protocol the authorization is done using HTTP Basic Authentication[11]. The Host part is required by HTTP version 1.1.

The server will respond in the following way, using a HTTP response.

```
HTTP/1.1 200 OK
Date: Mon, 13 Sep 2010 14:35:26 GMT
Server: Apache/2.2.14 (Unix) mod_ssl/2.2.14 [..]
Vary: Authorization,Cookie
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
X-Pad: avoid browser bug
922
{
"updated": "2010-09-02 16:03:24",
"participations": {

    "href": "http://test.dycapo.org/api/trips/3/participations/"
    },
"preferences": {
    "nonsmoking": false,
    "gender": "",
    "ride": false,
    "drive": false,
    "href": "http://test.dycapo.org/api/trips/4/preferences/",
    "age": "18-30"
},
"author": {
    "username": "driver1",
    "gender": "M",
    "href": "http://test.dycapo.org/api/persons/driver1/"
},
"expires": "2010-09-05 16:03:05",
"locations": [ {
    "town": "Bolzano",
    "point": "orig",
    "href": "http://test.dycapo.org/api/trips/3/locations/",
    "country": "",
    "region": "",
    "subregion": "",
    "days": "",
    "label": "Work",
    "street": "Rom Strasse",
    "postcode": 39100,
    "offset": 150,
```

[11] http://www.ietf.org/rfc/rfc2617.txt

```
        "leaves": "2010-09-02 16:02:22",
        "recurs": "",
        "georss_point": "46.490200 11.342294"

    }, {

        "town": "Bolzano",
        "point": "dest",
        "href": "http://test.dycapo.org/api/trips/3/locations/",
        "country": "",
        "region": "",
        "subregion": "",
        "days": "",
        "label": "Work",
        "street": "Piazza della Vittoria, 1",
        "postcode": 39100,
        "offset": 150,
        "leaves": "2010-09-02 16:02:22",
        "recurs": "",
        "georss_point": "46.500740 11.345073"

    } ],
    "href": "http://test.dycapo.org/api/trips/3/",
    "published": "2010-09-02 16:03:05",
    "modality": {

        "kind": "auto",
        "capacity": 4,
        "lic": "",
        "color": "",
        "make": "Ford",
        "vacancy": 1,
        "cost": 0.0,
        "href": "http://test.dycapo.org/api/trips/3/modality/",
        "year": 0,
        "model_name": "Fiesta"
        }

    }
```


The first lines are common HTTP response parts. The status line is a HTTP 200 OK, specifying that the request has been successful. The Content-Type header informs that the server is returning in JSON format that we now briefly analyze. What is returned in this example is a Trip object, enclosed in curly brackets. Each Trip attribute is in the form of: *"attribute name" : attribute-value*. All attributes are separated by a comma. Some members have simple value types like boolean, number and string, e.g. *published*, *href*. Other attributes have complex objects as values, e.g. *modality*, *preferences*. The *locations* attribute is represented as an array of Location elements, enclosed in square brackets.

The protocol operations are available in Appendix B. This dissertation focuses on the implementation of the functionalities in Part IV.

In Figure 9 we represent what we expect by clients when operating on a Participation element, using a UML State Diagram. For each transition, we report the value of attribute

status and the HTTP method performed by the operation. We also report those transitions that only the author of the Trip (i.e. the driver) can cause.

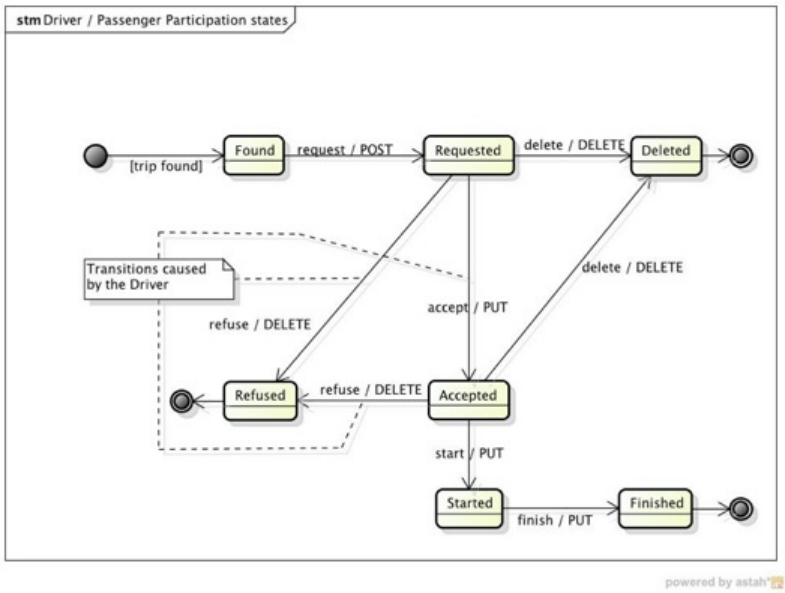

Figure 9: Participation Status

The UML 2.1 sequence diagram in Figure 10 summarizes a full user story - the creation of a Trip, the participation of a passenger and its end. We comment the first passages, up to the passenger ride request is accepted.

To create a Trip in Dycapo system, the driver client performs a HTTP POST operation of a Trip element against the URL (Message 1 in Figure 10). The POST request contains in its body the representation of the Trip (Figure 7) element using JSON notation. The server responds using a HTTP 201 Created message, returning in the response body the same Trip representation with an additional *href* attribute containing the unique URL for future accessing the stored resource. The passenger searches a trip by doing a POST request (Message 2 in Figure 10) containing in its body a representation in JSON of a Search (Figure 9) element. The server returns a HTTP 201 Created response with the Search element in its body, providing in the *href* attribute the URL for accessing the resource. In Message 4 the driver sets the Trip as active by doing a PUT request against the Trip previously stored, passing the updated Trip element. Afterwards, as illustrated in Message 5, the passenger retrieves the Search resource using a GET request against the unique URL of the Search resource, that will contain the Trip inserted by the driver. The passenger then requests a ride by posting a Participation element (Message 6) using the participations URL provided by the Trip element. The driver accepts the ride request by obtaining the participations of the Trip (Message 7) using a GET request, retrieving the passenger participation, modifying its status attribute to "*accept*" and updating it using a PUT request.

