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
Thesis Advisor: Paolo Massa, Ph.D.

October 2010

Abstract

Carpooling occurs when a driver share his/her private car with one or more passengers. The benefits of carpooling, also called ridesharing, are environmental, economical and social. Dynamic Carpooling is a specific type of Carpooling which allows drivers and passengers to find suitable lifts close to their desired departure time and directly on streets. This dissertation describes Dycapo, an open-source system to provide Dynamic Carpooling services. After a review of the state of the art, the two main "components" are described, namely the protocol and the server architecture. Dycapo Protocol is an open REST protocol for sharing trip information among dynamic transit services, taking inspiration from OpenTrip, a previously proposed protocol. Dycapo Server is a prototype providing a Web Service 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

Kurzfassung

Contents

List of Figures

List of Tables

Part I
Introduction

Using a private car is a transportation system very common in industrialized countries. Between year 2004 and 2009, the worldwide production of new private vehicles has been of 295 millions of 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 an efficient and wasteful way of transportation. 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 [Hartwig, S. and Buchmann, M. (2007)]. 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 to all these problems is carpooling, i.e. the act of sharing a trip on a private vehicle between one or more other passengers. Carpooling helps the environment by allowing to use oil wisely and 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. It gained government attention in the U.S.A. but unfortunately not in EU. 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 are usually carried out by e-mails of private messages in the web-site. Therefore, we may see carpooling as a static way of sharing a trip.

The availability of 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

[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]

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

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

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 parts. 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 Published Papers

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

1.1 Sociotechnical support for Ride Sharing

[Resnick, P. (2003)]

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 instant ridesharing is 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.

1.2 Pilot Tests of Dynamic Ridesharing

[Kirshner, D. (2006)]

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.

1.3 The smart Jitney: Rapid, Realistic Transport

[Murphy, P. (2007)]

The paper focuses on environmental benefits of dynamic carpooling. It asserts that dynamic car sharing 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 complaints that challenges are all focused in convincing the population to use the service, proposing a cooperative public development of the system.

1.4 Auction negotiation for mobile Rideshare service

[Abdel-Naby, S. et al. (2007)]

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

1.5 Casual Carpooling enhanced

[Kelley, L.K. (2007)]

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.

1.6 Empty seats travelling

[Hartwig, S. and Buchmann, M. (2007)]

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.

1.7 Interactive systems for real time dynamic multi hop carpooling

[Gruebele (2008)]

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 complaints 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.

1.8 Instant Social Ride Sharing

[Gidófalvi, G. et al. (2008)]

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.

1.9 Combining Ridesharing & Social Networks

[Wessels, R. (2009)]

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.

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

1.10 SafeRide: Reducing Single Occupancy Vehicles

[Morris, J. (2009)]

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 complaints that someone must discover a winning formula before anyone will invest.

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

1.11 Current Trends in Dynamic Ridesharing, identification of Bottleneck Problems and Propositions of Solutions

[Zimmerman, H. and Stempfel, Y. (2009)]

Reviews some papers about Dynamic Carpooling. Also reviews some applications. Identifies barriers and proposes solutions.

2 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. 5th2010. Each text enclosed in double quotes is cited verbatim from the website of the application.

2.1 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.

2.2 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, 10th 2009 the service offered 5310 empty seats.

2.3 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.

2.4 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.

2.5 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”

2.6 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.

2.7 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.

2.8 GoLoco

http://goloco.org/

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

2.9 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.

2.10 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.

2.11 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.

2.12 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.

2.13 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 [Zimmerman, H. and Stempfel, Y. (20
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

In Table 1,2 and 3, for each category (columns) we list the suggestions and interesting points made in the different research papers (rows). Table 1,2,3 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.

