A Mobile Phone Solution for Ad-hoc Hitch-Hiking in South Africa

Submitted in fulfilment
of the requirements of the degree of

Master of Science

of Rhodes University

Sacha P. Miteche

Grahamstown, South Africa
October 2013

Abstract

The purpose of this study was to investigate the use of mobile phones in organizing ad-hoc vehicle ridesharing based on hitch-hiking trips involving private car drivers and commuters in South Africa. A study was conducted to learn how hitch-hiking trips are arranged in the urban and rural areas of the Eastern Cape. This involved carrying out interviews with hitch-hikers and participating in several trips. The study results provided the design specifications for a Dynamic Ridesharing System (DRS) tailor-made to the hitch-hiking culture of this context. The design of the DRS considered the delivery of the ad-hoc ridesharing service to the anticipated mobile phones owned by people who use hitch-hiking. The implementation of the system used the available open source solutions and guidelines under the Siyakhula Living Lab project, which promotes the use of Information and Communication Technology (ICT) in marginalized communities of South Africa. The developed prototype was tested in both the simulated and live environments, then followed by usability tests to establish the viability of the system. The results from the tests indicate an initial breakthrough in the process of modernizing the ad-hoc ridesharing of hitch-hiking which is used by a section of people in the urban and rural areas of South Africa.

Acknowledgements

I would like to thank my supervisors, Alfredo Terzoli and Hannah Thinyane, for their efforts in providing endless support to complete this thesis. I am grateful to your guidance and encouragement in the years I have worked under your supervision.

To the members of the Convergence Research Group in the Computer Science Department and ReedHouse Systems, I appreciate the technical support and advice that you gave me and wish you all the best in your research work.

I would like to acknowledge the financial support of Telkom SA, Tellabs, Genband, Easttel, Bright Ideas 39 and THRIP. I would also like to thank the Malawi Government for the financial assistance to study at Rhodes University. This opportunity has given me tremendous exposure to various fields of computer science.

Lastly, I would like to thank my wife and family for the support throughout this long journey that was worthy going through, may the Almighty Lord bless you. All this would not have been possible without your continued prayers and encouragement.

Related Publications

The work that appears in this thesis has been presented in the following conference papers:

1. Sacha Miteche, Alfredo Terzoli, and Hannah Thinyane. A Mobile Phone Solution to Improve Geographical Mobility. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, 2011.

2. Sacha Miteche, Alfredo Terzoli, and Hannah Thinyane. A Mobile Phone Solution to Improve Geographical Mobility. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, 2012.

Contents

List of Figures

List of Tables

Glossary of Terms

API	Application Programming Interface
DRS	Dynamic Ridesharing System
FOSS	Free and Open Source Solution
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile
MMS	Multimedia Messaging Service
OSGi	Open Services Gateway Initiative
REST	Representational State Transfer
RHS	ReedHouse Systems
SLL	Siyakhula Living Lab
SMS	Short Message Service
SMSC	Short Message Service Center
URI	Universal Resource Identifier
URL	Universal Resource Locator
WAP	Wireless Application Protocol
XML	eXtensible Markup Language

Chapter 1

Introduction

This chapter introduces the research presented in this thesis by explaining the problem statement. This is followed by the objective of the research to solve the identified problems. Thereafter, the scope and limitations of the work in this thesis are presented together with the approach that was followed to achieve the goals of the research. The chapter concludes with an outline of the chapters that constitute this thesis.

1.1 Problem Statement

Vehicle ridesharing is used as an alternative to other means of collective land transportation methods, such as public buses and trains [61]. Ad-hoc ridesharing is one example of vehicle ridesharing which normally involves the use of privately owned vehicles in trips that are organized without any pre-arrangements or acquaintances between the driver and commuters. Hitch-hiking travels are an example of the ad-hoc vehicle ridesharing method.

Vehicle ridesharing systems have existed to assist in organizing ridesharing and realize several incentives such as reduced travel costs through cost sharing, minimized traffic congestion and assist in the availability of parking spaces. The ridesharing systems have advanced their capabilities in response to the advancement of technology such that the modern systems target mobile phone users to help arrange ad-hoc ridesharing whilst on the move. These ridesharing systems provide services that ease the ridesharing process for people without acquaintances, using features such as Global Positioning Satelite (GPS) and map data to facilitate the locating of rideshare partners. Some of the ridesharing systems for mobile phone users are available for use at no charge [9, 76].

In South Africa, ad-hoc ridesharing through hitch-hiking is popular in some areas as

observed in the Eastern Cape [94]. The available ridesharing systems are capable of being used in these ad-hoc ridesharings based on their technical capabilities. However, the majority of people in South Africa who use hitch-hiking as a means of travels cannot utilize them due to the following reasons:

1. The ridesharing systems target smartphones which the majority of mobile phone users do not own/have. Therefore, the ridesharing services are out of reach for the people who are expected to benefit from their operations.

2. The systems are designed with a particular context of use and user characteristics in mind, as is the case with all software [60]. The hitch-hiking culture used in their design is different from the one practiced in South Africa. Therefore, the designs of the available ridesharing systems are not adequate to satisfy the user requirements for hitch-hiking in South Africa.

1.2 Research Objective

This research investigates the possibility of using mobile phones, with different capabilities, to arrange ad-hoc ridesharing trips for hitch-hikers in the urban and rural areas of South Africa. The focus is on the ad-hoc ridesharing that involves private car owners (drivers) and commuters (usually unknown to each other) in a one-time trip arrangement. The use of mobile phones in organizing ad-hoc ridesharing trips should improve on the shortfalls identified in the current hitch-hiking process.

1.3 Scope and Limitations

The scope and limitations of this research are as follows:

- The thesis does not address the security concerns resulting from the credibility of data provided by the users of the system as no further verifications on details such as persons, vehicles and places are performed in the system's operation.

- The research covers the hitch-hiking in the Eastern Cape such that the localization of the developed system only considered the isiXhosa language.

- The implementation of a low level application for low end mobile phones only considered the Short Message Service (SMS) solution, without other menu driven solutions (for this level) such as the Unstructured Supplementary Service Data (USSD).

1.4 Research Approach

The research used the following steps to achieve its objective:

1. Understanding the concept of vehicle ridesharing and systems that target mobile phones for ad-hoc ridesharing.
2. Selecting a study area that includes the urban and rural contexts and performing a study on hitch-hiking travels to understand how they are organized.
3. Designing a Dynamic Ridesharing System (DRS) based on the identified characteristics of hitch-hiking in the urban and rural contexts of the study area.
4. Identifying the relevant free and open source softwares that were appropriate for the implementation of the DRS and its management.
5. Developing an experimental environment of the system and performing the tests in both the simulated and live environments.
6. Performing a usability study on the system.

1.5 Thesis Outline

The rest of the thesis is organized as follows:

Chapter 2 reviews the concept of vehicle ridesharing and the systems that facilitates driver and commuter ridesharing arrangements. It focuses on the ad-hoc ridesharing method and modern DRSs for organized vehicle ridesharing using mobile phones. The chapter also presents the findings from a related study on why hitch-hiking is popular in the Eastern Cape of South Africa. It further discusses mobile phones and applications with details about their status of use in South Africa. It concludes with a discussion on the use of the Living Lab methodology under the Siyakhula Living Lab (SLL) to promote the use of Information and Communication Technologies (ICT) in marginalized communities of South Africa.

Chapter 3 presents a study that was performed to understand hitch-hiking travels in urban and rural areas of the Eastern Cape. The findings of the study provided details that were used in the design process of a DRS that aims to meet the objective of this research.

Chapter 4 presents the design of the DRS for hitch-hiking travels. It also presents the components which make up the system architecture to provide the conceptual understanding.

Chapter 5 discusses the relevant free and open source software solutions that were used in the implementation of the proposed DRS.

Chapter 6 presents the tasks that were performed to implement the components of the DRS. It includes the configurations and references to code snippets in order to illustrate the implementation of the different components of the system.

Chapter 7 presents the results of the functional and non-functional tests on the DRS. The results are discussed to assess whether the objective of this research was achieved.

Chapter 8 concludes the thesis by discussing the theoretical and practical contributions, followed by the challenges that were encountered in the course of conducting this research. It also provides the suggested future work for this research.

1.6 Summary

This chapter introduced the research and discussed the research problem, objectives, and the outline of the thesis. The next chapter discusses the literature that was consulted in the research processes.

Chapter 2

Background and Related Work

This chapter begins with a discussion of the concept of vehicle ridesharing focusing on the ad-hoc or informal type. It then presents details about ridesharing systems that target mobile phone users in arranging ad-hoc rideshares. This is followed by a discussion on the ridesharing situation within South Africa. Thereafter, mobile phone devices and their applications development are discussed, including a report on their usage in South Africa. The chapter concludes with a discussion of the Living Lab methodology being used to promote the use of ICT in South Africa.

2.1 Ridesharing

Vehicle ridesharing involving drivers of privately owned cars and commuters with a common travel destination has existed since the World War II as an alternative transportation means [15]. Different terms are used to describe vehicle ridesharing, depending on where it takes place. For example, in North America terms such as ridesharing, carpooling or vanpooling (if a van is used) are used, while in the United Kingdom it is known as liftsharing or car sharing.

The most well-known term is carpooling, which is the shared use of a car by the driver and one or more passengers usually for commuting purposes with the arrangements varying in regularity and formality [108]. Ridesharing is sometimes said to be a synonym for carpooling, but it is increasingly used to indicate a form of ad-hoc carpooling, thus with less regularity and formality.

A common point of confusion in defining shared vehicle transportation is how to distinguish between ridesharing, carpooling, public transit and taxi services [6]. In this document, ridesharing uses a similar definition by Amey [6] who describes it as a trip

that the driver intends on taking whether or not they can find an appropriate passenger to share the ride with; and also with no profit-seeking motive. Whereas taxi drivers or public transits are profit seeking in their carriage of passengers, in most cases rideshare drivers seek only to share the costs of transport.

2.1.1 Motivating Factors

There are various reasons that encourage both drivers and commuters to participate in ridesharing travelling. The most common include:

1. Financial incentives

   Drivers see an opportunity to cut trip costs on items like fuel, toll fees, vehicle's wear and tear, etc. Therefore, filling the empty seats with commuters who contribute money reduces the overall trip cost. In some developed countries, discounted parking fees and separate transport lanes are offered to drivers who carry a set minimum number of passengers [15, 2].

2. Society benefits

   Willingness to help stranded commuters, referred as "Good-Heart-Hitch-Hiking" by Sicwetsha [94], strengthens the relationship between drivers and commuters in a community. Society also benefits in the reduction of congestion on roads and parking spaces, assuming that the riders also own private cars and each would have used them for the trip to the common destination [2].

3. Travel time saving

   In countries such as the USA and Canada, the use of faster High Occupancy Vehicle (HOV) highway lanes is an incentive to drivers with passengers occupying all car seats [41]. In this case, commuters target well-known places where drivers pick them up in order to qualify for the use of these special HOV lanes.

4. Environmental impact

   The average new car (e.g. VW Golf 1.6, Mini Cooper Convertible, Volvo C30, and Mercedes Benz CLC) emits 182g of carbon per kilometre. At the other end of the spectrum, a Hummer H2 may emit as much as 412g/km [27]. Participants in rideshare travels (assuming that they own cars), take part in the campaign to reduce Greenhouse Gas emissions [69].

5. Increased travel options

   In areas where public transport is limited or not available (e.g. between home location and the closest public transport service point), ridesharing provides the best travel option for people going to similar destinations. Such irregular trips, also called "last mile" trips, are difficult to be serviced by the public transport systems [58].

2.1.2 Classifications of Ridesharing

There are many terms that define the different rideshare arrangements and Chan [15] proposes a ridesharing scheme based on how ridesharing appears today and the relationship among its participants. Figure 2.1 shows the classifications that are available in ridesharing.

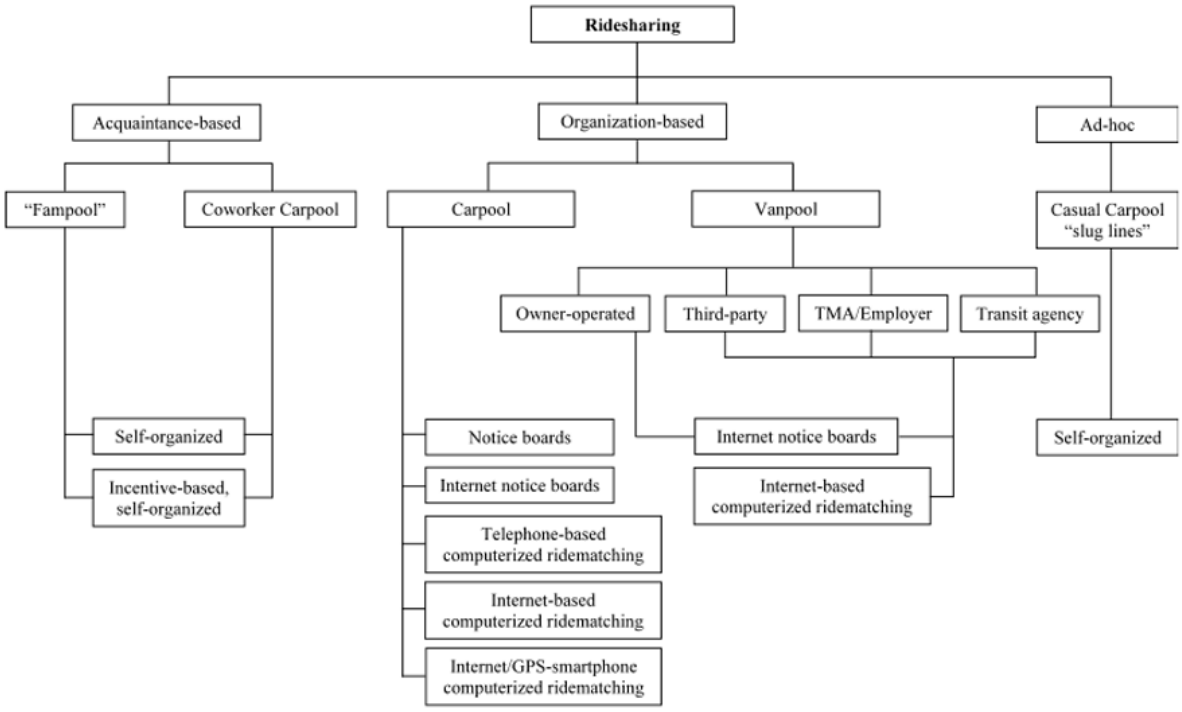

Figure 2.1: Ridesharing Classifications. Taken from [15].