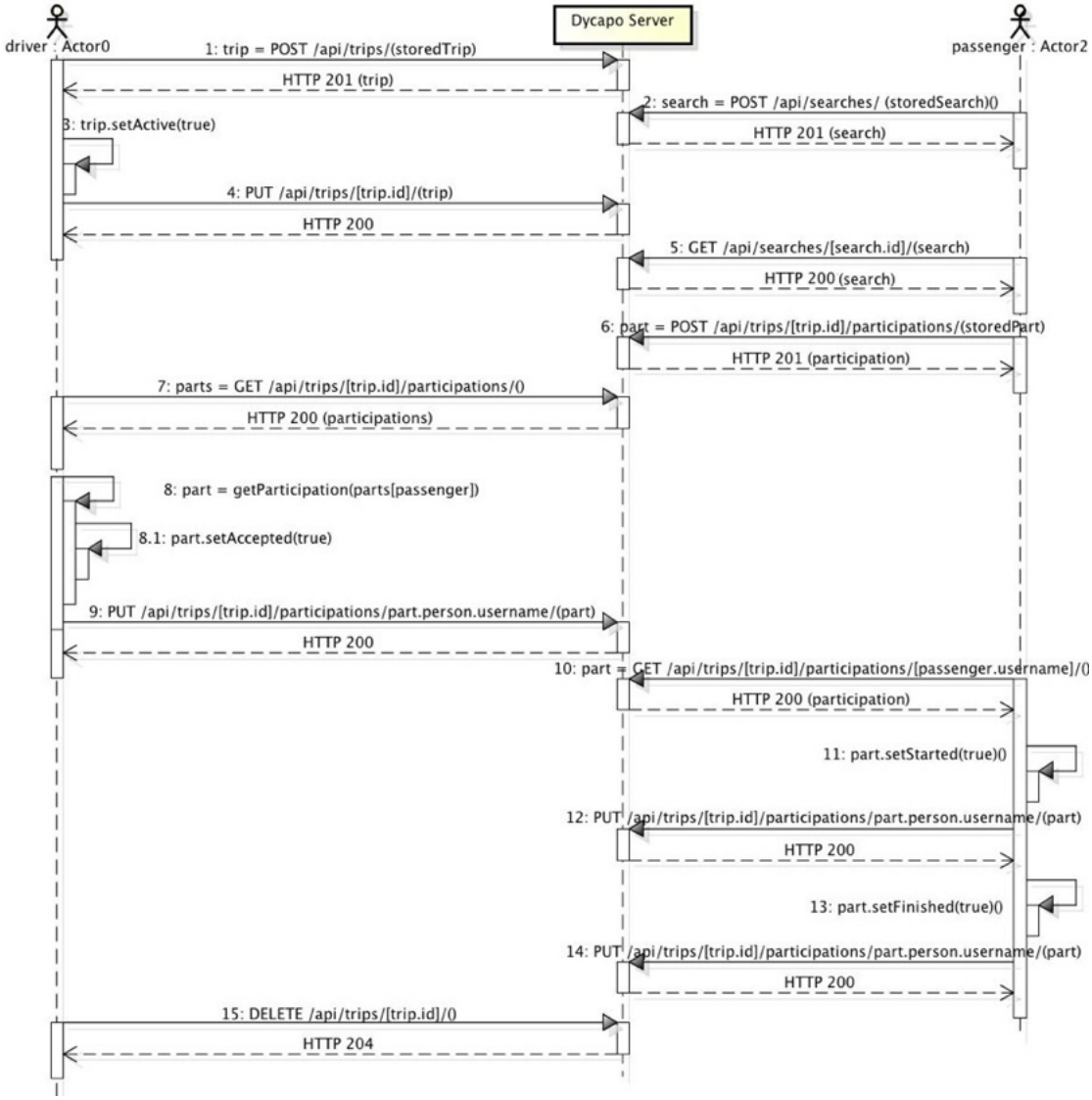

Figure 10: Interaction of two clients and Dycapo Server using Dycapo Protocol

6 Server

Dycapo Server (also known as DycapoS) is a software built using first throwaway then evolutionary prototyping - as defined by [2] - to demonstrate and support the definition of a standard protocol for dynamic carpooling. In this part we describe the software components, the enabling technologies, the models and the functionalities.

DycapoS source-code is available in an open GIT repository [12], under the Apache License, version 2.0[13]. The project was opened on Nov. 14 2009 and as of September 27, 2010 counts 177 commits, 14 git tags and 5 branches.

[12] http://github.com/BodomLx/dycapo
[13] http://www.apache.org/licenses/LICENSE-2.0.html

6.1 Enabling Technologies

DycapoS makes use of several open-source systems for achieve its goals. We list them in this sub-section.

- Python programming language and Django web framework[14]. Python is a general-purpose high-level and multi paradigm programming language, known for a human understandable, clear syntax. Django is a web framework following the Model-View-Controller architectural pattern. Even if Django is focused on a rapid creation of rich-content web-sites, its openness let the community create add-ons for different purposes, as it is in our case. DycapoS uses the following components of Django:
  - The object-relational mapper - that operates between Python objects, defined as classes, and a relational database, letting developers operate at object level instead of writing SQL queries.
  - The regular-expression-based URL dispatcher.
  - The authentication system that easily provides wrappers for doing HTTP basic authentication against stored users.
- Geopy[15], a geocoding library for Python programming language that helps to locate the coordinates of addresses across the world using third-party geocoders services. It also provides reverse geocoding functionalities, i.e. the retrieval of human-understandable data of a physical location, starting from its coordinates.
- Piston[16], a Django add-on that provides the creation of RESTful APIs.
- Py.test[17], a mature unit-testing framework for Python programming language
- Apache webserver and PostgreSQL database management system.

6.2 Components

Dycapo Server is written using Python programming language, which allows a partitioning of a software into modules and packages. A module is a component providing execution statements and definition of functions, variables, classes, etc. Each module correspond to a single Python file. A module has its own private symbol table. A package is a set of modules or other sub-packages, implemented as a directory. Dycapo Server is organized using separation of concerns, each concern held in a separate package.

We represent in Figure 11 a high-level overview of Dycapo Server components, here described:

- rest - this module holds all the wrappers of the resources exposed to the public using REST architecture. It contains a Python module for each DycapoP element, that defines behaviors to perform depending on the HTTP request done by clients. That is, each Python module contains a class defining behaviors to be done for each HTTP operation implemented.
- piston - this module contains an open-source framework that lets us create application programming interfaces implemented with REST principles.

[14](Accessed Sept. 22 2010) http://djangoproject.com
[15](Accessed Sept. 22 2010) http://code.google.com/p/geopy/
[16](Accessed Sept. 22 2010) http://bitbucket.org/jespern/django-piston/wiki/Home
[17](Accessed Sept. 22 2010) http://codespeak.net/py/dist/test/

- server - in this module we implemented the server functionalities and models. See sub-sections 6.3 and 6.4 for more information
- geopy - in this module we have a geocoding library for Python for our matching algorithm and for completing Location objects when received from clients.
- tests - in this module we have a full testing suite of REST functionalities (and therefore, the server internal functionalities). Actually, a full test run performs 199 HTTP requests doing multiple asserts on each call.

A HTTP request to the webserver connected to the Internet is wrapped by our Python web framework as a Python object. The requested URL is evaluated by an internal dispatcher that passes the control to the rest package. The *rest* package converts the resource representation provided to a Python object defined in the *server* module. Then it performs some actions and computation by calling a component of the *server* module. The *server* module eventually saves or retrieves states by querying the database and returns results, that are again converted by the *rest* package in JSON format and returned by the web server.