Paper	Interface Design	Algorithms	Coordination
[Resnick, P. (2003)]	Give start, ending points and clear indications. Filter what information to reveal		
[Kirshner, D. (2006)]	Provide lots of flexible settings to satisfy users.		Provide a static/dynamic approach, let users insert entries days before the start
[Murphy, P. (2007)]	Provide different levels of services: - simple: just destination and pickup - groups preferences (only women etc.) - scheduling of rides		
[Abdel-Naby, S. et al. (2007)]			
[Kelley, L.K. (2007)]			Implement one-time registration process, simple. Provide RFID devices for drivers and passengers
[Hartwig, S. and Buchmann, M. (2007)]			
[Gruebele (2008)]	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.
[Gidófalvi, G. et al. (2008)]		Given, built around social connections. Social network needed.	Built around social connection between users
[Wessels, R. (2009)]	Implement a simple registration system from mobile phone. In a second phase link social networks profiles, or manual fill. Develope a very simple UI		
[Morris, J. (2009)]		Both data structures and Algorithms for matching are given	
[Zimmerman, H. and Stempfel, Y. (2009)]	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 Grid Outcomes: Interface Design, Algorithms, Coordination, Trustiness

Paper	Trustiness	Safety	Social Aspects
[Resnick, P. (2003)]		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
[Kirshner, D. (2006)]		Create a PIN at registration phase to be used by the client	Add social networking support to help finding neighbours
[Murphy, P. (2007)]	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	
[Abdel-Naby, S. et al. (2007)]			
[Kelley, L.K. (2007)]	Record carpooling activity when cars pass through RFID readers	Build it around RFID, record lots of data and positions	
[Hartwig, S. and Buchmann, M. (2007)]	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
[Gruebele (2008)]		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.
[Gidófalvi, G. et al. (2008)]	Use social networks to enhance it.		
[Wessels, R. (2009)]	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.	
[Morris, J. (2009)]	Use social networks	Implement a GPS	

Paper	Critical Mass	Incentives	Suggestions
[Resnick, P. (2003)]			Provide a location-aware system Make use of mobile phones
[Kirshner, D. (2006)]	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 approaches. Start with a many-to-one system: all at a single destination
[Murphy, P. (2007)]		Use a cooperative, public development of the system	Implement a Web interface and mobile clients (using phone calls)
[Abdel-Naby, S. et al. (2007)]			
[Kelley, L.K. (2007)]		Make employers incentive employees. Involve Regional Transportation Boards	
[Hartwig, S. and Buchmann, M. (2007)]	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
[Gruebele (2008)]	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
[Gidófalvi, G. et al. (2008)]			Use mobile phones and sms. Use GPS. Use a provided high-level description of the system
[Wessels, R. (2009)]	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

Our rationalization of dynamic carpooling issues and possible solutions, summarized in Table 1,2,3 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. 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. Therefore, every project started from zero, "reinventing the wheel".

Based on this analysis, we decided to create to 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 providing 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 Introduction

Dycapo is the name given to a project aiming to share knowledge outcomes and technological products on dynamic carpooling. The information are 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.

For better introducing the protocol and the server prototype, we now present come user stories on system functionalities.

4.1 Terminology

In Table 4 we explain some nouns used in both the protocol and the server for defining entities operating in the system. The remaining concepts will be explained in the section regarding the protcol.

Term	Definition
Person	A member of the service, i.e. a user of the system
Trip	A single journey or course of travel taken as part of one's duty, work, etc.
Driver	The role assumed by a Person when he/she offers to share some seats when driving
Passenger	The role assumed by a Person when he/she participates to 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.
Location	A geographical place of settlement, activity, or residence.

Table 7: Key Terms of Dycapo

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 4.

4.2.1 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 smart-phone with GPS activated, opens his Dycapo client and gives the desired destination , therefore searching a trip. Angela (a Person) is travelling with her car along Corso Libertà, Bolzano (a Location). Her destination is Laives, Italy, southern to the cimitery (a Location). 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 Rider) and Angela meet and share the trip, dividing the costs.

4.2.2 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 smart-phone running a Dycapo client and inputs her travel intentions. 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 Cremona, 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 Architecture

Dycapo system follows the client-server model, in which a server receives messages from clients (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 and a Passenger) and returns the results using the protocol. The openness of the protocol and the server prototype will ensure multiple servers running in different areas and contexts, and are thought for future server-to-server communication.