Acquaintance-based and organization-based ridesharing are usually run by a public or private entity in a formal way. Ad-hoc ridesharing as observed in casual carpooling (also known as "slugging", "instant carpooling" and "dynamic ridesharing") is informally arranged and does not involve family, friends or co-workers, but rather runs on first-come, first-served deal [55].

2.2 Ridesharing Systems

According to Chan [15], ridesharing's evolution in major industrial countries of the world can be categorized into five phases: 1) World War II car-sharing (or carpooling) clubs; 2) major responses to the 1970s energy crisis; 3) early organized ridesharing schemes; 4) reliable ridesharing systems; and 5) strategy-based, technology-enabled ridematching.

In general, a ridesharing system helps people to establish rideshares by automatically matching up drivers and riders [2]. Therefore the viability of a ridesharing system depends on the critical number of participants in a given driver/passenger ratio to ensure success of the system [45]. To ensure high participation, some systems offer additional incentives to its users such as financial benefits or subsidies offered by governmental or employer entities [104].

2.2.1 Conventional Ridesharing Systems

Based on the classification shown in Figure 2.1, technology-based conventional ridesharing systems operate under the "organization-based" division, which involves ridesharing that requires participants to join a service whether through formal membership or simply by visiting the organization's website.

These systems formally or informally link riders and drivers assuming they all have a fixed schedule and fixed origin and destination points [53]. Examples of such systems include a rideshare system for workers of a company or students at a University who have a defined schedule of travel. The matching process may take days and results are sent to users as early as possible using communication methods like emails.

Conventional ridesharing systems ensure that people involved in ridesharing trips are well acquainted with each other or share social relationships. The members decide who can join the group based on their assessment of the interested individuals. This improves the safety of ridesharing trips as strangers are not allowed to take part [16, 29].

Rideshare systems ensure that trips are sheduled in advance which guarantees the availability of transport to a destination and should any changes occur, users can communicate via the system's messaging service or emails prior to the travel time [29, 39]. This provides a user with enough time to arrange alternative transport should the need arise.

However, rigid rideshare arrangements have proven to be less desirable due to the restraints they place on participants' extra activities which do not fit into a trip's fixed schedule [21]. Conventional ridesharing systems suit people travelling for common endevours e.g. to their work place or an event at the destination [58]. Therefore, rideshare

arrangements between two or more unrelated individuals for commuting purposes end up being relatively inflexible long-term arrangements [74].

2.2.2 Dynamic Ridesharing Systems

Dynamic Ridesharing Systems (DRSs) target the "ad-hoc" ridesharing classification, according to Figure 2.1, which requires little relationship between participants.

Dynamic ridesharing describes an automated system that facilitates once-off trips close to the desired departure times of drivers and riders. The concept is also known as real-time ridesharing, ad-hoc ridesharing and instant ridesharing [2]. Amey [6] defines real-time ridesharing as a single or recurring rideshare trip with no fixed schedule, organized on a one-time basis, with matching of participants occurring as little as a few minutes before departure or as far in advance as the evening before a trip is scheduled to take place.

Over the years a number of pilot projects aimed at developing dynamic ridesharing systems have been carried out using the available technologies at each point [41]. Some examples include: Bellevue Smart Traveler, Washington (1993) [14]; RideNow, Dublin & Pleasanton, California (2006) [71]; and Avego, University Cork, Ireland (2010) [9].

2.2.2.1 Common Features in Dynamic Ridesharing Solutions

Organized dynamic ridesharing involves people either offering or requesting rides from a system that matches registered potential drivers and riders to the approximate time and destination of the request. The general features that are common in DRSs are as follows:

- Dynamic nature

  The rideshare can be established on short notice, which can range from a few minutes to a few hours before departure time. Thus, communication technology is a key enabler to dynamic, on-demand ridesharing and mobile phones are the best suited devices because of their Internet and Short Message Service (SMS) capabilities. In addition, the origin and destination points are not based on a particular schedule of locations thereby making the points also dynamic [2].

- Organizing non-recurring trips

  Dynamic ridesharing focuses on single, non-recurring trips. This distinguishes it from conventional ridesharing which require a long-term commitment among two or more people to travel together on recurring trips for a particular purpose. Therefore, a DRS assists in organizing a single ridesharing trip which is flexible because it does not require rigid time schedules or itineraries over time [53, 2].

- Pre-arranged trips

  The trips are pre-arranged which means that the participants agree to share a ride in advance, typically while they are not yet at the same location. This is different from the spontaneous, so-called casual ridesharing (slugging or hitch-hiking) in which riders and drivers establish a rideshare on the spot on the side of the street [2].

- Automated matching

  Participants establish rideshares with minimal effort and ride matching is automated in a dynamic setting. The system matches up riders and drivers and communicates the matches to the participants [2, 58].

- Independent drivers

  The drivers who participate are independent individuals that do not belong to any organization for ridesharing travels [2].

- Cost sharing by participants

  The variable trip-related costs are shared among the rideshare participants in a way that makes it beneficial for them to participate from the perspective of cost reduction [2, 15].

2.2.2.2 The Underlying Technologies in DRSs

The earliest form of dynamic ridesharing involved the use of telephone-based ridematching [15]. In this approach, users send their request for rides, offer rides and receive ridematching information in real-time over the telephone. Either human operators or an automated interface communicates with users.

According to Amey [6], the underlying technological features of current DRSs are:

- Smartphone devices

  Many service designs rely on the recent proliferation of smartphones in the marketplace. The firms developing the underlying software for "real-time" ridesharing have focused their efforts on platforms with easy-to-use attractive user interfaces such as Apple's iPhone software and Google's Android platform.

- Constant network connectivity

  The need to communicate ride requests and accept offers on short notice requires that one be constantly connected to the network. Many smartphones offer unlimited data plans with constant network connectivity.

- Ride matching algorithm

  All of the underlying systems use some form of algorithm to match riders and passengers. Some of the algorithms do so based only on origin and destination, while some of the newer algorithms match drivers and passengers based on the commonality of their travel route.

- Global Positioning Satelite (GPS) functionality

  The use of GPS functionality helps participants seeking a ride not to key in their current location because the GPS built into their phone (smartphone) knows where they are located and communicates this information automatically when trips are logged.

- Social network integration

  Some providers have linked their services to existing social networks in an effort to improve successful matches. Lack of trust and safety are the main problems for ridesharing. Wessel [108] recommends the integration of ridesharing systems with social networks for verification, but points out that rather than being a complete solution, this is simply a step in the right direction.

- Participant evaluation

  Participants rate each other in a similar way to the online service eBay [25]. After a ride has been successfully completed, both the passenger and driver are asked to rate each other. Those with higher ratings are likely to be preferable shared ride partners.

- Automated financial transactions

  Some services may allow for financial transactions between participants. Some allow participants to name their own price, while others recommend a value based on standard Internal Revenue Service (IRS) vehicle cost estimates. Some providers facilitate automatic transactions through the use of online payment systems such as PayPal [82]. Other providers simply calculate the recommended shared cost and allow drivers and passengers to negotiate and agree on a final amount and payment method.

2.2.2.3 Advantages

DRSs provide flexible ridesharing services that overcome the barrier of fixed travel times in conventional ridesharing systems. The proliferation of mobile phones coupled with their

ability to perform computing operations and use the internet has opened the possiblity of arranging instant ridesharing [60].

The ability to arrange ridesharing on a per trip basis rather than for trips made on a regular basis suits peoples' less predictable lifestyles [58] (A trip being a single instance of travel from one geographic area to another [53]). In addition, real-time ridesharing attempts to provide added flexibility to rideshare arrangements by allowing drivers and passengers to partake in occasional shared rides when their schedules allow [6].

2.2.2.4 Disadvantages

DRSs are based on the casual carpooling or ad-hoc ridesharing methods which involve matching of people without acquaintances. Therefore, safety and security are the major problems. People have a natural distrust of strangers and are hesitant to ride in a car with one [53]. The security risks include car hijacking, rape and attacks leading to loss of life as indicated in the study by Sicwetsha [94]. Travelling with a driver with an unknown record of driving and in a vehicle with unknown road fitness status poses a major safety risk [23].

There is a trade-off in DRS systems through the provision of valuable travel data against the loss of privacy. The collection of real time data (e.g. GPS and social networks data) involves the loss of personal privacy for the user of the phone (smartphone). The challenge lies in balancing the use of technology for innovative data gathering while ensuring that personal privacy is respected [6].

Participation in most modern DRS systems is limited to smartphone users with high connection speeds and additional features like GPS. This has a significant impact on the number of people who can use the ridesharing services in the context of developing nations, where the penetration of smartphones is still smaller than in developed nations [85].

2.2.2.5 Motivating Factors for Ad-hoc Ridesharing

Ad-hoc ridesharing continues to exist despite most people being aware of the risks associated with it. Levofsky and Greenberg [53] identify the following reasons for the success of ad-hoc ridesharing despite its risks:

1. Incentives to the participants

   Incentives such as money and time saving remain the main motivation for drivers due to the continued increase in driving costs, as discussed in Section 2.1.1. Similarly

for commuters, the flexibility of travelling to a destination not based on a scheduled arrangement, and limited or non existing public transport service.

2. Known pick-up locations

   It is easy for drivers and passengers to locate each other when there are well-known places for people interested in ridesharing or in search of ridesharing. Drivers are assured of finding commuters with common travel destinations at known places in a particular area.

3. Familiarity among participants

   Regular ad-hoc ridesharing partners build trust amongst themselves which encourages more participation in future trips as partners.

2.2.2.6 Mitigating Risks through Technology

Whilst most modern DRSs simplify spontaneous arrangement of trips, efforts are being made to mitigate the risks associated with ridesharing for people without acquaintances by incorporating the available technologies of social networks and GPS [9, 76].

The rise of social networking platforms, such as Facebook, has enabled ridesharing applications to use this interface to match potential rides between friends or acquaintances more easily [15, 108]. The hope is that social networking will build trust among participants, thereby addressing safety considerations. But Chan [15] observes that this may limit users of the system as isolated groups and exclude less tech-savvy users. Some examples of ridesharing applications that focus on social networking platforms are ZimRide [113] and PickupPal [83].

Other systems, such as Avego [9], use GPS and location data to keep track of driver and passenger positions throughout their rideshare trip. This provides a security check that ensures that travel partners of the system's arranged trip arrive at their set destinations.

2.2.2.7 Examples of Modern DRS Systems

This section discusses some examples of DRSs that use modern technology in providing dynamic ridesharing operations.

Avego is a free real time ridesharing application available to iPhone and Windows Phone users. A driver is matched in real-time with anyone searching for a ride along the driver's route.

The system combines GPS-enabled real-time ride matching with fully automated payment transaction management and real-time passenger information [9]. Figure 2.2 shows an example screenshot of the Avego application for Iphone.

Figure 2.2: Avego on Iphone. Taken from [9].

Where safety is concerned, Avego uses a self-policing rating mechanism, whereby drivers and riders rate each other at the end of a journey. Drivers also authenticate riders by entering their auto-generated PIN at the start of each journey. The system uses audio notifications, so drivers don't need to interact with the application while driving. Another security feature of the system is the use of GPS to log the journey progress in real-time [9].

Another example of a DRS is OpenRide. This is a free and open source application for arranging spontaneous shared rides using smartphones. It is device independent as the end device component runs as a mobile web page in the web browser accessible by any internet-enabled device e.g. smartphone, netbook or tablet.

Drivers and passengers on the move can offer or search for rides. The search engine does not only consider the start and end point of the route, but the complete route so passengers can be picked up anywhere along the route and rideshare for any part of the route [76].

To increase the level of trust between a driver and passengers, the system matches user

profiles via a two sided rating system [76].

2.2.3 Flexibility and Reliability Trade-Off

Amey [6] observes that there is a large trade-off involved in the use of real-time ridesharing with the loss of trip reliability in exchange for trip flexibility. However, the degree to which these two features are traded-off depends on the type of rideshare trip being sought. Figure 2.3 shows the trade-off between flexibility and reliability. While conventional rideshare opportunities suffer from a lack of flexibility, they are quite reliable. On the opposite end of the spectrum, immediate rideshare trips are very flexible, but provide little service reliability. Occasional trips, where matching takes place sufficiently far in advance of the start of the trip to allow for alternate travel arrangements to be made, tend to offer a balance between flexibility and reliability.