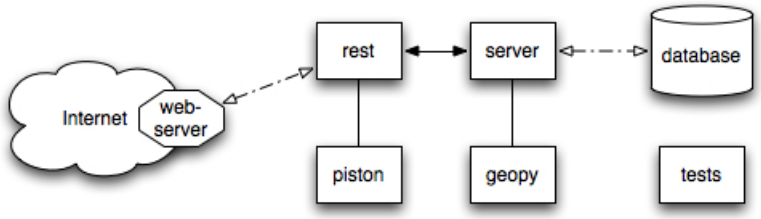

Figure 11: Dycapo Server components

6.3 Models

Dycapo Server models are developed as Python classes and trace the elements of Dycapo Protocol. They are stored in the models sub-package inside the *server* package. Since we already mentioned them in Section 5.1, we include in Figure 12 a UML 2.1 Class diagram to represent them and their relations.

The slight differences in their Python implementations are in their data type, that are a specialization of the data type of DycapoP attributes. There are other tiny differences. For example, we have separate attributes in Location for GeoRSS longitude and latitude, for facilitating database queries.

In Figure 12 we also have a Response class, with a yellow background. Response is a special object type used internally in DycapoS for wrapping return values and exceptions. Its attribute *code* is for representing statuses, using HTTP status codes. Attribute *type* is a helper attribute to include the type of the object returned by functions, or exception. The attribute *value* holds the value returned from functions, and can be of any Python type.

The Participation class is the only one which is completely different when compared to the one in DycapoP (Figure 6). Its implementation is more complex because we decided to implement the status as a set of boolean values, for facilitating database queries. Through a helper method, the combinations of the boolean values are translated to the human-understandable strings of the *status* attribute of the Participation element of DycapoP (Figure 9).

Moreover, we store important safety-related data for each Participation statuses, that are the timestamp and the Person location related to the change of status.

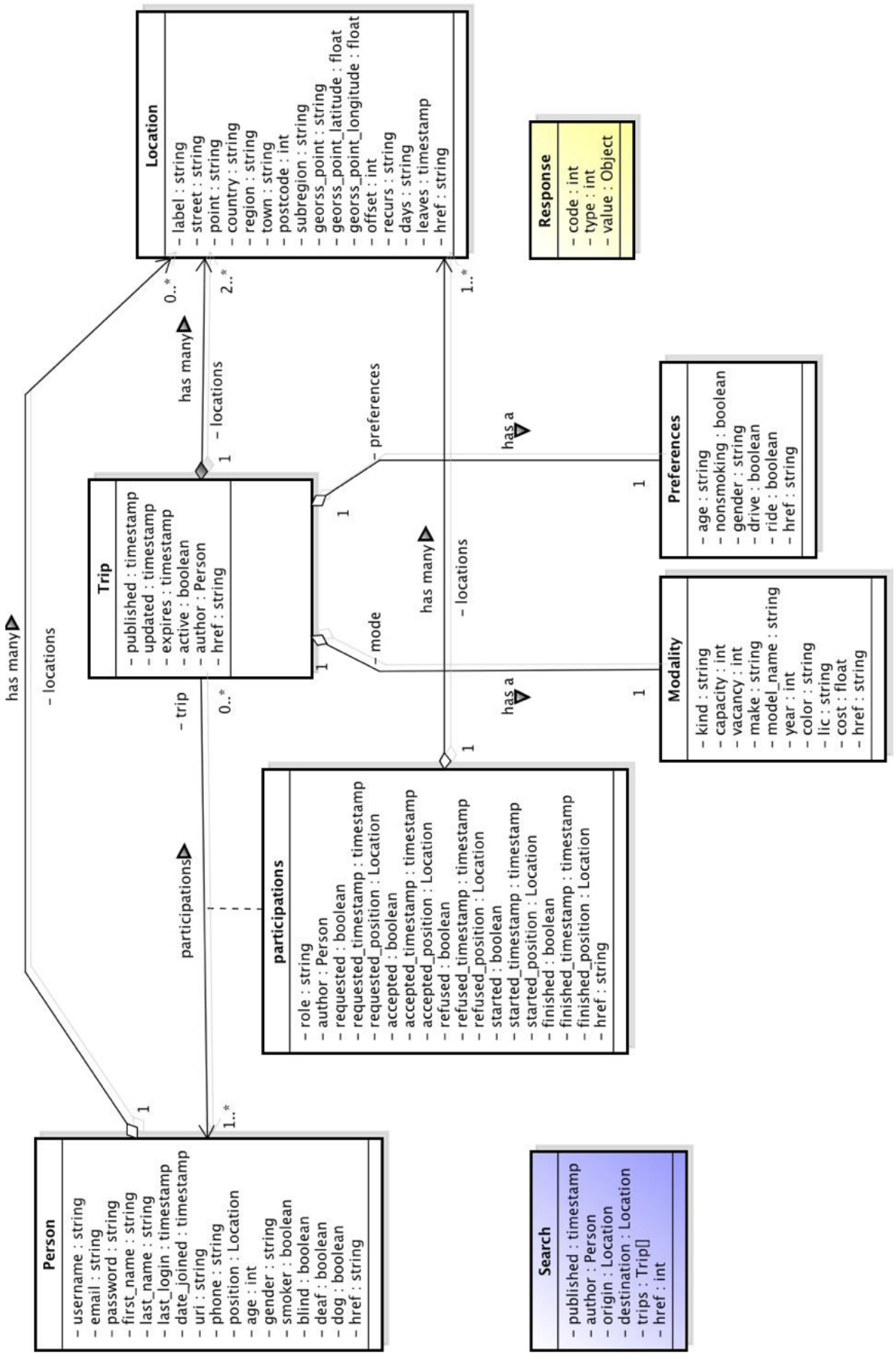

Figure 12: Dycapo Server class diagram

6.4 Functionalities

All the functionalities of DycapoS are stored in packages as pure functions, inside the server module. They are divided in five packages, according to a separation of concerns division. In this sub-section we list the functionalities, we include a short comment on what they perform and we provide how they are generally used by REST operations.

Common

This package contains all the functions that a driver and a passenger have in common. That is, all the operations regarding geographical position, registration and profile changes are in the common package.

`setPosition(current_user, position)`

It verifies the validity of the Location object, then links it to the given Person, as current Person position and/or Participation location depending whether the person is participating to a Trip or not.

`getPosition(current_user, person)`

It verifies the privacy-related permissions of both users given - the first one requesting the position of the second one, then eventually returns the location of the Person.

`register(person)`

This function registers a Person to the system.

`updatePerson(current_user, person)`

It updates current_user Person with the attributes of person Person. We use this function for user profile changes.

`changePassword(person)`

It changes the password of the user, supplied in person.password

Driver

In the driver package we implemented all the functionalities needed by a Person when he/she assumes the role of a driver. That is, all the operations regarding the manipulation of Trips and acceptation/refusal of Trip Participation are developed in this package.