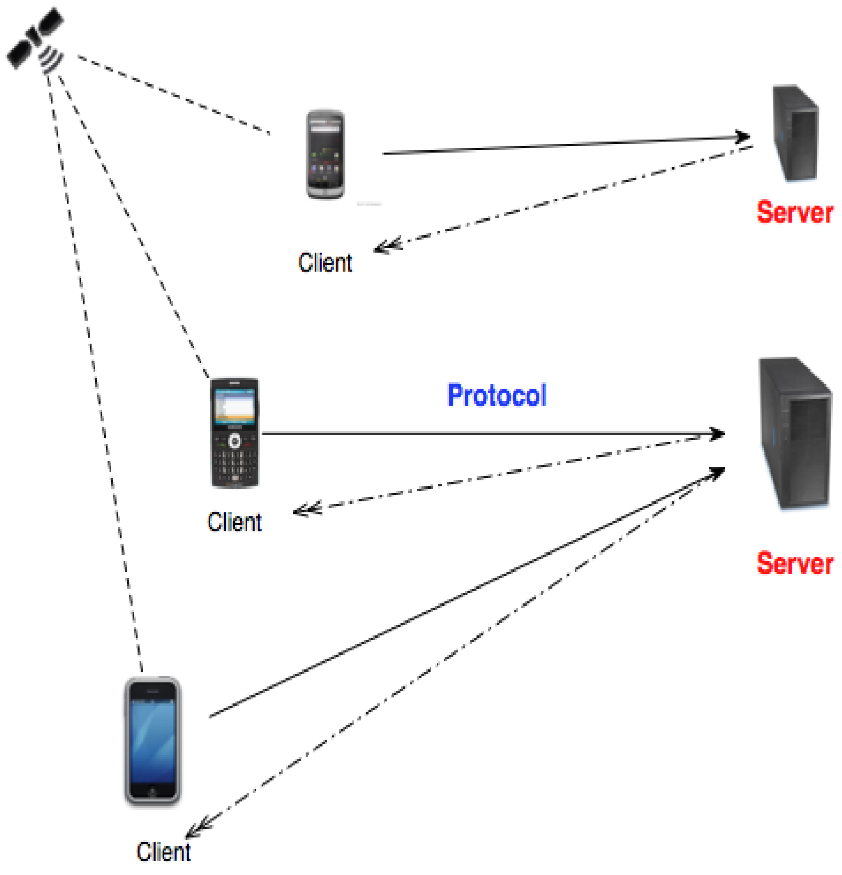

Figure 1: High level view of Dycapo components architecture

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 are Paul and Angela in user story “A simple trip”) and a Location (as it is “Via Roma, Bolzano, Italy” in user story “A simple trip”). *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 user story “A simple trip” 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 19, 2010is included in the Appendix A. Each element has a *href* attribute, that holds the unique URL for doing operations on the resource. It will be further explained in the operations sub-section.

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].

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

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

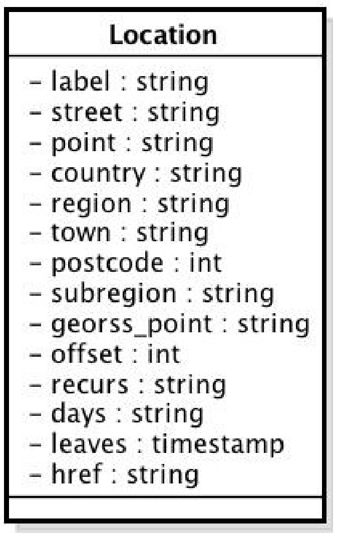

Figure 2: Location Element

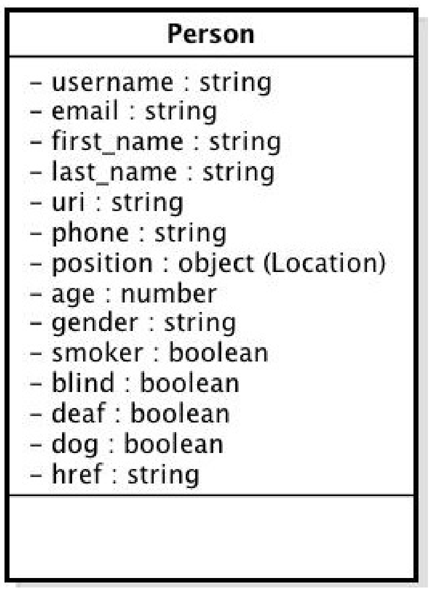

Figure 3: Person Element

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 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.