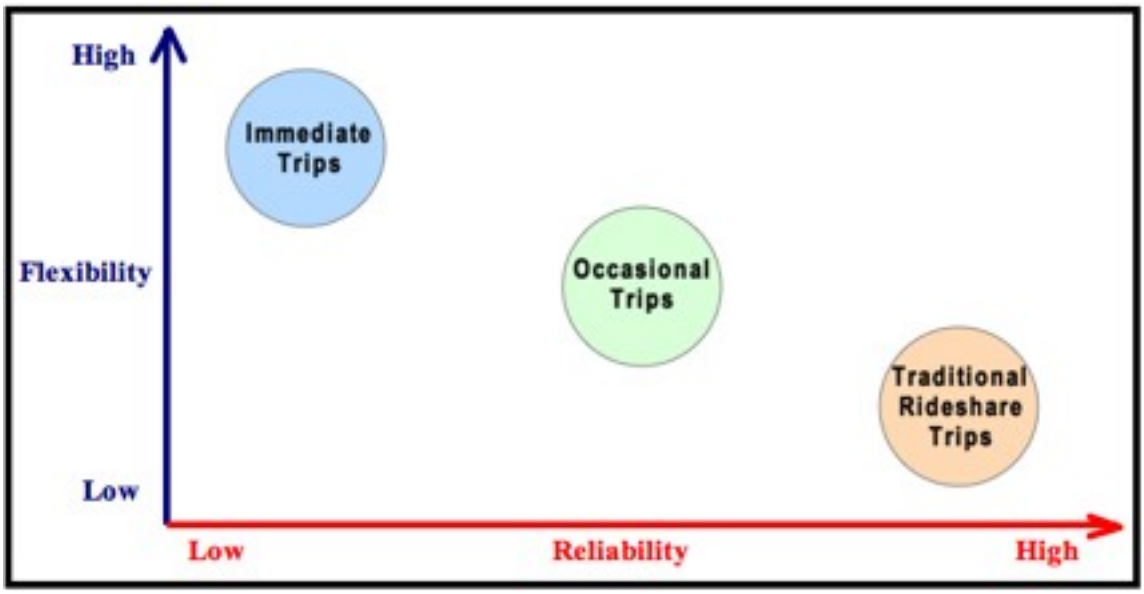

Figure 2.3: Reliability vs Flexibility in Ridesharing. Taken from [74].

2.3 Ridesharing in South Africa's Context

In this section, the ridesharing situation for both conventional and ad-hoc methods in the South African context will be discussed. The section first discusses the state of public transportation and then discusses the ridesharing situation for conventional and ad-hoc methods in South Africa.

2.3.1 Status of the Public Transportation Systems

South Africa provides land based public transportation in the form of trains, buses and taxis (minibus, sedan or bakkie). The significance of taxis as a convenient form of public transport among the three modes of public transport is shown in the survey results in Figure 2.4 [24]. The relative inaccessibility of trains is also shown in the chart. On average, households are approximately half an hour away from train stations. Bus stops and taxi services can be found on average within 12 minutes' walk of peoples' homes [24].

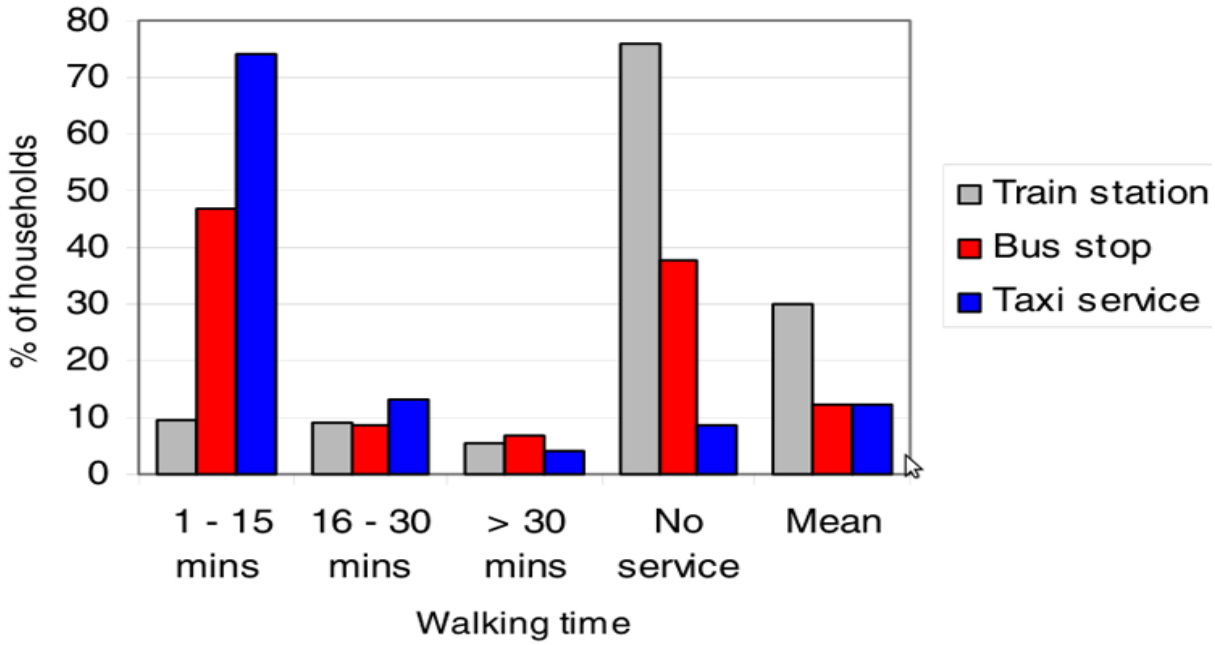

Figure 2.4: Household Access to Public Transport. Taken from [24].

Minibus taxis are responsible for 65% of the 2.5 billion annual passenger trips in urban areas, as well as a high percentage of rural and intercity transport. Buses and trains account for 21% and 14% respectively of all public transport [75].

The statistics clearly indicate that the South African taxi industry plays an important role in the economy considering that the majority of South Africans are dependent on public transport. The taxi industry consists of minibuses, dominating 90% of the market, and metered taxis which are active in the remaining 10% of the market [3].

The public transport sector has its challenges which influence the way people choose their travel modes. The main factors for people not to use the available travel modes, as found in the survey [24], are as follows:

1. Train services: The long distance between peoples' homes and the train stations as

well as the lack of security at stations and on the trains.

2. Bus services: Too infrequent, did not depart/arrive at appropriate times and travel times were too long.

3. Minibus taxi services: Safety from taxi accidents, lack of facilities at ranks and lack of roadworthiness of vehicles.

This section has described the state of the public transport system in South Africa and the challenges that are faced. The next section will focus on how people arrange alternative travels methods in the form of vehicle ridesharing.

2.3.2 Conventional Ridesharing

In South Africa, organized vehicle ridesharing services that are run by local private entities are available. Example websites are FindAlift [29], Gumtree carpool/rideshare [39], Carpoolmates [13] and N1 Lift Club [17]. Generally, these websites offer services of ridesharing for people living in a community wishing to travel to a common destination. People can eventually form lift clubs for regular ride shares based on thier agreed travel schedules. These services fall under the category of conventional carpooling/rideshare systems covered in Section 2.2.1. Some websites (e.g. N1 Lift Club [17]) offer their services at a fee while others are free (e.g. Carpoolmates [13]).

The incentives offered by these web based ridesharing systems are similar to those that were explained in Section 2.1.1 as motivating factors for ridesharing. For example, they all encourage ridesharing for the incentives such as cost sharing (uniting to save money on fuel), reducing their carbon footprint, helping to cut traffic for faster journey times and making new friends [29, 13].

The major limitation with these systems is that their website designs target computer web browsers. This makes their services inaccessible to the majority of internet users in South Africa who rely on mobile phones to access the internet [85].

Since the designs mainly target conventional ridesharing, the flexibility of rideshare schedules could be problematic for the participants, as explained in Section 2.2.1.

2.3.3 Ad-hoc Ridesharing: Hitch-Hiking

Ad-hoc ridesharing in the form of hitch-hiking is a popular method of travel in South Africa, as reported in a study by the Eastern Cape Department of Transport [94].

The study was conducted following the observation of the rise in popularity of hitch-hiking travels involving drivers (private car owners) and commuters for trips within the provincial towns/cities. It involved drivers, commuters and minibus taxi operators who provided their responses through a questionnaire on why hitch-hiking is popular.

The study revealed the reasons behind hitch-hiking and these may also be applicable to other provinces of South Africa.

2.3.3.1 Types of Hitch-Hiking

Sicwetsha [94] defines hitch-hiking (also known as thumbing, hitching, autostop or thumbing up a ride) as a means of transport that is gained by asking people, usually strangers, for a ride in their vehicle to travel a distance that may either be short or long. Commuters use a hand gesture where a thumb is pointed at the road or in some cases write an abbreviated name of their destination on a piece of paper or cardboard for the driver to see.

The study acknowledges the existence of two types of hitch-hiking which are defined as:

1. Good-Heart-Hitch-Hiking: Willingness to help stranded commuters e.g. Drivers "with a good heart" give transport to the elderly, disabled, women or children without expecting any monetary contribution. This has been part of society's ubuntu way of living in most communities. Ubuntu can be explained as the consciousness of our natural desire to affirm our fellow human beings and to work and act towards each other with the communal good in the forefront of our minds [73].

2. Commercial Hitch-Hiking: Rewards motivated hitch-hiking whereby the driver expects a monetary contribution, which in most cases is lower than the taxi cost for the same trip. The growth of this type of hitch-hiking (as observed in the province) indicates that there are inadequacies in the public transport system.

2.3.3.2 Why Hitch-Hiking is Popular

From the commuter's perspective (41 out of 66 [94]) the main reasons for preferring hitch-hiking to taxis were found as follows:

1. The lack of customer care by the taxi drivers citing cases of rude behaviour, bad attitude, noisy music, etc. This is based on the fact that there is no management mechanism in place for the taxi industry as it is run by individuals and not managed by corporate governance structures [94]. In addition, the taxi industry has no customer service plan aimed at providing quality service to its customers.

2. Lack of safety from accidents. The dissatisfaction with minibus taxis due to safety from accidents is also found in the results of the national household survey [24] and is indicated as the overwhelming travel choice factor by commuters. Taxi violence is another reason that commuters do not feel safe and switch to other modes of transport for long and short distance travelling [94].

3. Delays due to time spent at the ranks.

For the drivers who own private cars and pick-up hitch-hikers, 75 respondents [94] gave their reasons as:

1. To reduce fuel cost for their trips.

2. Good will reasons i.e. helping hitch-hikers with transport. It was found that most drivers indicated that they were commuters before and they understood the inconveniences of using taxi services.

3. Use the ridesharing trip as a preaching platform.

The taxi owners and drivers (66 respondents [94]) gave their views on reasons for hitch-hiking as follows:

1. Hitch-hiking was encouraged due to the high taxi fares. They suggested that Government should subsidize the taxi industry in the same way it does with buses and trains.

2. Their services need improvement to win back the commuters. This was admitted by five respondents [94].

3. The time spent at the ranks by commuters. A total of 30 repondents [94] indicated that commuters feel that their time is wasted when taxis delay their departures by waiting to fill all passenger seats.

2.3.3.3 The Risks

In the previous section some of the reasons why people in the Eastern Cape use hitch-hiking as a means of travel were highlighted. Below is a discussion of the risks of this travelling method.

Ad-hoc ridesharing has its associated risks, mainly safety and security as discussed in Section 1.2.2.4. In South Africa's context (as observed in the Eastern Cape), the major risks found in the study [94] were:

1. Increasing rate of criminal activities related to hitch-hiking.

   This makes both the drivers and commuters vulnerable to crimes such as car hijacking, theft, murder, etc. Therefore, trusting a stranger is a problem as was indicated by most drivers (57 out of 75 [94]) who admitted that they do not trust hitch-hikers they pick-up. Only 15 drivers [94] responded that they trust them and had no reasons not to do so.

2. Drivers face attacks from taxi operators.

   Taxi drivers feel that their business is affected by the increase in hitch-hiking travels. In some cases, they confront drivers who are found offering rides to commuters.

The study reports that many people involved indicated reluctance to stop doing it as a means of transport.

2.3.3.4 Legality of Hitch-Hiking

Historically, hitch-hiking is a common practice worldwide with the exception of a few places in the world where laws exist to restrict it. For example, in the United States some local governments have laws to outlaw hitch-hiking for safety reasons [94].

According to Sicwetsha [94], there are no strict and clear laws in the National Land Transport Act No. 5 of 2009 [59] about hitch-hiking in South Africa. The taxi industry has been calling for a legislative ban on hitch-hiking on the basis that it involves drivers who participate in public transport services without proper permits [94].

This has sparked a debate on the rights of citizens and Scwetsha [94] recommends a consultative process involving the Government, taxi industry, commuters and drivers who provide transport to hitch-hikers. This would assist in bringing up relevant regulations and laws concerning the matter.

2.4 Mobile Phone Devices and Applications

In the previous sections, we have discussed ad-hoc ride sharing in the form of hitch-hiking as an alternative method of travel in South africa that is popular in the Eastern Cape. We have also discussed the current state of ridesharing systems that provide real-time ridesharing services and primarily target mobile phone platforms.

This section provides information about the mobile phone devices, applications and their usage status within Africa, paying particular attention to South Africa's perspective. This

information helps to explain the reasons why the majority of people (living in this part of the world) who own mobile phones and participate in ridesharing travels do not benefit from the existing DRS solutions.

2.4.1 Proliferation of Mobile Phones

Africa has overtaken Latin America to become the second largest mobile market in the world after Asia, with over 620 million mobile connections as of September 2011 [37]. Over the past 10 years, the number of mobile connections in Africa has grown an average of 30% per year and is predicted to reach 735 million by the end of 2012 [37]. Figure 2.5 shows the mobile market overview in Sub-Saharan Africa, showing the trend of the increase in the number of mobile connections.

Category	2008	2010	2012	2014	2016
Mobile Connections	262,942,000	387,703,000	502,934,000	605,817,000	700,368,000
Mobile 3G Connections	6,154,000	23,505,000	89,407,000	147,220,000	186,752,000
Service revenue (USD)	30,964,800,000	36,840,000,000	48,850,900,000	59,257,500,000	68,286,300,000
ARPU (USD)	11.51	8.66	8.53	8.49	8.42
Mobile penetration	32.1%	45.3%	57.1%	66.8%	75.4%

Figure 2.5: Mobile Market Overview: Sub-Saharan Africa. Taken from [37].

South African residents are some of the highest users of mobile technology and mobile social networking on the continent. The results of Census 2011 [1] show that the proportion of households owning cellphones significantly increased from 31.9% in 2001 to 88.9% in 2011. According to the Mobile Africa Report 2012 [85], the rural penetration rate in South Africa is at 38%.