`insertTrip(trip, author, source, destination, modality, preferences)`

This function inserts a Trip in the system. It may be either started or not. It also checks the validity of each object supplied

`startTrip(trip, driver)`

It sets a Trip as started.

`getRides(trip, driver)`

This function returns all pending ride requests for a trip

`acceptRide(trip, driver, passenger)`

It accepts a ride request of a passenger

`refuseRide(trip, passenger)`

This function refuses a ride request of a passenger

`finishTrip(trip, driver)`

It sets a Trip as finished.

Passenger

This package contains the functionalities needed by a Person when he/she assumes the role of a passenger. That is, all the operations regarding the search of a Trip, the request/cancellation/start/end of a Participation are implemented in this package.

`searchRide(source, destination, passenger)`

Given a source, a destination and the passenger, this function searches for a suitable active Trip.

`requestRide(trip, passenger)`

It sends a request for a ride to the Trip author.

`statusRide(trip, passenger)`

It returns the status of the Participation related to the Person passenger.

`cancelRide(trip, passenger)`

This function deletes a ride previously requested by a passenger.

`startRide(trip, passenger)`

It lets the system know that a ride successfully started, i.e. the driver has arrived to pick the passenger

`finishRide(trip, passenger)`

It lets the system know that a ride successfully finished, i.e. the Passenger has arrived to destination

Matching

We created the matching package separated from the passenger package for better handling its functionalities. This package holds the matching algorithm of Persons when a Trip is searched. The current matching algorithm has been implemented ex-novo by this dissertation author. It is based on air distances, not on real routing vector. It is considered suitable for testing the server prototype by a tiny community of conscious volunteers but not for real-world usage. The algorithm is heavily tested by the testing framework of DycapoS.

`search_ride(location, passenger)`

This function returns all the Trips with a destination near a given location, by creating a virtual geographical box around the location. We query the database for all active Trips, that also have seats available, searching a destination that is inside this box. The retrieved trips are then filtered by looking if the driver air position from the destination is greater than the passenger position. Then, the remaining trips are filtered by calculating if the driver tends to move approaching the passenger position or to move away from him/her. The remaining trips are returned by the function. See the remaining helper functions for more information.

`get_proximity_factor(person, position)`

Given a person and a location, this function determines if the person is approaching it or getting away from it, by retrieving some recent locations of the person and computing their distance from the location. The set of ordered distances is then passed to location_proximity_factor that retrieves the proximity factor.

`location_proximity_factor(distances)`

Given a list of distances, it computes the approaching factor which is a natural number in the interval (-inf , +inf). If the factor is greater than 0, the numbers in list tend to decrease and therefore, the driver is approaching the passenger. If the factor is 0, the numbers in list tend to stay around the same value and therefore, the driver is neither approaching nor moving away from the passenger. If the factor is less than 0, the numbers in list tend to increase and therefore the driver is moving away from the passenger.

Utils

This package holds some utility functions related to time and object's attribute synchronization.

REST to Server mapping

DycapoS is built over a resource-oriented architecture (that is REST) but internally implements functions as if it was a service-oriented architecture. As we introduced in sub-section 6.1, a HTTP request is dispatched to the *rest* package, that contains the handlers responsible for react to the requests according to the HTTP method used. The handler overrides up to four methods, one for each HTTP method available. The method is responsible for calling the correct function inside the *server* package and return its results.

As example, we consider the most simple handler of DycapoS, that is the Location handler for Person's locations:

```
import piston.handler
import piston.utils
import server.models
import server.utils
import server.common
import rest.utils

class LocationPersonHandler(piston.handler.BaseHandler):
    allowed_methods = ['GET','POST','PUT']
    model = server.models.Location
        fields = ("href","town","point","country","region",
                            "subregion","days","label","street","postcode",
                "offset","leaves", "recurs","georss_point")

    def read(self, request, username=None):
        user = rest.utils.get_rest_user(request)
        try:
            if username:
                person = server.models.Person.objects.get(username=username)
                result = server.common.getPosition(user,person)
```

```
                return rest.utils.extract_result_from_response(result)
                    else:
                return piston.utils.rc.NOT_FOUND
            except server.models.Person.DoesNotExist:
                return piston.utils.rc.NOT_FOUND
            except server.models.Location.DoesNotExist:
                return piston.utils.rc.NOT_FOUND


    def create(self, request, username):
        user = rest.utils.get_rest_user(request)
        if user.username != username:
            return piston.utils.rc.FORBIDDEN
        data = request.data
        location = server.models.Location()
        dict_location = rest.utils.clean_ids(data)
        location = rest.utils.populate_object_from_dictionary(location,
                                    dict_location)

        result = server.common.setPosition(user, location)

        if result.code == server.models.Response.CREATED:
            result.value.href = rest.utils.get_href(request,
                                "location_person_handler", [user.username])
            result.value.save()
            return result.value
        else:
            return rest.utils.extract_result_from_response(result)

    def update(self, request, username):
        return self.create(request, username)
```

The class LocationPersonHandler extends BaseHandler defined by the Piston framework. The lines between 9 and 13 are instance variables for enabling/disabling HTTP methods, for setting the Django model to be mapped as REST resource and for choosing which model attributes to expose to the public. The methods defined afterwards are those that manage the HTTP requests according to their HTTP method:

read() - lines 15 to 29 For handling HTTP GET requests. We retrieve the user doing the GET request in line 16. Then, according to the presence of the *username* parameter, we either return a HTTP 404 error code (wrapped inside the piston.utils.rc.NOT_FOUND object) or we call the *server.common.getPosition()* function - presented in sub-section 6.4.1. We then quite-directly return the results of the function, using the helper function *rest.utils.extract_result_from_response()*, since each function in the *server* package returns a Response object (discussed in section 6.3).

create() - lines 32 to 48 For managing HTTP POST requests. As we do for the *read()* method, we retrieve the user doing the GET request, and return a HTTP 403 Forbidden if the user trying to perform the operation is not the same user targeted for it. Then, in

a similar way as we do for the *read()* method, we prepare the Location object with the data provided in the POST request (lines 34-38) and pass it to the *server.common.setPosition()* function. Again, we return to the client according to the Response object provided by the function. We populate the location's *href* attribute if the operation was successful.

update() - lines 50 to 51 For handling HTTP PUT requests - in the current implementation, we just perform a call to the *create()* method.

delete() - not in the example For handling HTTP DELETE requests.

Summarizing In this part we presented the technical outcomes of this dissertation, divided in three sections. In section 4, we gave an introduction of the Dycapo Project and the domain of interest of the architecture of the system. In section 5 we wrote about Dycapo Protocol, a REST-based protocol aiming to become a standard for the format and the transmission of data for dynamic carpooling services. In section 6 we presented Dycapo Server, a prototype system providing dynamic carpooling functionalities, implemented using Python programming language and demonstrating the use of the protocol. In the next part we will draw some conclusions and envision future work of the project.