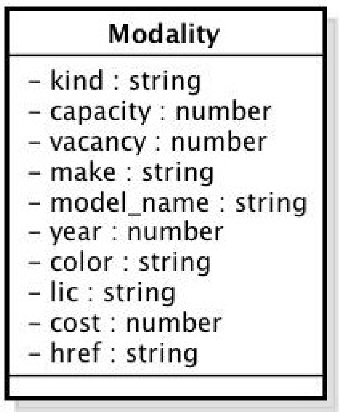

Figure 4: Modality Element

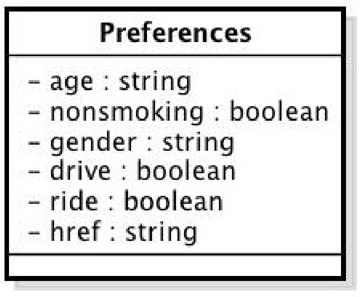

Figure 5: Preferences Element

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.

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.

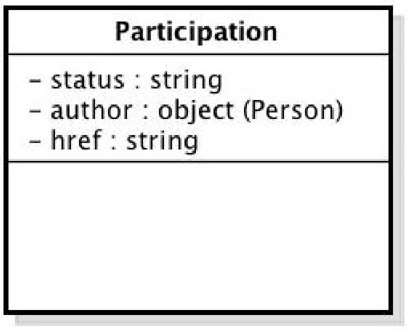

Figure 6: Participation Element

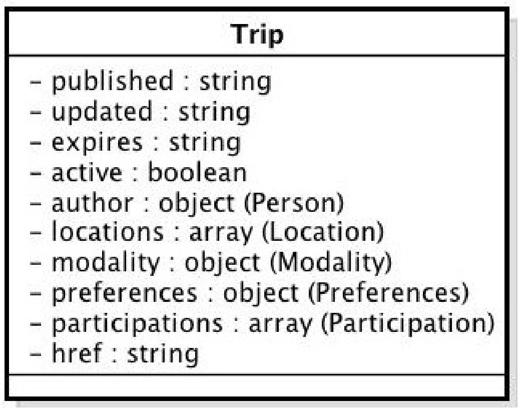
Trip
- published : string
- updated : string
- expires : string
- active : boolean
- author : object (Person)
- locations : array (Location)
- modality : object (Modality)
- preferences : object (Preferences)
- participations : array (Participation)
- href : string

Figure 7: Trip Element

In Figure 8 we represent the possible states of a Participation, using a UML State Diagram. For each transition, we report the value of attribute status and the HTTP operation performed by the operation. We also report those transitions that only the author of the Trip (i.e. the driver) can cause.