2.4.2 Feature Phones Versus SmartPhones

According to Kaigwa [43], there is no official definition to distinguish a feature phone and a smartphone. The definitions change so rapidly due to the improvements in technology and cost efficiencies in manufacturing.

Smartphones describe high end mobile phone devices while feature phones are the corresponding low end devices. The latter can be classified as mobile phones that supports the

Wireless Application Protocol (WAP) on GPRS/Edge connectivity, have a colour screen, can send an MMS message, have low-resolution cameras and support J2ME native applications [43].

The majority of the world's mobile users still make calls and access data using feature phones [100]. Figure 2.6 shows the size of the feature phone market in comparison to smartphones globally (collectively called the "connected device" market) as observed in 2011. Smartphones make up 27% of the graph below, leaving 73% that are feature phone devices.

Nokia is the dominant OEM (Original Equipment Manufacturer) of mobile handsets in Africa. One estimate has positioned Nokia with a 65% market share across the whole of Africa [43].

The dominance of feature phones' mobile operating systems (73%) is shown in the inner circle of Figure 2.6. Such details should have an influence when developing mobile phone applications that target many users in regions such as Africa.

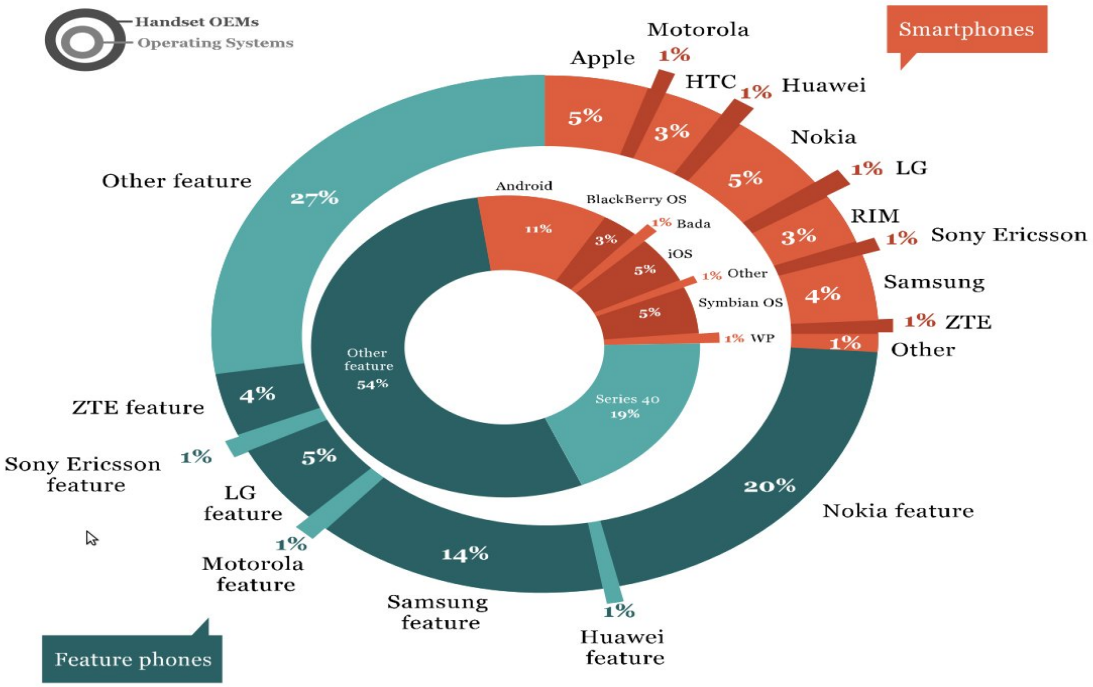

Figure 2.6: Clash of Ecosystems. Taken from [62].

Despite the sharp rise in smartphone shipments in 2010-2011, the rate of smartphone adoption is uneven across different countries. South Africa recorded a 15% (Base: Total population) penetration rate of smartphones in the year 2011 [84]. Developed countries such as UK and USA reported 30% and 31% respectively.

2.4.3 Mobile Phone Applications

We now focus on mobile application development and the different mediums that are available to provide user access to services that can be hosted by content servers on the internet.

2.4.3.1 Development Platforms

Mobile phone application development requires a target platform that provides access to the devices. The platform can be a licensed a product (e.g. Java Micro Edition, JME), or proprietory (e.g. Blackberry); and also open source (e.g. Android) [31]. The selection of an appropriate platform depends on the client devices that the application targets. For example, JME is the most predominant software platform that provides a collection of Java APIs for the development of software for resource constrained devices such as phones[31].

2.4.3.2 Types of Applications

The JME provides one of the preferred choices when developing native applications that can run in most mobile phones that are either feature phones or smartphones. Native applications are specifically developed to run on a device's operating system and machine firmware [4].

Native applications have several advantages which include the ease of tapping into phone features (e.g. GPS, accelerometer, and camera), working offline and also offering rich designs, which enables greater user experience [31, 47].

Native applications are at a disadvantage to support the mobile device fragmentation due to the growing number of platforms (e.g. Windows, Android, iOs, etc). This results in issues such as the requirement of expertise in each respective operating system, use of different environments for each platform and provision of consistent user experiences across all supported platforms [31].

In recent years, the mobile web framework has become one of the most rapidly growing mobile application platform. Content that is device, platform, and operating system independent is rendered by a web browser [31]. Most mobile phones come with pre-installed native applications that are web browsers.

The mobile web applications are easy to develop as they require knowledge in familiar web programming tools such as Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. Web applications enable direct control over own distribution

as any change on the application can be made on the server and clients download the latest application as they access it [4].

The downside of web applications is in the need for constant network connections for their operation. Their performance is bad in areas where network connections are erratic. In addition, it is difficult to develop web applications that access phone features such as the file system (e.g. for the user's address book, photos, etc) [31]. Such file system access presents a large security and privacy issue due to the availability of web based threats that use the internet.

The SMS applications are the other type of mobile applications that can reach many users given the ubiquity of devices that support SMS [31]. They are useful for sending timely alerts to the user and providing two way interactive messaging applications (request and respond)[42]. Their disadvantages include the limited use of only text characters limited to the size of 160 [30], as well as the cost in using them which could be expensive in the course of sending multiple messages [31]. In addition, using SMS applications demands that the user remembers the required keywords and proper structure of the message parameters. This leads to mistakes such as incorrect spacing and misspelled words being made since there is no check on the input parameters before sending an SMS message.

This section has explained some of the types of mobile phone applications that can be developed to provide services to the various mobile phone platforms that are in existence. The next section provides information about the example mobile phone applications that are in common use by people in South Africa.

2.4.3.3 Mobile Applications Usage in South Africa

Recent statistics show that many people in the continent of Africa are slowly being pulled into the world of Internet connectivity by the phones they use [34]. In 2011, 12.8% of the population had Internet access and only 3.8% had broadband connections [34]. Social networks are the entry point to the Internet for many, even in rural areas. It is projected that in five years more than half of those using phones will be using smartphones: one in four Africans [34].

Since the growth of Information Communication Technologies (ICTs), many South Africans living in urban and rural communities are able to explore, share and access digital information through mobile and computer Internet connectivity. The practice of Internet browsing on mobile phones is expanding with nearly 39% of urban dwellers and 27% of rural residents browsing the Internet on their mobile phones [12]. One of the key factors behind the mobile push onto the Internet is the low penetration of both computers and fixed line Internet access in South Africa [34].

According to The Mobility 2012 report [112] by World Wide Worx [111], adult South African cellphone owners are increasingly adjusting their budgets for data use. The report shows that the proportion of the average user's cellphone spend on data has increased by half from 8% of budget at the end of 2010 to 12% in mid-2012 [112]. In the same period, browsing on the phone increased from 33% to 41% of users, application downloads rose from 13% of users to 24%, while Facebook use rose by more than half from 22% to 38%.

While spending on voice has dropped from 77% to 73% in the same period, SMS spend remains steady at 12%. The biggest increases in specific uses of data on the phone were seen in instant messaging services such as Blackberry Messager (BBM) and WhatsApp [109].

Such mobile phone statistics have helped businesses like First National Bank (FNB) to develop strategies to provide their mobile banking services across new channels and platforms like Facebook on top of the more familiar text-based services like USSD and SMS [112].

The Mobility 2012 research study [112] covered South Africans across all age demographics from 16 years and upwards and in all metropolitan and rural areas. However, it did not cover the deep-rural population.

This section has discussed mobile phone devices and some of the types of mobile applications that can be developed. It has also presented information on how the people of South Africa are adopting the use of mobile phones for data access on top of the voice communication services.

The next section will present one of the example methodologies in ICT for Development (ICT4D) for the empowerment of the use of ICT in marginalized communities of South Africa.

2.5 The Living Lab Concept

The Living Lab concept was developed by Professor William J. Mitchell [49] of the MIT Media Lab and School of Architecture. He proposed user-centric research methods in real life environments to identify and build prototypes and to evaluate multiple solutions.

Living Labs use different stakeholders in cooperation, bringing together Enablers, Utilizers, Developers and Users. These co-creators include the public sector, business and science parks, incubators, universities, companies and, of course, the end-user communities, both non-professional and professional [48]. Use of real world testing by end-users in an authentic digital, physical and social environment ensures that emerging technologies

and the innovative products and services are fully developed before they reach the market [48].

Living Labs in Southern Africa (LLiSA) [87] is a project that focuses on building a network and community of Living Labs practitioners in Southern Africa, with the purpose of advancing and supporting open user-centric innovations and Living Labs in South Africa [90]. Living Labs that are part of the LLiSA include Siyakhula Living Lab [51], Limpopo Living Lab [54] and Sekhukhune Living Lab [50].

This research is part of the Siyakhula Living Lab and Appendix F discusses the work that is currently taking place under this Living Lab.

2.6 Summary

The chapter provided background information on vehicle ridesharing focusing on dynamic ridesharing. This was followed by a look at the designs of modern DRSs that enable real-time ridesharing using mobile phones. In the context of South Africa, hitch-hiking as a form of ad-hoc ridesharing was discussed by looking at some of the reasons behind its popularity and its associated risks. Thereafter, information about the status of mobile phone usage in-terms of the types of mobile phones and applications used by most people was also provided. In conclusion, the chapter highlighted the initiatives, through the Siyakhula Living Lab, that are taking place to promote the use of ICT in South Africa.

Chapter 3

Data Collection

This chapter discusses a study that was conducted to understand how hitch-hiking travels are organized in South Africa. The study builds on the results of the study on the popularity of hitch-hiking in the Eastern Cape, discussed in Section 2.3.3. The chapter provides a background of the areas where the study was conducted to consider the urban and rural contexts. Then it presents the methods that were applied to gather the facts about hitch-hiking travels and the results that were found. The chapter concludes with a description of two models of hitch-hiking, for the urban and rural contexts. These models were used in the design of a DRS that is relevant to this context.

3.1 The Eastern Cape

Our study on hitch-hiking was conducted in the Eastern Cape province of South Africa. The Eastern Cape is the second largest province in South Africa with a population of about 6,562,053 (2011) [89]. The majority of the people speak isiXhosa (78.8%), while the other major languages are Afrikaans (10.6%) and English [89].

The metropolitan economies of Port Elizabeth and East London are based primarily on manufacturing, the most important being automotive manufacturing [89]. The province is the hub of South Africa's motor industry. The major towns in the Eastern Cape include Bisho, King Williams Town, Grahamstown and Mthatha.

The deep rural areas of the Eastern Cape are in the former Transkei area. The inhabitants of Dwesa/Cwebe are traditionally subsistence farmers who depend on their crops for their livelihood [102].

Like most marginalized communities, Dwesa/Cwebe suffers from major infrastructure problems including limited electricity availability and connectivity, minimal telecommu-

nication infrastructure, and poor quality of the transport infrastructure [101].

The importance of public road transport to the majority of people in the Eastern Cape is indicated through the results of the South African National Household Travel Survey in 2003 [24]. It was the first national survey of the travel habits of individuals and households. It provides information on workers and commuter trips, and includes the modes of travel, periods of travel, travel times, and travel costs.

The survey ranked the Eastern Cape as the lowest in the household car ownership by province, as shown in Table 3.1. The results showed that in the Western Cape, over 45% of households have access to a car (ownership of one or more privately-owned or use of company-owned cars), compared with approximately 16% in the Eastern Cape.

Province	**% of households with car access**	**No. of cars per household**
Western Cape	45.5	0.68
Eastern Cape	15.5	0.23
Northern Cape	25.4	0.41
Free State	21.8	0.32
Kwazulu-Natal	23.2	0.34
North-West	22.4	0.33
Gauteng	33.0	0.56
Mpumalanga	23.5	0.37
Limpopo	17.2	0.24
RSA	**26.1**	**0.40**

Table 3.1: Households' Car Access and Ownership. Taken from [24].

Among the available transport modes used by people in the province, public transport in the form of buses and taxis provides the major means of land transportation. The significance of the taxi mode (minibus, sedan and bakkie) as the most widely used form of public transport is shown in Table 3.2 [24].

	Transport modes used by all household members in the week prior to survey (Percentage of all people)
Train	0.7
Bus	3.3
Metered taxi	0.5
Minibus taxi	15.9
Sedan taxi	1.2
Bakkie taxi	4.9
Car	8.6

Table 3.2: Transport Modes Used in the Eastern Cape. Adapted from [24].

Given that a large percentage of people in the province rely on taxis, the issues of the quality and reliability of the taxi industry are critical. The dissatisfaction with the taxi industry in the Eastern Cape has been found to be the main reason behind the increase in popularity of hitch-hiking travels, as found in the study [94] by Scwetsa (presented in Section 2.3.3).

The background of the study area for this research has been presented, the next section will discuss a study that was conducted to understand hitch-hiking travels in this area.

3.2 Study on Hitch-Hiking Travels

The primary objective of the study on hitch-hiking was to understand how hitch-hiking travels are arranged between drivers (private car owners) and commuters (hitch-hikers) within the Eastern Cape. This was key to the design of our DRS so that its functions are tailored to the familiar hitch-hiking operations.