Part IV
Conclusions and future work

In this dissertation we presented the outcomes of Dycapo, a project aimed at providing a better understanding and a technological solution to the open problem of dynamic carpooling. Through an analysis of the state of the art (reported in Part II), we identified the key issues as well as the functional and non-functional requirements in the domain of dynamic carpooling. Based on the comparative analysis of the open issues, we noticed that the basic building block was missing: an open and extendable technological infrastructure. Hence we decided to focus on two key components: a protocol for dynamic carpooling services, and a server able to handle calls from clients using the protocol, both of them presented in Part III. Dycapo Protocol is an open REST-based protocol that aims to become the missing standard for the format and the transmission of data related to dynamic carpooling services. Dycapo Server is a prototype software written using Python programming language that provides an essential set of dynamic carpooling functionalitites, using Dycapo Protocol. Its aim is to become a solid base or an inspiration for the creation of full dynamic carpooling services, run by companies and organizations.

The domain of interest, dynamic carpooling, is quite complex as it is possible to notice from the UML diagrams we created: the entities involved are complex and relate each other in non-trivial ways. In fact, there is very little empirical research on the topic, mainly proposals about how a system should be. There are also few working systems, with very few users and moreover not open source so that it is not possible to study and extend them. It was not easy to come up with a reasonable and consistent set of requirements and entities, then modeled by UML diagrams, but we are confident that Dycapo protocol is a useful resource for the community that can be easily extended and built on.

The aim to become a world standard is a high target but not impossible. We targeted this goal by choosing an open development process: the code was released as open source since the very first days and the evolving discussion about the state of the art and the proposed protocol was carried on an open wiki at www.dycapo.org since the beginning of the project. This strategy already started to pay back. In fact, we received many positive feedbacks from people related to the domain of interest. We have been contacted by the author of a paper analyzed, by many people involved in dynamic carpooling projects, some of them analyzed in the research phase and by local entrepreneurs. We have also received some proposals of joint ventures and protocol adoptions. There is also the possibility of a collaboration with MIT's Mobile Experience Laboratory, since there is a collaboration among FBK, where most of this work was done, and MIT. We have a first but empirical evidence about the quality of the protocol and how it is described: a University colleague is developing an Android client for Dycapo in the context of his bachelor thesis and he is able to figure out how the protocol should work mostly by looking at the public documentation.

Regarding future improvements, most of them are related to real world usage: we would like to test our system with real users. In this case, it would be very interesting to perform stress testing for understanding how the system can scale with many users and to extend some functionalities, for example a better handling of unforeseen situations. Many cases have already been covered by the testing framework we developed but obviously not all of them. Some possible inconsistencies in the protocol or in the server would be better spotted by real world usage and we really look forward for a deployment of Dycapo with real users. The develop of a feedback system would also be interesting: would users start to use it, it would be very important to give them the possibility to express how they felt about

the ride when it is finished. Moreover, there would be a requirement for the addition of Geographic Information Systems (GIS) and relative algorithms for the real-time matching of drivers and passengers.

Concluding, we believe the topic of this thesis is a recent and challenging one, still waiting for at least initial solutions and steps forwards. We are confident that our thesis was able to create some basic, open building blocks that others after us would be able to improve and extend with the common goal of solving an important problem for our world: too many cars on our streets with just one passenger in them.

A Appendix: Comparative Analysis Outcomes

Paper	Interface Design	Algorithms	Coordination
[11]	Give start, ending points and clear indications. Filter what information to reveal		
[8]	Provide lots of flexible settings to satisfy users.		Provide a static/dynamic approach, let users insert entries days before the start
[10]	Provide different levels of services: - simple: just destination and pickup - groups preferences (only women etc.) - scheduling of rides		
[1]			
[7]			Implement one-time registration process, simple. Provide RFID devices for drivers and passengers
[6]			
[5]	Focus on simplicity. Provide voice, speech recognition. Allow users to communicate each other.		Driving passenger away from the destination but near transportation locations (e.g. a bus station) increases quality of service and enhances coordination.
[4]		Given, built around social connections. Social network needed.	Built around social connection between users
[12]	Implement a simple registration system from mobile phone. In a second phase link social networks profiles, or manual fill. Develop a very simple UI		
[9]		Both data structures and Algorithms for matching are given	
[13]	Build it similar, simple and intuitive like Twitter. Use parameters like "where are you going?". Car position is essential: drivers should get a message and just confirm or refuse a ride		Use legal pick up points

Table 2: Paper Analysis: Interface Design, Algorithms, Coordination

Paper	Trustiness	Safety	Social Aspects
[11]		Authenticate before the match: password / PIN monitor arrival at destination Provide a feedback system a la EBay	Announce matching items in profiles before the ride Do research in social capital aspects
[8]		Create a PIN at registration phase to be used by the client	Add social networking support to help finding neighbours
[10]	Brand the idea: apply stickers on every car that participates. Give limitations to drivers: age limits, extra driving tests, check on criminal records etc.	Provide Auto Event Recorders on cars. Implement an emergency button on mobile phone, record GPS data. Provide a feedback system a la EBay	
[1]			
[7]	Record carpooling activity when cars pass through RFID readers	Build it around RFID, record lots of data and positions	
[6]	Involve community and governments in planning and implementation phases	Let the service be available only to registered users; Provide a Feedback system	Give the possibility to create social connections
[5]		Use RFID, GPS. Implement a complete rating system. Display vehicle and driver information before entering a vehicle. Display participants pictures. Assign random numbers for passenger pickups to confirm the ride. Provide voice and video features.	Match passengers socially. Link the application to social networks.
[4]	Use social networks to enhance it.		

Table 3: Paper Analysis: Trustiness, Safety, Social Aspects Pt. 1

Paper	Trustiness	Safety	Social Aspects
[12]	People are ready to spend 17% more time to pickup a friend of the social network rather than a stranger. Implement it.	Implement a rating system. Use and record GPS data. Do extra research in this field.	
[9]	Use social networks to enhance it.	Implement a GPS Help button. Record time, place, and sound. Develop a Feedback system	
[13]	Market against negative prejudgments of people: ride-sharing is associated with unreliability, problems with passengers and crime. Use the survey provided.	Solve problems related to reliability, politeness and customs. Make people change mind about the dangers of carpooling. Implement a rating with rate of confirmed trip-requests, rate of canceled trips and time accuracy. Use RFID, GPS, Blue-tooth, event recorders and everything else you could add. Measure and record the speed of vehicles.	