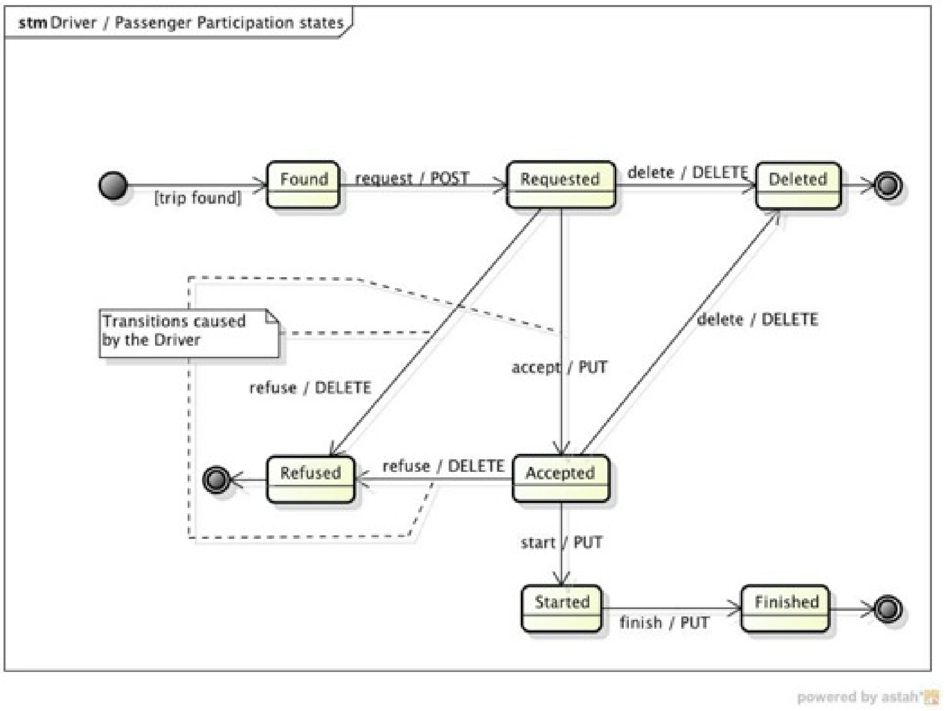

Figure 8: Participation Status

Search A query to the server when a passenger searches an available trip is represented by a Search element (Figure 9).

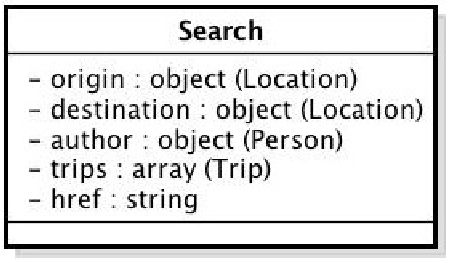
Search
- origin : object (Location)
- destination : object (Location)
- author : object (Person)
- trips : array (Trip)
- href : string

Figure 9: Search Element

Attributes of elements can be mandatory or optional. The specific settings for each entity are described in the complete protocol at Appendix 1.

5.2 Operations

DycapoP is implemented using a resource oriented architecture, i.e. the definition of a full REST [Fielding, R.T. (2000)] 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 the list of all the stored trips, a client must perform the following HTTP request:

```
GET /api/trips/ HTTP/1.1
Authorization: Basic cmlkZXIxOnBhc3N3b3Jk
Host: 127.0.0.1
```

It is self-descriptive: a GET request of the collection of trips 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://127.0.0.1/api/trips/3/participations/" },
"preferences": {

    "nonsmoking": false,
    "gender": "",
    "ride": false,
    "drive": false,
    "href": "http://127.0.0.1/api/trips/4/preferences/",
    "age": "18-30"

},
"author": {

    "username": "driver1",
    "gender": "M",
    "href": "http://127.0.0.1/api/persons/driver1/"
```

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

```
},
"expires": "2010-09-05 16:03:05",
"locations": [ {

    "town": "Bolzano",
    "point": "orig",
    "href": "http://127.0.0.1/api/trips/3/locations/",
    "country": "",
    "region": "",
    "subregion": "",
    "days": "",
    "label": "Work",
    "street": "Rom Strasse",
    "postcode": 39100,
    "offset": 150,
    "leaves": "2010-09-02 16:02:22",
    "recurs": "",
    "georss_point": "46.490200 11.342294"

}, {

    "town": "Bolzano",
    "point": "dest",
    "href": "http://127.0.0.1/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://127.0.0.1/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://127.0.0.1/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 an array of Trip objects, enclosed in square brackets. It currently contains just one 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 protocol operations are available in Appendix A. This dissertation focuses on the implementation of the functionalities in Part IV.

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 6, 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 (Figure 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.