The second objective was to get first-hand information on hitch-hiking travels from the participants' point of view, specifically on the incentives, risks and the possible suggestions for how it could be improved. Information gathered in the process was used to improve on the existing hitch-hiking operations and to mitigate the identified risks.

3.3 Methodology

In conducting the study, several research methods were followed and the steps that were undertaken are discussed in this section.

3.3.1 Study Area

To have a broad understanding of hitch-hiking in the Eastern Cape, we conducted our study in urban, peri-urban and rural contexts. It was thought that understanding about hitch-hiking in these contexts could provide a general picture of how hitch-hiking is done in South Africa as the context in other provinces is similar to that of the Eastern Cape.

3.3.1.1 Urban and Peri-Urban Areas

The urban and peri-urban areas in our study were Grahamstown, King Williams Town, Port Elizabeth and East London. Most long distance hitch-hiking travels involve people moving between these urban areas.

3.3.1.2 Rural Areas

To understand hitch-hiking in rural areas, we performed our study in the community of Dwesa and its surrounding areas, located on the south-eastern coast of South Africa in what was previously the homeland of Transkei.

This community is characteristic of many third world rural realities [101]. It provides a good representative area for our study on rural hitch-hiking as it has most of the characteristics that other rural areas in South Africa possess.

The closest towns to Dwesa are Willlowvale and Idutywa, 50 and 75 km inland respectively [102]. People from the rural communities usually travel to these towns for services such as shopping and banking.

3.3.1.3 Map of the Study Area

The study area of our research is shown in Figure 3.1. The map shows the locations of the urban and rural areas that are involved in hitch-hiking travels within the Eastern Cape.

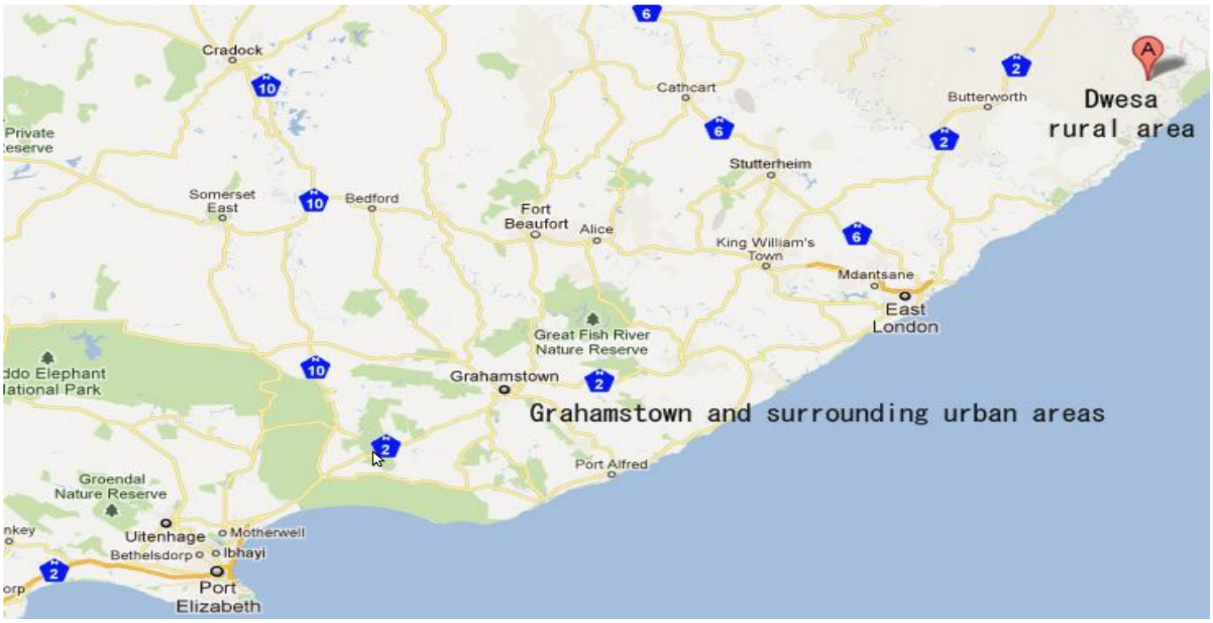

Figure 3.1: Map of the Eastern Cape. Adapted from Google Maps.

3.3.2 Semi-structured Interviews

To get the required information on hitch-hiking travels, we conducted semi-structured interviews with people in the study area. Semi-structured interviews are interviews in which the researcher uses a set of predetermined questions to guide the interviewing process [102]. This is opposed to an open interview process in which no schedule is used. A questionnaire was designed (available in Appendix A.1) as a guide for the interviews and it focused on the following areas:

1. How hitch-hiking travels are arranged between the drivers and commuters.
2. The perceived incentives when participating in hitch-hiking trips.
3. The risks associated with hitch-hiking travels.
4. How hitch-hiking travels can be improved.

The use of semi-structured interviews allowed us to get more detailed information through the ability to use follow up questions; and also get clarifications on some of the responses. In addition, these interviews were important in the rural areas which have scattered communities that makes it difficult to manage survey questionnaire distribution.

People who were not ready to take part in the interview were asked to complete the questionnaire at a later time.

Our sample targeted only people familiar with hitch-hiking travels. Hence, the snowball sampling method [103] was used. Using this technique, participants referred us to other people from the target population.

3.3.3 Participating in Hitch-Hiking Trips

To gain further understanding on hitch-hiking travels, several trips were arranged within the Eastern Cape covering the urban and peri-urban areas. In these trips, we took on the role of a hitch-hiker.

Participating in the trips gave us the experience of hitch-hiking, as well as an opportunity to closely observe:

- How drivers and hitch-hikers find appropriate meeting points (hitch-hiking spots).
- How agreements on the sharing of costs are made between drivers and commuters .
- How the final destination for a hitch-hiker is decided.
- The travel experience compared to the available public transport by minibus taxis.

Taking part in the hitch-hiking trips also provided the opportunity to carry out informal interviews with ride partners, both drivers and commuters. Conducting such informal interviews gave us information from random people, most of whom were regular hitch-hikers or drivers who usually pick-up hitch-hikers. This widened our source of information within the study area.

3.4 Study Results and Discussions

This section presents and discusses the results of the study.

3.4.1 Urban and Peri-Urban Areas

The semi-structured interviews targeted a sample size of 40 people within the Grahamstown area. A total of 22 people gave their feedback with 15 people taking part in the semi-structured interviews. The remaining seven people only provided their responses by filling in the questionnaire. Table 3.3 shows a summary of the responses that were given for the main questions.

Question	**General Response**
How do you find an appropriate hitch-hiking spot?	1) There are well-known places. 2) Strategic points along a street that lead to the intended destination.
What are the benefits of hitch-hiking travel?	**As a hitch-hiker**
	1) Quick travel time. 2) Cheaper than minibus taxis.
	As a driver
	Save travel costs (mainly on fuel).
What are the risks associated with hitch-hiking travels?	**As a hitch-hiker**
	1) Theft cases. 2) Rape cases. 3) Use of unknown vehicles and their condition. 4) Travelling with strangers who could be criminals. 5) Not finding a ride offer.
	As a driver
	1) Car hijacks. 2) Transporting strangers who could be criminals carrying unlawful items.
What recommendations would be suggested to improve hitch-hiking travels?	1) Do not travel alone. 2) Travel details like the car used and the driver must be made available to friends or relatives.
What information would be important before participating in hitch-hiking travel.	1) Details of ride partners, driver and other passengers. 2) Details about the car to be used.
What estimated waiting time would you allow in organized hitch-hiking travel?	Most people indicated a waiting time of between 15 minutes and an hour.

Table 3.3: Hitch-Hiking Survey Responses in Urban Setup.

The respondents indicated that they know the hitch-hiking spots that drivers target when going to a particular destination. Most of the hitch-hiking trips cover long distances to neighbouring places like Peddie, Port Elizabeth (PE) and King Williams Town (KWT). Therefore, Beaufort street which links the main Grahamstown area to the main road to other major towns/cities provides the best hitch-hiking spots. For example, when hitch-hiking to KWT the hitch-hiking spots are along the street in the direction of KWT (near the Shoprite building) whilst when going to PE the best spots are towards the opposite end of the street in the direction of PE.

The benefits of hitch-hiking were given with reference to the minibus taxi services since they are the main providers of public transportation in the area (as was explained in Section 3.1). All respondents emphasized that hitch-hiking provides them with lower travel costs and quicker travel time compared to the taxi services. No respondent indicated any incentive that targets society benefits in the form of reducing traffic congestion or carbon emissions which are some of the main incentives in ridesharing in the context of many developed nations, as presented in Section 2.1.1.

The risks that people face when hitch-hiking were linked to the involvement of strangers which brings in the fear of crimes as is the case in any ad-hoc ridesharing arrangement. The major crimes indicated were car hijacking, theft and rape. All respondents were aware of the risks associated with hitch-hiking travels.

The need for details about travel partners and the vehicle used in the trip were indicated as one way of improving hitch-hiking travelling.

3.4.2 Rural Areas

A similar approach of using semi-structured interviews was done in the rural area of Dwesa, using the same questions as the urban interviews.

There were 14 respondents from around the areas of Ngwane (five respondents) and Nqabara (nine respondents). The responses that were provided are summarized in Table 3.4.

Question	General Response
How do you find an appropriate hitch-hiking spot?	1) Any nearest suitable position along the road. 2) Known hitch-hiking spots usually at bus stops.
What are the benefits of hitch-hiking travels?	**As a hitch-hiker**
	1) Unlimited time schedule. 2) Ability to negotiate with drivers e.g. on cost and drop off point.
	As a driver
	Save travel costs (mainly on fuel).
What are the risks associated with hitch-hiking travels?	**As a hitch-hiker**
	1) Risk on accidents due to the use of unroadworthy vehicles and the overloading of passengers and goods. 2) Not reliable to find a ride offer on expected time. 3) Theft cases. 4) Rape cases.
	As a driver
	1) Car hijacks. 2) Travelling with strangers is risky.
What recommendations would be suggested to improve hitch-hiking travels?	Details of the ride partners and the car must be made available.
What information would be important before participating in hitch-hiking travel?	1) Details of driver and ride partners. 2) Details about the car to be used.
What estimated waiting time would you allow in an organized hitch-hiking travel?	Most people indicated a waiting time of at least 30 minutes to 1 hour.

Table 3.4: Hitch-Hiking Survey Responses in Rural Set-up.

The responses indicated that hitch-hiking in deep rural areas like Dwesa, does not nor-

mally involve specific hitch-hiking spots. Any nearest point along the road can be used as a hitch-hiking spot. One of the reasons could be that there are no specified illegal pick-up points for passengers as is the case with most urban areas. The known hitch-hiking spots in the rural areas are usually the public bus stops.

One of the major benefits of hitch-hiking is the ability to travel at any time, without being restricted to a time schedule. People explained that the rural areas do not have frequent public transport services. The public bus service in the Dwesa area, which is the cheapest compared to taxis and hitch-hiking, is available twice a day (early morning and late afternoon). In addition, the taxis also operate at regular times which makes hitch-hiking the only possible travel option during the day.

The major risk indicated was the safety of the trips. Some of the vehicles used by drivers in the hitch-hiking trips are not roadworthy. And in some cases, drivers overload their vehicles with passengers and goods.

3.4.3 Hitch-Hiking Experiences

Several trips between major towns/cities of the Eastern Cape were arranged to understand hitch-hiking, and they were as follows:

1. Grahamstown and Port Elizabeth (three times)

   In all the three trips, we used hitch-hiking for the direct travel from Grahamstown to Port Elizaberth and taxis were used for the return trips.

2. Grahamstown and King Williams Town

   This hitch-hiking trip was used on our way to East London. King Williams Town was a transit destination.

3. King Williams Town and East London

   This was the final hitch-hiking trip to reach East London from Grahamstown. In the return trip to Grahamstown, a single taxi was used all the way.

By taking part in hitch-hiking travels within the urban setup, we observed the following:

- Use of hand gestures or holding a written post of the destination name to signal rideshare requests to approaching vehicles. The use of written posts is common at hitch-hiking spots that serve multiple destinations. The destination names are usually in short-codes e.g. King Williams Town is KWT.

- The destination name is used when making rideshare agreements between a driver and hitch-hiker. Further details like the final drop off points are discussed whilst the trip is in progress. Generally, the driver decides the final drop off point which depends on his choice of routes at the destination location. Commuters proceeding further than the driver's destination are usually dropped at a hitch-hiking spot to hitch-hike to their final destination. In our three trips to Port Elizabeth, we were dropped at different points in each trip and it all depended on the driver's decision.

- There is a well-known cost of each trip to a particular destination. The cost is usually lower than the equivalent taxi service cost. For example, we paid an amount of R50 to travel between Grahamstown and Port Elizabeth which the driver and the other hitch-hikers were all aware of.

- To reach some destinations it is easier to go through a transit location. For example, going to East London from Grahamstown, it is quicker to go to King Williams Town and then hitch-hike a second time for a trip to East London.

Based on the experience we had in participating in the hitch-hiking travels, we found the benefits in comparison to the taxi services (minibus and sedan) as follows:

- No strict time schedules of travel. As a commuter, you start your journey at your own convenience by standing at a known hitch-hiking spot and waiting. Drivers who are looking for ride partners will stop when they choose to.

- Flexible departure points for commuters. Whilst there are usual hitch-hiking spots, any safe parking point for a driver along the road can be your departure point. Therefore, it provides the ability to start a journey at your nearest point along the main road.

- Taxis depart at the terminal when all passenger seats are filled which in most cases delays commuters. For hitch-hiking, the drivers do not wait for seats to be filled at a hitch-hiking spot but rather start the journey and pick-up additional passengers along the way. This explains why people said that hitch-hiking is a quicker means of travel than taxi services.

- The travel cost is cheaper than taxis. For example, it cost R50 to hitch-hike between Grahamstown and Port Elizabeth whilst travelling by taxi would have cost R80. The costs were also cheaper for the other hitch-hiking trips that we undertook (e.g. between King Williams Town and East London). This verified our earlier findings on why people prefer hitch-hiking to taxis for their travels within the province.