Table 4: Paper Analysis: Trustiness, Safety, Social Aspects Pt. 2

Paper	Critical Mass	Incentives	Suggestions
[11]			Provide a location-aware system Make use of mobile phones
[8]	Provide mass marketing before, during and after deployment. Search for start-up incentives	Search an institutional sponsor. Make the government provide parking spaces to participants	Implement both Web and mobile clients. Implement a static and a dynamic approach. Start with a many-to-one system: all at a single destination
[10]		Use a cooperative, public development of the system	Implement a Web interface and mobile clients (using phone calls)
[1]			
[7]		Make employers incentive employees. Involve Regional Transportation Boards	
[6]	Create an incremental service, starting from a thread of backwards compatible services (bus, taxi). Don't introduce new devices for the service, use mobile phones	Find a way to make the service a business case. Search for public incentives	Implement the system mobile only. Record GPS data. Provide a non-obtrusive system for authentication Research on quality of service measures
[5]	A multi-hop system will solve the problem, as more rides will be available, waiting times will decrease and quality will rise.	Convince governments to change laws to enforce carpooling	Use a dynamic, multi-hop, real- time mobile system to minimize waiting times, one hop at a time
[4]			Use mobile phones and sms. Use GPS. Use a provided high-level description of the system

Table 5: Paper Analysis: Critical Mass, Incentives, Suggestions Pt. 1

Paper	Critical Mass	Incentives	Suggestions
[12]	Involve users in some parts of development process. Research further on this topic.		Implement a mobile and a web system that interacts with social networks profiles. Use Opensocial and other social networks. Use our high level description of the whole system
[9]	Market-formation problem: discover a new, winning formula. Start with an existing service, like taxis. Find large employers. Serve events (i.e.. concerts)	Find money. Search for incentives from governments	Implement our Use Cases Provide our functional requirements. Provide our non-functional requirements
[13]	Convince the user to use the system for the first time. Pay the user directly for the first N times of use. Integrate public transportation system. Implement the system first for an existing taxi network, and then private cars could participate.	Provide a free taxi of public transport ride in case of no return possibility. Include existing carsharing projects in the system startup. Create an own currency system to be converted in real money. Convince governments to adapt the laws for dynamic carpooling	Implement a mobile, GPS, Multi-hop system

Table 6: Paper Analysis: Critical Mass, Incentives, Suggestions Pt.2

B Appendix: Dycapo Protocol

Protocol

Dycapo Protocol is an open protocol for sharing trip data among dynamic transit services. It is currently in the first stages of development.

It is an JSON RESTful protocol, heavily inspired by OpenTrip (http://opentrip.info/) Core protocol.

Contents

Introduction

Dycapo Protocol is an application-level protocol for enabling communication between Dynamic Carpooling servers and clients, using HTTP [RFC2616] and JSON [RFC4627].

Where to start

To better understand OpenTrip Dynamic, you should first read the OpenTrip Core (http://opentrip.info/wiki/OpenTrip_Core) specification, as we are inspiring from it. After that, we may summarize Dycapo Protocol as "OpenTrip entities extended and encoded in JSON". That is, we took all OpenTrip Core entities as described in the draft, created a convenient UML class diagram for developing Dycapo and extended the entities to suit our needs. That is how Dycapo Protocol is coming out.

Specification

Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 (http://tools.ietf.org/html/rfc2119) .

Each attribute requirement is specified when submitting objects to the system, not when returning them to the clients. Some attributes MAY be hidden when Resources are accessed because of privacy issues, e.g. the actual position of a user. This goes beyond the scopes of this Protocol and depends on the implementation of the server system.

JSON-Related Conventions

Dycapo Protocol Document formats are specified in terms of the JavaScript Object Notation [rfc4627].

Every data type is a JSON data type, as described in JSON specs (http://www.ietf.org/rfc/rfc4627.txt) .

Other Data-Type Conventions

The protocol makes use of GeoRSS Simple Point (http://georss.org/GeoRSS_Simple#Point) notation for expressing Geographical coordinates, encapsulated in a JSON string.

Date/Time objects are expressed using JSON string data type, using [ISO 8601] Convention. The following is the preferred format: "YYYY-MM-DD HH:MM:SS", as example: "2010-08-21 21:36:23"

Terminology

For convenience, this protocol can be referred to as the "Dycapo Protocol" or "DycapoP". The following terminology is used by this specification:

- URI - A Uniform Resource Identifier as defined in [RFC3986]. In this specification, the phrase "the URI of a document" is shorthand for "a URI which, when dereferenced, is expected to produce that document as a representation".
- IRI - An Internationalized Resource Identifier as defined in [RFC3987]. Before an IRI found in a document is used by HTTP, the IRI is first converted to a URI. See Section 4.1.
- Resource - A network-accessible data object or service identified by an IRI, as defined in [RFC2616]. See [REC-webarch] for further discussion on Resources.
- Object - An unordered collection of zero or more name/value pairs, where a name is a string and a value is a string, number, boolean, null, object, or array. Defined in [rfc4627]
- Array - An ordered sequence of zero or more values, as in [rfc4627]
- Driver - the role assumed by a user when he/she offers a Trip and drives a vehicle.
- Passenger - the role assumed by a user when he/she searches for a ride. A user which is not a Driver is automatically considered a Passenger.
- Trip - a single journey or course of travel taken as part of one's duty, work, etc. A Driver offers Trips. In DycapoP, a Trip is composed by some simple attributes described below plus a mode, a preferences, more than two locations.
- Location - a place of settlement, activity, or residence.
- Mode - a description about the mode of transportation being used by the Driver when performing a Trip.
- Preferences - a description about the preferences of a Driver when performing a Trip.
- Participation - the fact of taking part, as in some action or attempt, in a Trip. Both a Driver and Passengers participate in a Trip.

Protocol Model

DycapoP specifies operations for publishing, editing and deleting specific Resources using HTTP. It uses JSON-formatted representations to describe the state and metadata of those Resources.

Objects as inner Properties of other Objects

Some DycapoP objects can be obtained as Objects alone or be included in other objects.

As example, a Person object has a *location* attribute, that holds the current Person position.

When a DycapoP object is returned as an inner property of another object, just the *href* attribute MUST be included. All the other properties MAY be included.

The href attribute

Each Protocol object MUST include an attribute called **href** when returned as a resource. The type of this attribute is a string and its value MUST be the URL uniquely identifying the object.

For each Object a URL structure is proposed for sake of comprehension. It SHOULD NOT be followed as it is against REST principles.

The author attribute

Some Protocol objects have an attribute called **author**. This attribute specifies the Person that created the resource. The requirement of this attribute is always a MAY because its presence depends on the internal implementation of the security systems of the server implementing the protocol. Therefore, its value is usually filled in by the server after an authentication happened.

Initial URI

Dycapo Protocol attempts to be as more REST as possible. Therefore, it defines resources as hypertext driven. Clients SHOULD use an inital URI, here defined as **http://example.com/api". Each other access SHOULD be performed using the** href **attribute of each object.**

A GET request to the initial URI should return a list of accessible resources. As example:

```
{
    "searches": {
        "href": "http://example.com/api/searches/"
    },
    "persons": {
        "href": "http://example.com/api/persons/"
    },
    "trips": {
        "href": "http://example.com/api/trips/"
    }
}
```

Elements

Location

Represents a single location. See OpenTrip_Core#Location_Constructs (http://opentrip.info/wiki/OpenTrip_Core#Location_Constructs) for more info.