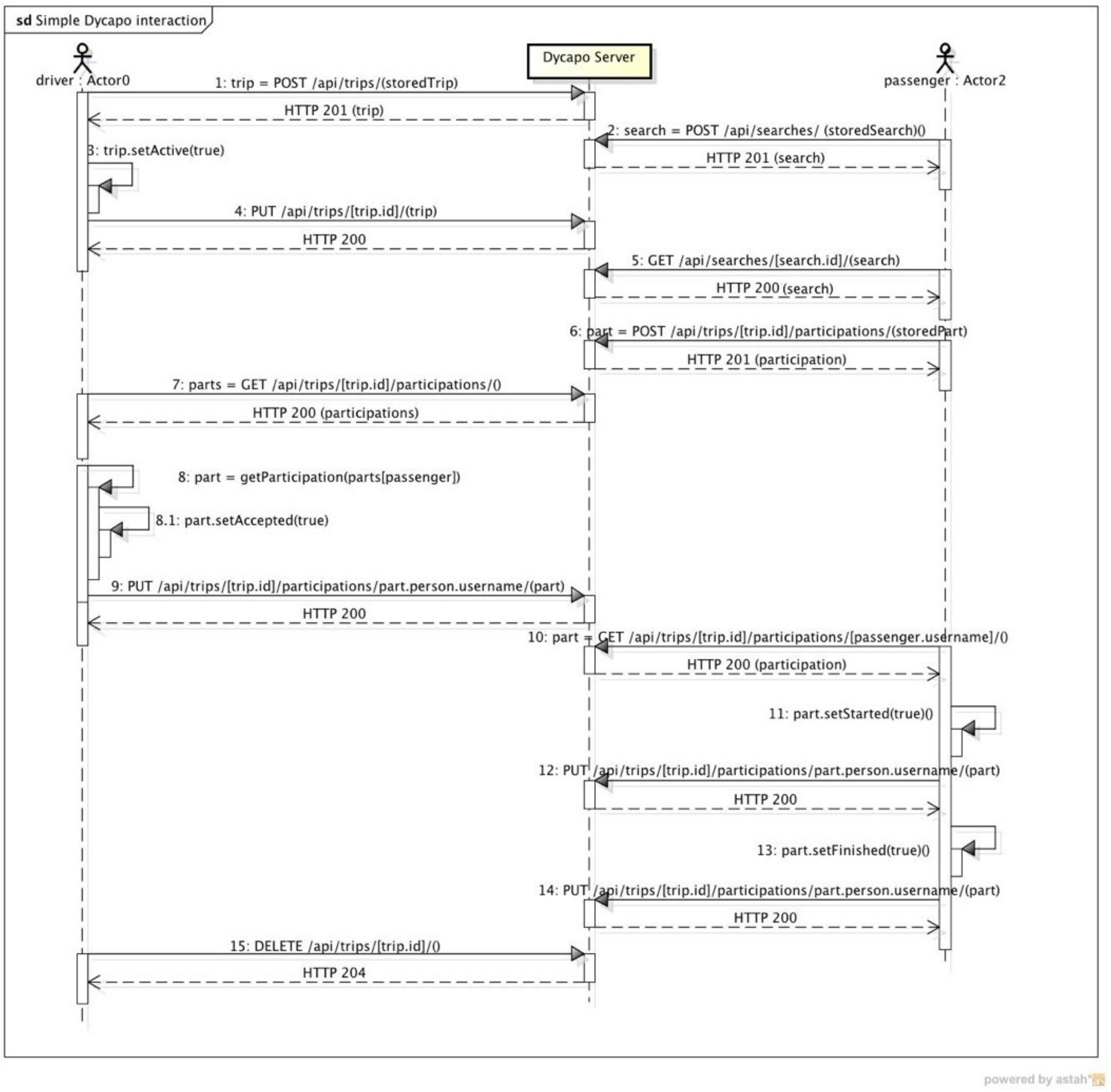
sd Simple Dycapo interaction
driver : Actor0
Dycapo Server
passenger : Actor2
1: trip = POST /api/trips/(storedTrip)
HTTP 201 (trip)
2: search = POST /api/searches/ (storedSearch)()
3: trip.setActive(true)
HTTP 201 (search)
4: PUT /api/trips/[trip.id]/(trip)
HTTP 200
5: GET /api/searches/[search.id]/(search)
HTTP 200 (search)
6: part = POST /api/trips/[trip.id]/participations/(storedPart)
HTTP 201 (participation)
7: parts = GET /api/trips/[trip.id]/participations/()
HTTP 200 (participations)
8: part = getParticipation(parts[passenger])
8.1: part.setAccepted(true)
9: PUT /api/trips/[trip.id]/participations/part.person.username/(part)
HTTP 200
10: part = GET /api/trips/[trip.id]/participations/[passenger.username]/()
HTTP 200 (participation)
11: part.setStarted(true)()
12: PUT /api/trips/[trip.id]/participations/part.person.username/(part)
HTTP 200
13: part.setFinished(true)()
14: PUT /api/trips/[trip.id]/participations/part.person.username/(part)
HTTP 200
15: DELETE /api/trips/[trip.id]/()
HTTP 204
powered by astah*