The risks that were observed were similar to those we found in the interviews. But we also noted the following:

- Difficulty in finding a hitch-hiking spot in large urban areas like Port Elizabeth.
- Does not guarantee, as a hitch-hiker, that you will be dropped close to your final destination. On two of the trips to Port Elizabeth we were dropped in the township area where we had to take a local minibus taxi to reach our intended destination (city center).
- Cannot be sure of the actual departure time which is risky if you are travelling on a strict time schedule which requires a specific arrival time at the destination.

We have discussed how we conducted our study on hitch-hiking travels in our study area. In the next section we will present the hitch-hiking models that were derived from the findings of the study to represent urban and rural hitch-hiking travels.

3.5 The Hitch-Hiking Models

We summarized our findings on hitch-hiking travels with two models that highlight the differences that exist in urban and rural hitch-hiking travels.

3.5.1 Urban Hitch-Hiking Model

Using the results of the study, a model of hitch-hiking that describes the characteristics in urban contexts was designed as shown in Figure 3.2.

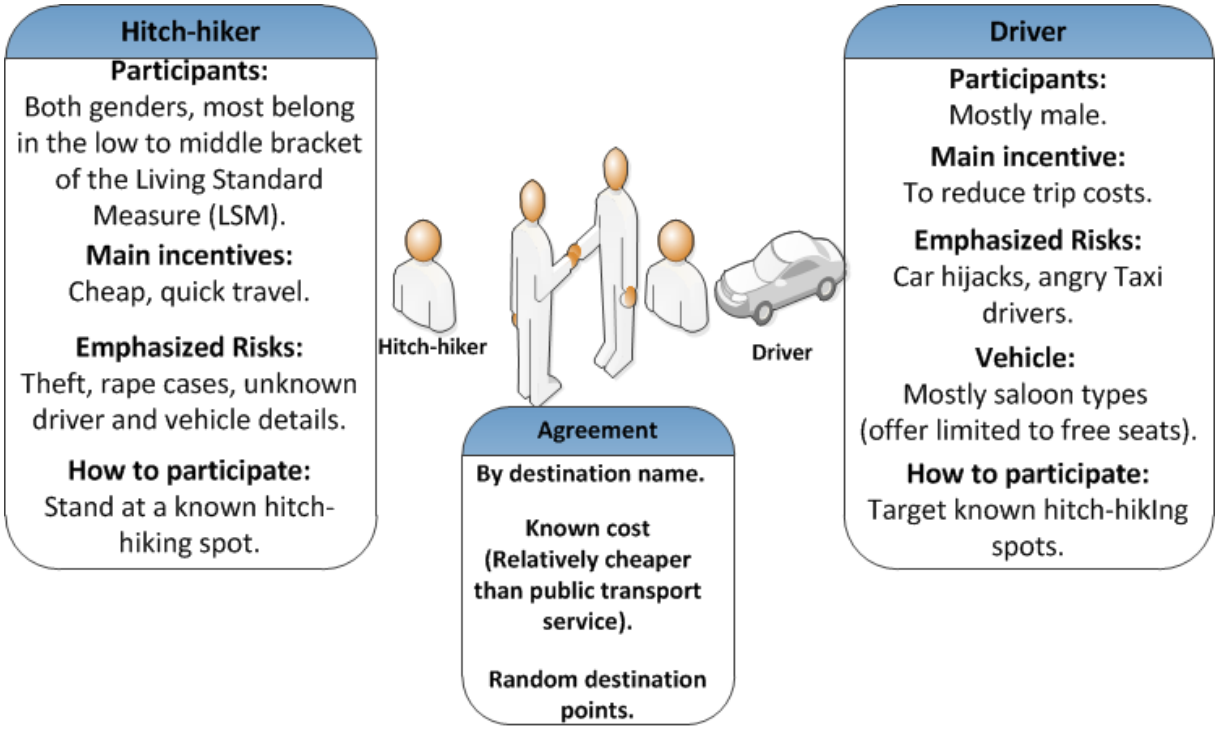

Figure 3.2: Urban Hitch-Hiking Model.

The model shows that male drivers are mostly involved in hitch-hiking travels as we found from our observations and the people that were interviewed. The drivers aim to cut trip costs (fuel) in their long distance trips between towns or cities of the province. The major risk they face in the process of picking-up hitch-hikers are car hijackings; and in some instances they also face resistance from taxi drivers, who feel that their business is affected when they pick-up hitch-hikers.

In the urban context, vehicles that are used for hitch-hiking are mostly saloon types, hence the number of hitch-hikers to be picked-up depends on the available free seats. The drivers search for hitch-hikers at known hitch-hiking spots which possibly increases their chances of picking-up genuine commuters.

Most people who hitch-hike in the urban areas belong in the low to middle bracket of the South African Living Standard Measure (LSM), as indicated in a study by the Department of Transport [94]. Therefore, the main incentive is the opportunity for cheaper travel costs on a similar route to public transport, usually taxis. The other incentives are due to the inefficiencies of the taxi industry such as delays in travel time.

The main risks that hitch-hikers face are crime related in the form of theft and rape; and also travelling with criminals, who may be transporting unlawful items.

The hitch-hikers hold a written post of their intended travel destination name or short-

code to enable drivers to see the final intended destination for a hitch-hiking trip. Once a driver picks up a hitch-hiker, further discussions about the cost and the final drop point can be held whilst the trip is in progress. The final drop point of a hitch-hiker is the driver's decision and mostly depends on his driving route at the destination location.

3.5.2 Rural Hitch-Hiking Model

The rural hitch-hiking model is represented by the diagram in Figure 3.3. The main difference with the urban context is the higher possibility of picking-up hitch-hikers as part of the ubuntu philosophy. The rural context, being composed of community set-ups, have people who are more connected in community settlements than in urban areas. Therefore, people who own cars, in some cases, offer rides out of good will as part of community living. We found that this type of hitch-hiking is defined as "Good-Heart-Hitch-Hiking" by Sicwetsha [94], in Section 2.3.3.1.

Another observed difference is in the type of vehicles used to pick-up hitch-hikers. In the rural context, bakkies are most commonly used vehicles as they are ideal for the untarred road conditions. Therefore, there is no defined seating capacity when people sit at the back, which in some cases leads to the overloading of passengers in a trip.

In rural areas almost all age groups and both genders participate in hitch-hiking due to the limited availability of public transport services. In some instances hitch-hiking offers the only alternative to public transport services and it also provides flexible travel times which are beneficial to the traveller.

One of the major risks people face when hitch-hiking in rural areas is the unroadworthiness of the vehicles used by drivers offering them a lift. Due to limited transport options, sometimes hitch-hikers have no choice but to accept lifts in unroadworthy vehicles. Another safety issue is that the use of bakkies encourages overloading of passengers and goods.

Most hitch-hikers do not need to stand at specific hitch-hiking spots as the road networks are simple and it is easy to locate a good place to catch vehicles going to a particular destination.

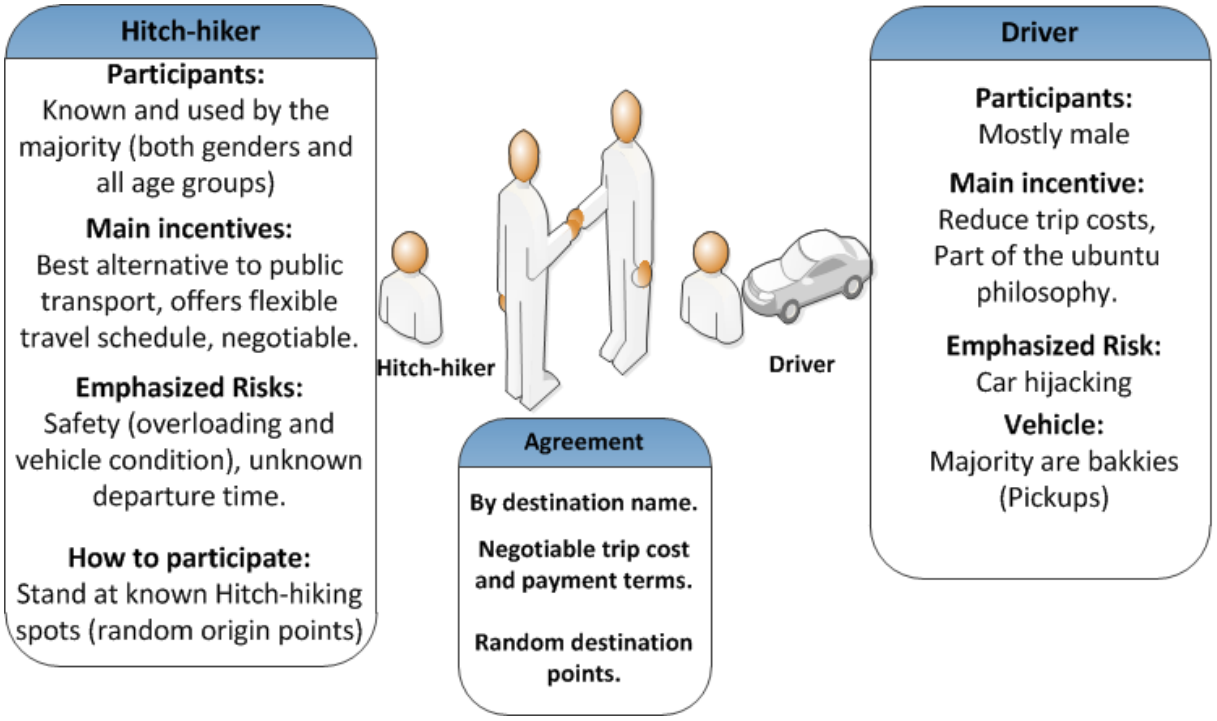

Figure 3.3: Rural Hitch-Hiking Model.

3.6 Summary

In this chapter we have presented the study that was undertaken to understand hitch-hiking travels in the Eastern Cape as part of the requirements gathering to design a DRS for ad-hoc hitch-hiking in South Africa. The area of the study has been explained in such a way that it covers both the urban and rural contexts. The results of the study have been discussed leading to the development of models that represent the urban and rural hitch-hiking contexts. The next chapter focuses on the design stage activities which take into consideration the findings that have been presented in this chapter.

Chapter 4

System Design and Architecture

The previous chapter presented our study on understanding hitch-hiking travels in South Africa. This chapter continues with the development of the DRS by providing details of the design phase activities, including the definition of the requirements specifications. The chapter concludes with a discussion of the overall system architecture, explaining the conceptual design of the DRS.

4.1 System Design

This section explains the design of the DRS by providing details about the system's requirement specifications and the Unified Modeling Language (UML) diagrams that elaborate on the structure and the interaction design.

4.1.1 System Goals

Based on our understanding of DRSs in Section 2.2.2 and the results of the study on hitch-hiking in Section 3.4, the following requirements were identified for our DRS:

1. Registration of users of the system

   Like other DRSs discussed in Chapter 2, all users must be registered to use the DRS. In addition, registering users' phone numbers provides a valuable link for additional user information that can be sourced from South Africa's phone number registration system, RICA (Regulation of Interception of Communication Act) [65]. This might be useful if further identification of a user is required, for example, after a reported bad incident in a ridesharing trip.

2. Ridesharing trips arranged spontaneously

   Trips are arranged in a random manner. In our study on hitch-hiking, we identified that most drivers start their trips by driving towards the known hitch-hiking spots to pick-up hitch-hikers who are ready to travel and the driver does not wait until all seats are filled. Therefore, the system must accomodate this kind of trip arrangement which requires quick responses once a request is recieved and returns the necessary information for quick decisions to be made.

3. Record trip information

   Like other DRSs, the details of a ridesharing trip must be logged in a database and the information should be available to other registered users.

4. Match ride offers (driver) to ride requests (hitch-hiker) and vice versa.

   Ride offers must be matched to hitch-hiker requests who meet the trip requirements specified by the driver. The system should consider scenarios of a ride offer and a ride request posted without available matches.

5. Use simplified operations

   Considering that the majority of our target users are in the low to middle income bracket of the South African LSM, the system operations must be usable by people with low ICT literacy levels.

4.1.2 Use Case Diagram

Based on the system goals, the Use Case diagram that highlights the system's actors and key operations is shown in Figure 4.1.

The system identifies two possible roles by a user which are driver and hitch-hiker. This was considered after our understanding that some people change roles from driver to hitch-hiker and vice versa. As a driver, you are responsible for creating a trip and then offering it according to a set of conditions such as cost, offer duration and number of free seats. As a hitch-hiker, you seek a ride by posting your hitch-hiking details which include your current location and the name of the preferred destination.

The driver decides which hitch-hiker to include in a trip by accepting a list of the matching hitch-hikers or by selecting one at a time.