Attribute	Type	Requirement
label	string	MAY
street	string	MUST*
point	string	MUST
country	string	MAY
region	string	MAY
town	string	MUST*
postcode	number	MUST*
subregion	string	MAY
georss_point	string	MUST*
offset	number	SHOULD
recurs	string	MAY
days	string	MAY
leaves	string (see Dates)	MUST
href	string	MUST NOT

- Either **georss_point** OR all from set {**street**,**town**,**postcode**} MUST be specified
- **point** value MUST be any from the set {**orig**, **dest**, **wayp**, **posi**}.

See OpenTrip_Core#Attributes (http://opentrip.info/wiki/OpenTrip_Core#Attributes) for more info. **posi** is an extension and is for indicating that the Location represents the current position of a Person.

Data Representation Example

The following is a valid Location object:

```
{
                "town": "Bolzano",
                "point": "orig",
                "country": "",
                "region": "",
                "subregion": "",
                "days": "",
                "label": "Work",
                "street": "Rom Strasse",
                "postcode": 39100,
                "offset": 150,
                "leaves": "2010-09-02 13:32:34",
                "recurs": "",
```

```
            "georss_point": "46.490200 11.342294"
}
```

Operations

URL	http://example.com/persons/[username]/location/
Method	GET
Description	Returns a Person position

URL	http://example.com/persons/[username]/location/
Method	POST
Request Body	Location
Description	Updates (or creates) a Person's position

URL	http://example.com/persons/[username]/location/
Method	PUT
Request Body	Location
Description	Updates (or creates) a Person's position

URL	http://example.com/trips/[id]/locations/
Method	GET
Description	Returns the Locations involved in a Trip

URL	http://example.com/trips/[trip_id]/locations/[location_id]/
Method	GET
Description	Returns a Location involved in a Trip

Person

Represents a Person as described on OpenTrip_Core#Person_Constructs (http://opentrip.info/wiki/OpenTrip_Core#Person_Constructs)

Attribute	Type	Requirement
username	string	MUST
email	string	MUST
first_name	string	SHOULD
last_name	string	SHOULD
uri	string	MAY
phone	string	SHOULD
location	object (Location)	MUST NOT
age	number	SHOULD
gender	string	SHOULD
smoker	boolean	MAY
blind	boolean	SHOULD
deaf	boolean	SHOULD
dog	boolean	SHOULD
href	string	MUST NOT

Data Representation Example

The following is a valid Dycapo Protocol Person:

```
{
    "username": "driver1",
    "gender": "M",
    "phone": "123456",
    "email": "driver@drivers.com"
}
```

Operations

URL	http://example.com/persons/
Method	GET
Description	Retrieves a collection of Persons

URL	http://example.com/persons/
Method	POST
Request Body	Person
Description	Creates a Person resource

URL	http://example.com/persons/[username]/
Method	GET
Description	Retrieves a Person

URL	http://example.com/persons/[username]/
Method	PUT
Request Body	Person
Description	Updates a Person object

Modality

Represents additional information about the mode of transportation being used. See OpenTrip_Core#Mode_Constructs (http://opentrip.info/wiki/OpenTrip_Core#Mode_Constructs) for more info.

Attribute	Type	Requirement
kind	string	MUST
capacity	number	MUST
vacancy	number	MUST
make	string	MUST
model_name	string	MUST
year	string	MAY
color	string	SHOULD
lic	string	SHOULD
cost	number	SHOULD
href	string	MUST NOT

- Please use as **capacity** the total capacity of your car MINUS the driver. E.G. If a car has a capacity of 5 seats, use 4 as value for capacity.

Data Representation Example

The following is a valid DycapoP Modality object:

```
{
            "kind": "auto",
            "capacity": 4,
            "lic": "",
            "color": "",
            "make": "Ford",
            "vacancy": 4,
            "cost": 0.0,
            "year": 0,
            "model_name": "Fiesta",
}
```

Operations

URL	http://example.com/trips/[trip_id]/modality/
Method	GET
Description	Returns the Modality object of that Trip

Preferences

Stores the preferences of a Trip set by the Person who creates it. See OpenTrip_Core#Preference_Constructs (http://opentrip.info/wiki/OpenTrip_Core#Preference_Constructs) for more info. We kept drive and ride attributes just for compatibility reasons: in OpenTrip Dynamic just a driver should be the author of a Trip.

Attribute	Type	Requirement
age	string	MAY
nonsmoking	boolean	MAY
gender	string	MAY
drive	boolean	MAY
ride	boolean	MAY
href	string	MUST NOT

- Even if all attributes of Prefs objects are optional, objects of type Prefs MUST be provided when doing an operation that involves this object. In case of zero attributes provided, an empty object MUST be provided
- **gender** MUST be any of the values {'M', 'F', 'B'}, meaning 'male', 'female', 'both'

Data Representation Example

The following is a valid DycapoP Prefs object:

```
{
            "ride": false,
            "gender": "",
            "age": "18-30",
            "drive": false,
            "nonsmoking": false
```

```
}
```

Operations

URL	http://example.com/trips/[trip_id]/preferences/
Method	GET
Description	Returns the Preferences of the Trip

Trip

Represents a Trip. See OpenTrip_Core#Entry_Elements (http://opentrip.info/wiki/OpenTrip_Core#Entry_Elements) for more info.

Attribute	Type	Requirement
published	string (Date)	MUST NOT
active	boolean	MUST
updated	string (Date)	MUST NOT
expires	string (Date)	MUST
author	object (Person)	MAY
locations	array (Location)	MUST
mode	object (Mode)	MUST
preferences	object (Preferences)	MUST
href	string	MUST NOT
participations	array (Participation)	MUST NOT

Data Representation Example

The following is a complete DycapoP Trip object, containing the other Entities used as example in the rest of the document

```
{
    "preferences": {
        "nonsmoking": false,
        "gender": "",
        "ride": false,
        "drive": false,
        "age": "18-30"
    },
    "expires": "2010-09-05 13:33:08",
    "locations": [
        {
            "town": "Bolzano",
            "point": "orig",
            "country": "",
            "region": "",
            "subregion": "",
            "days": "",
            "label": "Work",
            "street": "Rom Strasse",
            "postcode": 39100,
            "offset": 150,
            "leaves": "2010-09-02 13:32:34",
            "recurs": "",
            "georss_point": "46.490200 11.342294"
        },
        {
            "town": "Bolzano",
            "point": "dest",
            "country": "",
            "region": "",
            "subregion": "",
            "days": "",
            "label": "Work",
            "street": "Piazza della Vittoria, 1",
            "postcode": 39100,
            "offset": 150,
            "leaves": "2010-09-02 13:32:34",
            "recurs": "",
            "georss_point": "46.500740 11.345073"
        }
    ],
    "modality": {
        "kind": "auto",
        "capacity": 4,
        "lic": "",
        "color": "",
        "make": "Ford",
        "id": 4,
        "vacancy": 4,
        "cost": 0.0,
        "year": 0,
        "model_name": "Fiesta"
    }
}
```

Operations

URL	http://example.com/trips/
Method	GET
Description	Retrieves a collection of Trips

URL	http://example.com/trips/
Method	POST
Request Body	Trip
Description	Creates a Trip object

URL	http://example.com/trips/[trip_id]/
Method	GET
Request Body	Person
Description	Returns a Trip Resource

URL	http://example.com/trips/[trip_id]/
Method	PUT
Request Body	Trip
Description	Updates a Trip object

URL	http://example.com/trips/[trip_id]/
Method	DELETE
Request Body	Trip
Description	Deletes a Trip object

Participation

Represents a Participation in a Trip.