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 throw-away then evolutionary prototyping - as defined by [Davis, A.M. (1992)] 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 19, 2010counts 177 commits, 14 git tags and 5 branches.

6.1 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 RESTful principles.

- 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

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

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.

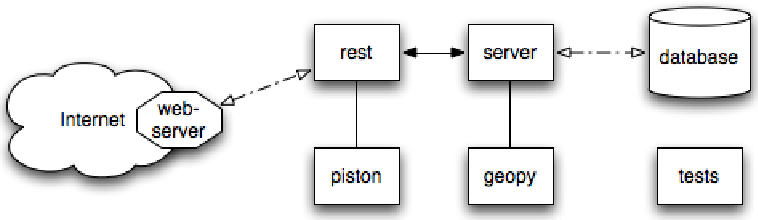

Figure 11: Dycapo Server components

6.2 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. 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, 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, a Django add-on that provides the creation of RESTful APIs.
- Py.test, a mature unit-testing framework for Python programming language
- Apache webserver and PostgreSQL database management system.

6.3 Models

Dycapo Server models are implemented as Python classes and stored in packages in the models sub-package inside the server package.

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.

6.4.1 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 permissions of both users given, 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

6.4.2 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.

6.4.3 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

6.4.4 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, rider)
```

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.

6.4.5 Utils

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

Part IV
Conclusions

A Appendix: Dycapo Protocol

References

[Abdel-Naby, S. et al. (2007)]	Auctions Negotiation for Mobile Rideshare Service
[Davis, A.M. (1992)]	Operational prototyping: a new development approach
[Fielding, R.T. (2000)]	Architectural Styles and the Design of Network-based Software Architectures
[Fielding, R.T. et al. (1999)]	Hypertext Transfer Protocol – HTTP/1.1
[Gidófalvi, G. et al. (2008)]	Instant Social Ride Sharing
[Gruebele (2008)]	Interactive System for Real Time Dynamic Multi-hop Carpooling
[Hartwig, S. and Buchmann, M. (2007)]	Empty Seats Traveling
[Kelley, L.K. (2007)]	Casual carpooling—enhanced
[Kirshner, D. (2006)]	Pilot Tests of Dynamic Ridesharing
[Morris, J. (2009)]	SafeRide: Reducing Single Occupancy Vehicles
[Murphy, P. (2007)]	The Smart Jitney: Rapid, Realistic Transport
[Resnick, P. (2003)]	SocioTechnical Support for Ride Sharing
[Wessels, R. (2009)]	Combining Ridesharing and Social Networks
[Zimmerman, H. and Stempfel, Y. (2009)]	Current Trends in Dynamic Ridesharing, identification of Bottleneck Problems and Propositions of Solutions

References

[1] SafeRide: reducing single occupancy vehicles.

[2] Davis, A. Operational prototyping: a new development approach. *Software, IEEE 9*, 5 (1992), 70–78.

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

[5] Gidófalvi, G., Herényi, G., and Pedersen, T. B. INSTANT SOCIAL RIDE-SHARING.

[6] Hartwig, S., and Buchmann, M. Empty seats traveling,". *Nokia Research Center Bochum* (2007).

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

[8] 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).*

[9] Zimmermann, H., and Stempfel, Y. Current trends in dynamic ridesharing, identification of bottleneck problems and propositions of solutions.