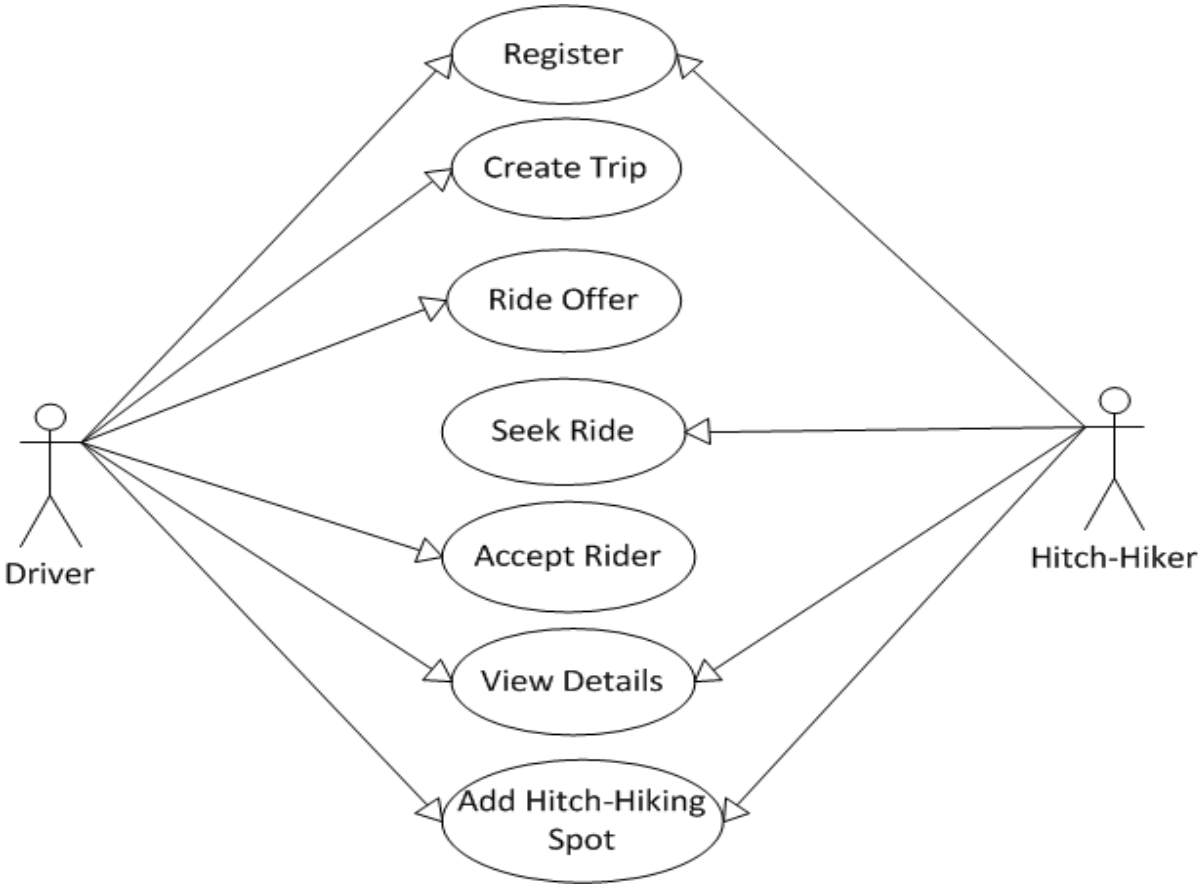

Figure 4.1: Use Case Diagram.

Any users can register a hitch-hiking spot in a location. The hitch-hiking spot can be searched by other users who do not know the location of known hitch-hiking spots or who are looking for the short-codes of locations that are used in hitch-hiking.

The users can enquire about the details of other users, as well as the trips that have been arranged. Information about a particular trip includes the driver, passengers (hitch-hikers) and the vehicle used.

4.1.3 Requirements Specifications

This section presents the functional and non-functional requirements of the DRS. These requirements can be tested on the final developed system to verify that the objectives have been met.

4.1.3.1 Functional Requirements

The key functional requirements based on the objectives of the DRS were set as follows:

1. All users must be registered with their personal details that include their full name, sex, home location and phone number.

2. As a driver:

   - create a trip with details about the final rideshare destination and a preferred hitch-hiker pick-up location.
   - register a vehicle that is used for a rideshare trip under your profile. The vehicle details should be the registration number, make, seating capacity, body type and colour.
   - offer a trip to hitch-hikers with conditions that include a ride offer validity period, the expected rideshare cost and the maximum number of seats.
   - accept a hitch-hiker in a trip to confirm that his/her passenger seat has been reserved.
   - search for available hitch-hikers at a hitch-hiking spot who are ready for a ride to a particular destination name.

3. As a hitch-hiker:

   - post a ride request with the preferred destination and your current location details (name, street and nearest landmark).
   - search for available ride offers to a particular destination from a specified pick-up point location.

4. Match driver and hitch-hiker requests and send instant notifications of matches. The matcher should pend both requests from a driver and a hitch-hiker when no matching result is immediately found. The match alerts should be recieved as SMS messages.

5. Each trip must be independent and have a secret code that is shared between a driver and any hitch-hiker who is accepted. This code can be used by the driver to verify a hitch-hiker at the pick-up point. This should add an extra security check to verify that the person was matched using the DRS, as is the case with the Avego system [9] discussed in Section 2.2.2.7.

6. The rideshare trip information must be recorded, including the day of travel, passengers (hitch-hikers) who were accepted and the vehicle that was used.

7. The known hitch-hiking spots in a particular location must be registered. Details must include the location name, location short-code, street and nearest landmark. Therefore, short-codes must be allowed when sending a ride offer or a ride request as an alternative to full location names. This should enable the use of simple short-codes, which are already familiar to people who participate in hitch-hiking travels; and this could also save time when entering location names that are long.

8. An optional function that adds next-of-kin contact details (full name and phone number). This should enable a user to notify a relative or friend about a hitch-hiking trip they are taking part in. The information about the trip (e.g. vehicle, driver, and other passengers) must be sent to the next-of-kin person by a simple request to the system.

4.1.3.2 Matching Algorithm

In our study on hitch-hiking (in Chapter 3), we found that drivers and hitch-hikers make rideshare agreements using the destination name (full name or short-code). Similarly, we designed our matching algorithm to be based on the destination name entered by a driver and a hitch-hiker.

In the hitch-hiking culture of our study area, we also found that a driver is not expected to drop each passenger (hitch-hiker) at his/her exact destination point. Following this, the system design does not enforce the driver to agree to a hitch-hiker's ride request with an exact drop off point in the destination location as is the case with other DRSs such as Avego [9]. We left the decision on the final drop off point to be agreed by the participants whilst the trip is in progress, as is the case in the normal hitch-hiking travels.

Matching the travel plans of a driver and a hitch-hiker requires a departure time schedule that suits both users. Agatz et al. [2] states that letting each user specify a desired depature time can be problematic for a system to decide the best time to allocate for departure that suits both the driver and possible ride partners. They suggest that a time window representation may capture a participant's time preferences more accurately. One could, for example, let a participant specify an earliest possible departure time and latest possible arrival time as represented in Figure 4.2.

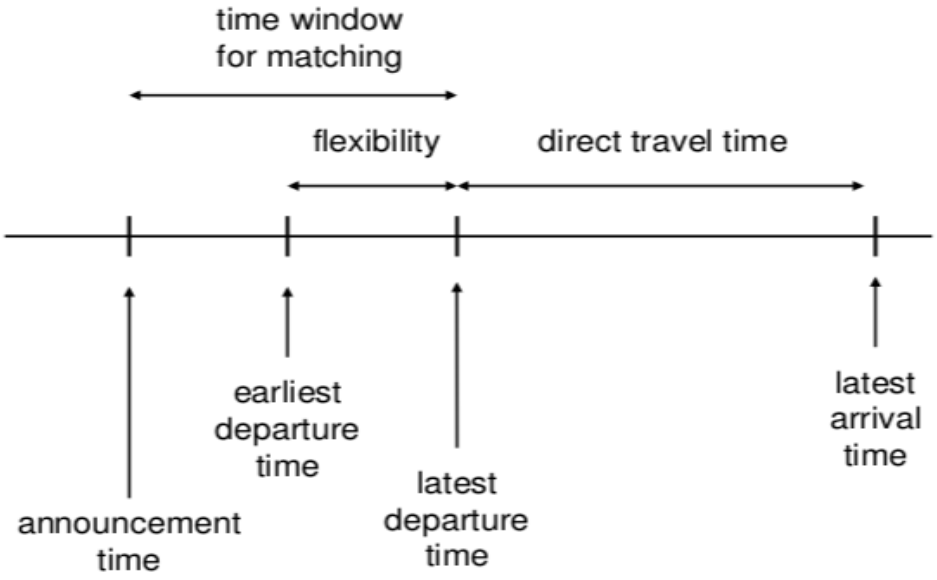

Figure 4.2: Time Window for a Ride Offer. Taken from [2].

Our system adapts the use of a time window by allowing a driver to specify a duration in which a ride offer is valid. We considered the time window for the driver based on the realization that the drivers are solely responsible for the rideshare trip and hence decide when to depart.

Based on Figure 4.2, the time-stamp of the ride offer becomes the announcement time and the latest departure time is calculated by adding the time window to the announcement time. For example, specifying 30 minutes means that the driver is offering a ride from the current time to the end of the next 30 minutes. In this case, any hitch-hiker meeting the trip requirements within this period should be matched. Therefore, a hitch-hiker would be expecting the departure time between the time a successiful match message is received and the latest depature time indicated on the trip (as calculated by the system). But the actual departure time still depends on how long it will take the driver to reach the pick-up point.

The matching algorithm should match a trip ride offer to a hitch-hiker with the same destination name at the trip's indicated pick-up point location. Therefore, a ride offer should return a list of hitch-hikers who match the ride offer request. Considering that devices in use are mobile phones (with small display screens and low memory size), the size of the list is limited by the passenger seats specified by the driver. If the required number of seats is not specified, then the vehicle seating capacity excluding the driver seat must be used. The order of the hitch-hiker requests should be based on the time-stamp of each ride request. The earliest hitch-hiker request must lead the list of ride requests

available to a driver. Then, the driver must decide whether to accept all or select preferred hitch-hikers from the list.

In the scenario of a hitch-hiker's ride request arriving in the system and more than one matching ride offers (by drivers) are available, the ride offer with the nearest departure time is selected. This was done to enable a hitch-hiker to get a matching trip that would best suit their requirement.

4.1.3.3 Non Functional Requirements

The key non functional requirements identified were as follows:

1. System stability
    - The system must be stable in terms of the availability of its ridesharing services.
2. Accessible to the target variety of mobile phone platforms
    - Targeting people who own different types of mobile phones requires access methods that accommodate as many phones as possible. Therefore, the DRS must target the majority of mobile phone owners who participate in hitch-hiking trips in South Africa.
3. System usability
    - Effectiveness - assess the extent to which users successfully complete certain tasks on the system [64].
    - Efficiency - measure how much effort is used in achieving the objectives [90].
    - Satisfaction - find out if the user experience is satisfactory [90].

4.1.4 Class Diagram

The UML class digram in Figure 4.3 shows the static structure of the system. It also shows the classes and relationships that were designed following the requirements specifications.

In the diagram, the *Person* class represents a registered user. The user is represented as a *Hitchhiker* object when the role is to seek for driver ride offers. A ride request by a *Hitchhiker* is performed at a particular point (e.g. hitch-hiking spot) represented by a *RoutePoint* object.

When a *Person* takes the role of a driver, he/she starts by creating a *Trip* object that contains the required trip details (e.g. final destination, preferred pick-up point, *Vehicle* used, etc). Then, a ride offer of the *Trip* can be made for any *Hitchhiker* at the specified pick-up point, i.e. a *RoutePoint* object. A ride offer at a *RoutePoint* is represented by a *TripRoutePoint* object and its properties are the conditions on the offered *Trip* (e.g. trip cost, offer time window, available seats, etc).

The *Vehicle* class has the properties of a vehicle that is used in the ridesharing trip. A *Person* can have zero to many *Vehicle* objects registered under his/her profile. And a single *Vehicle* can be used in many trips organized by different *Person* objects in the role of a driver. We designed it in this way because we were taking into account the fact that a single person cannot solely drive a registered vehicle for all possible trips it may be involved in. Any other *Person* in the role of a driver can register the same vehicle under his/her profile and use it in a ridesharing trip.

The *Trip* class has a property called *Passengers* that contains a list of *Hitchhikers.* The list is updated when a driver accepts a list of matched hitch-hikers or when a single hitch-hiker is added.

The *NextOfKin* class represents the details of a close relative or friend to whom a user can send details of a trip that he/she is participating in. As such, the name and phone number details of the next-of-kin are required. The number of registered *NextOfKins* to a *Person* can be zero to many depending on the user's preference.

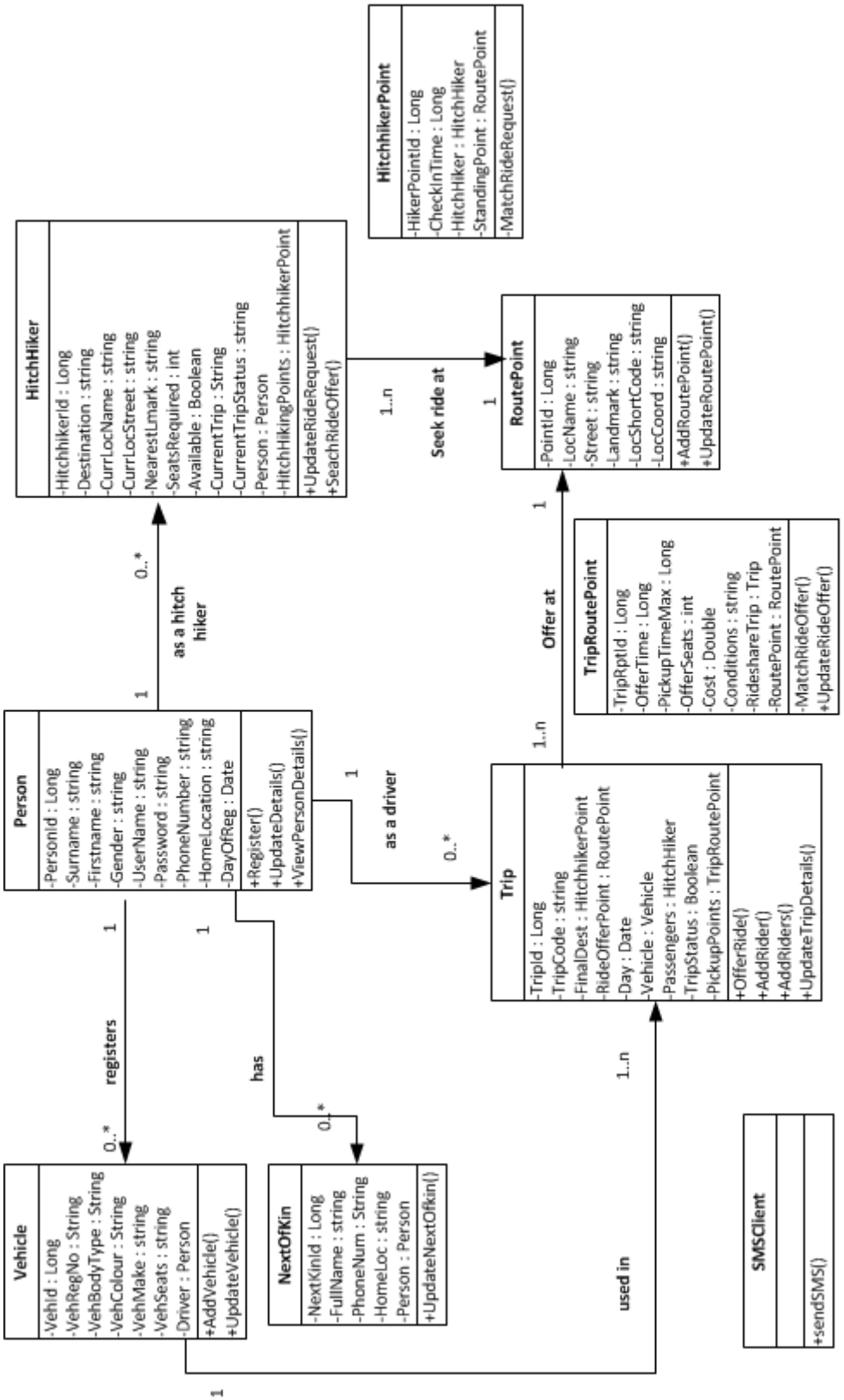

Figure 4.3: UML Class Diagram.