Attribute	Type	Requirement
author	object (Person)	MAY
status	string	MUST
href	string	MUST

- status attribute value MUST be from the set {"**request**","**accept**","**start**","**finish**"} and represents the current Participation status of a user

Data Representation Example

The following is a valid Dycapo Protocol Participation:

```
{
        "status": "accept",
        "person": {
            "username": "driver1",
            "href": "http://example.com/api/persons/rider1/",
            "location": {
                "href": "http://example.com/api/persons/rider1/location/"
            }
        },
        "href": "http://example.com/api/trips/4/participations/rider1/"
    }
```

Operations

URL	http://example.com/trips/[trip_id]/participations/
Method	GET
Description	Retrieves the Participations of the Trip

URL	http://example.com/trips/[trip_id]/participations/
Method	POST
Request Body	Participation
Description	Creates a Participation object related to the Trip (Requesting a Ride)

URL	http://example.com/trips/[trip_id]/participations/[username]/
Method	PUT
Request Body	Person
Description	Updates a Participation object related to the Trip (Accepting a Ride request, Starting a Ride, Finishing a Ride)

URL	http://example.com/trips/[trip_id]/participations/[username]/
Method	DELETE
Description	Deletes a Participation resource (Refusing a Ride request, Deleting a Ride request).

Search

This resource represents a search between Trips. Due to the complexity of the objects involved when searching a Trip, there is the necessity of creating state-ful Search objects, accessible each time a search is needed.

Attribute	Type	Requirement
origin	object (Location)	MUST
destination	object (Location)	MUST
author	object (Person)	MAY
trips	array (Trip)	MUST NOT

Data Representation Example

The following is a valid Search object:

```
{
    "origin": {
        "town": "Bolzano",
        "point": "posi",
        "href": "http://example.com/api/searches/37/",
        "country": "",
        "region": "",
        "subregion": "",
        "days": "",
        "label": "Work",
        "street": "Drususallee, 43/a",
        "postcode": 39100,
        "offset": 150,
        "id": 140,
        "leaves": "2010-09-02 13:32:34",
        "recurs": "",
        "georss_point": "46.494957  11.340239"
    },
    "author": {
        "username": "rider1",
        "href": "http://example.com/api/persons/rider1/"
    },
    "destination": {
        "town": "Bolzano",
        "point": "dest",
        "href": "http://example.com/api/searches/37/",
        "country": "",
        "region": "",
        "subregion": "",
        "days": "",
        "label": "Work",
        "street": "Piazza della Vittoria, 1",
        "postcode": 39100,
        "offset": 150,
        "id": 141,
        "leaves": "2010-09-02 13:32:34",
        "recurs": "",
        "georss_point": "46.500891  11.344306"
    }
}
```

Operations

URL	http://example.com/searches/
Method	POST
Request Body	Search
Description	Creates a Search object

URL	http://example.com/searches/[search_id]/
Method	GET
Description	Retrieves the Search object and the results, if any

Use of HTTP Response Codes

The Dycapo Protocol uses the response status codes defined in HTTP to indicate the success or failure of an operation. Consult the HTTP specification [RFC2616] for detailed definitions of each status code. In detail, DycapoP makes use of the following codes:

Code	Name	Description
200	OK	Denotes a successful operation, an entity containing additional information SHOULD be provided
201	Created	Denotes the creation of a resource. A representation of the Resource SHOULD be provided
204	No Content	Denotes the deletion of a resource. A representation of the Resource SHOULD NOT be provided
401	Unauthorized	The client provided wrong (or did not provide) credentials
403	Forbidden	The client does not have the rights for perform the request
404	Not Found	No Resource has been found at the given URI
415	Unsupported Media Type	A Protocol Error Occurred. A description of the error SHOULD be provided

License

References

[1] Abdel-Naby, S., Fante, S.: Auctions negotiation for mobile rideshare service. In Proc. IEEE Second International Conference on Pervasive Computing and Applications (2007)

[2] Davis, A. Operational prototyping: a new development approach. Software, IEEE 9, 5 (1992)

[3] Fielding, R. T. Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine (2000)

[3] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. RFC2616: hypertext transfer Protocol-HTTP/1.1. RFC Editor United States (1999).

[4] Gidófalvi, G. et al. Instant Social Ride-Sharing. In Proc. 15th World Congress on Intelligent Transport Systems, p 8, Intelligent Transportation Society of America (2008)

[5] Gruebele, P.A., Interactive System for Real Time Dynamic Multi-hop Carpooling. Global Transport Knowledge Partnership (2008).

[6] Hartwig, S., Buchmann, M.: Empty seats traveling: Next-generation ridesharing and its potential to mitigate traffic and emission problems in the 21st century. Technical report, Nokia (2007), http://research.nokia.com/tr/NRC-TR-2007-003.pdf

[7] Kelley, K. L. Casual Carpooling-Enhanced. Journal of Public Transportation 10, 4 (2007), 119.

[8] Kirshner, D. Pilot Tests of Dynamic Ridesharing. Technical report (2006), http://www.ridenow.org/ridenow_summary.html

[9] Morris, J. et. al. SafeRide: Reducing Single Occupancy Vehicles. Technical report (2008), http://www.cs.cmu.edu/~jhm/SafeRide.pdf

[10] Murphy, P.: The smart jitney: Rapid, realistic transport. New Solutions Journal (4) (2007)

[11] Resnick, P. SocioTechnical support for ride sharing. In Working Notes of the Symposium on Crossing Disciplinary Boundaries in the Urban and Regional Contex (UTEP-03) (2006).

[12] Wessels, R. Combining Ridesharing & Social Networks. Technical report (2009), http://www.aida.utwente.nl/education/ITS2-RW-Pooll.pdf

[13] Zimmermann, H., and Stempfel, Y. Current Trends in Dynamic Ridesharing, identification of Bottleneck Problems and Propositions of Solutions. Indian Institute of Technology Delhi, India. (2008), http://dynamicridesharing.org/~dynami11/wiki/images/9/95/IIT_Delhi_-_Dynamic_Ridesharing.PDF

Dedicated to My Family.