4.1.5 Sequence Diagrams

Representing the activity flow of organizing a rideshare trip, two scenarios can be considered on how rideshare requests from a driver and a hitch-hiker are handled by the system. The scenarios are as follows:

1. A ride offer arriving before a matching ride request

   Referring to the sequence diagram of Figure 4.4, the flow of activities involved starts with a driver creating a trip and then offering it whilst no matching hitch-hiker request is available. As explained in the description of the matching algorithm (in Section 4.1.3.2), a ride offer is only available within a set time window.

   When a hitch-hiker posts a ride request, the matcher looks for any available ride offer. If an available matching ride offer is found, both the owner of the ride offer (driver) and the hitch-hiker are instantly informed about the details of their match.

2. A ride request arriving before a matching ride offer

   In this scenario (shown in Figure 4.5), a hitch-hiker posts a ride request but there is no matching ride offer by a driver. The ride request is pended until a matching ride offer is available. When a ride offer is posted by a driver, the matcher searches for hitch-hikers, at the specified pick-up point or along the same street, seeking a ride to the same destination. A list of hitch-hikers is returned (limited by the number of seats offered) and is sent to the driver. Each hitch-hiker in the matched list is notified about a possible ride offer that has been found and is informed to wait for a confirmation by the driver. Figure 4.4 illustrates the activities that are involved.

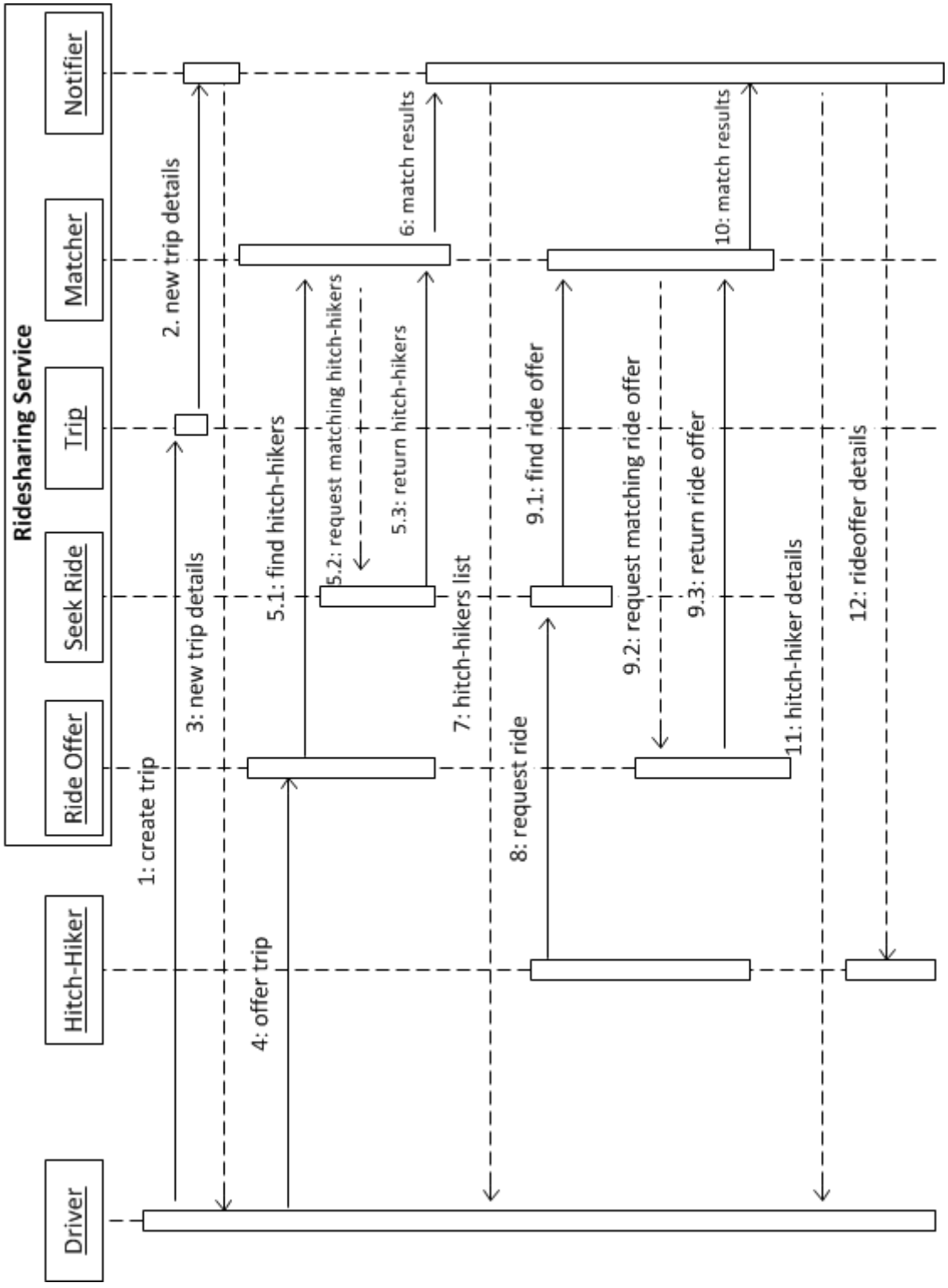

Figure 4.4: Sequence Diagram: Ride Offer Before Ride Request.

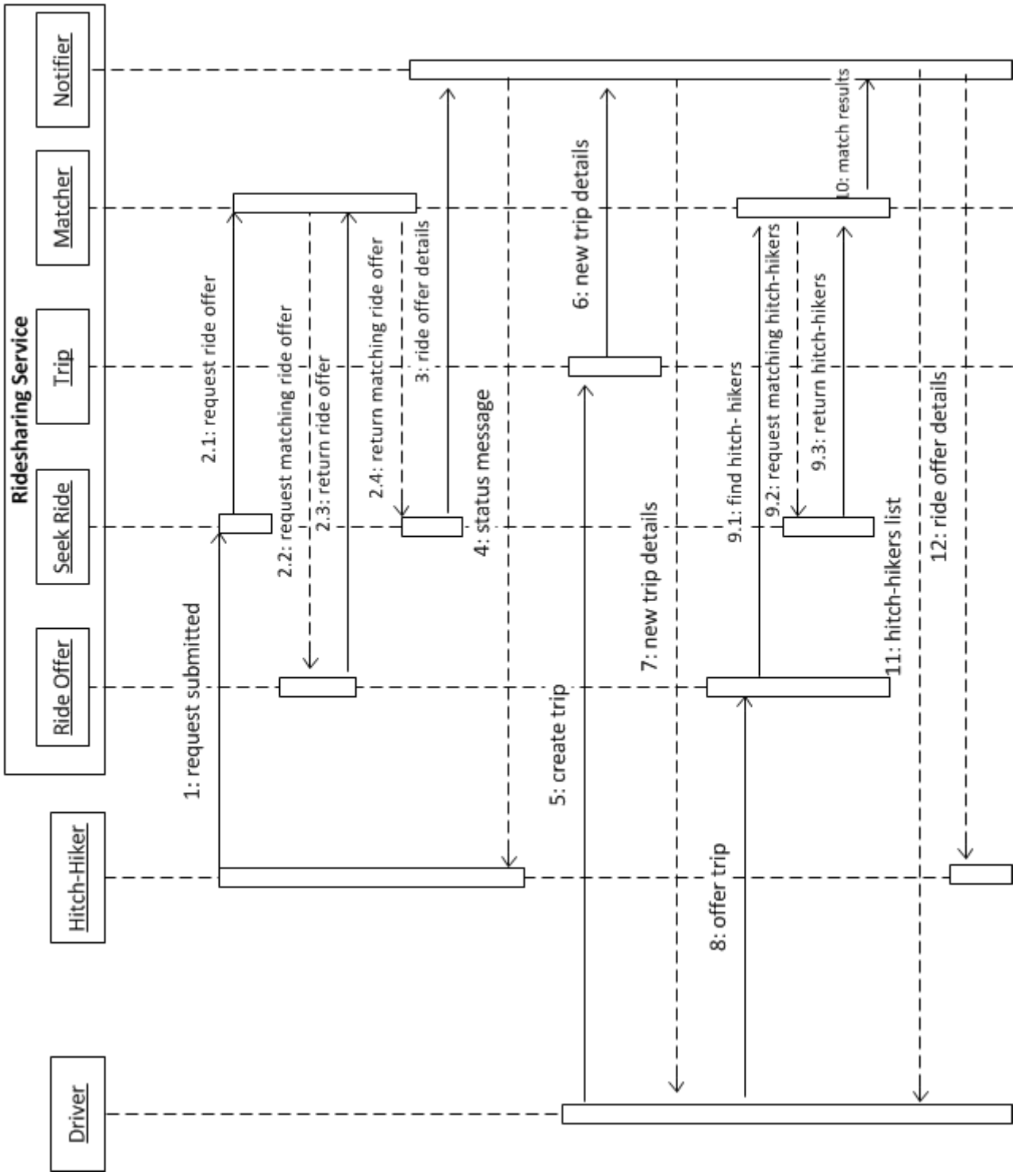

Figure 4.5: Sequence Diagram: Ride Request Before Ride Offer.

4.2 User Interface Design

The User Interface (UI) layer provides interfaces that the user interacts with in order to make use of the services the system offers [90]. Our system's client side is composed of various mobile phone platforms used by drivers and hitch-hikers to organize hitch-hiking travels.

Section 2.4.3.2 discussed the different types of mobile phone applications in the form of web and native applications. Based on the expectation that our target users do not own a single mobile phone platform, solutions that target cross platforms were considered . The prospective users come from a section of the population of mobile phone users in South Africa who own high and low end phones (both categories of smartphones and feature phones).

The next sections discuss our selected mobile phone application types that provide the user interface for the DRS.

4.2.1 SMS Application

The ubiquity of mobile phones that support SMS was highlighted in Section 2.4.3.2, Fink et al. [30] states that almost all GSM mobile phones can send and receive SMS messages. Based on the information about the penetration of mobile phones in South Africa (in section 2.4.1.1), we expect the majority of our target users to own feature phones that operate on the GSM network. By making the DRS's functions accessible through the SMS technology, the accessibility is increased by including a population of mobile phone users belonging in the different levels of the South African LSM (including the lowest bracket).

The other advantages of using the SMS application are the consistent user interfaces and familiarity to most phone users. An SMS text operates in almost the same way across different mobile phone types. This means that users who migrate to different phones would not experience any differences in terms of their interactions with the system. The person-to-person text messaging is the most commonly used by mobile phone users and it is what the SMS technology was originally designed for [42]. This leads to familiarity on how to use the SMS operations for the DRS.

The major setback with the SMS application would be the costs incurred in the process of using the service and the lack of checks on the user entered parameters (e.g. correct requests) before sending an SMS, as indicated in Section 2.4.3.2. This could be expensive to some users in our target population of hitch-hiking participants. The cost factor was considered by minimizing the number of operations when using the system. For example, in a hitch-hiker role only a single SMS message is required to post a ride request, while

for a driver a maximum of three SMSs can be used to create, offer and accept all matched hitch-hikers.

To perform the spontaneous ridesharing arrangements using SMS technology, the system uses a two-way interactive text messaging application [42]. This means that a driver or hitch-hiker sends an SMS that queries the availability of a possible ride partner and the system responds immediately with the results.

The user interaction in SMS applications involves the use of a keyword and parameters (e.g. format: KEYWORD<space>PARAMETER<space>PARAMETER) in a normal text message. The keyword can be used to indicate the type of operation, for example using the keyword "Reg" meaning a registration operation. This is followed by the required parameters (e.g. firstname, username) separated by a delimiter which can be any input character (e.g. a space character).

The design of the user interactions for the SMS application for the DRS is shown in Figure 4.6. The request SMS containing a unique keyword and parameters can be sent to a phone number used by the DRS. The request SMS could be, for example, a ride request containing the details of a hitch-hiker (e.g. origin hiking spot and preferred destination).

Each keyword is associated with an action which is triggered once an SMS with a matching keyword is recieved. The action calls a content server to perform the required task and return the result which goes back to the sender of the request as a response SMS message. This completes the two-way interactive text messaging session.

To make sure that only registered users access the ridesharing service, each request SMS is validated by checking the sender's phone number against the registered users' phone numbers which are unique for each user.

More details about the SMS application are discussed in Chapter 6 of this documents which covers the implementation of the DRS.

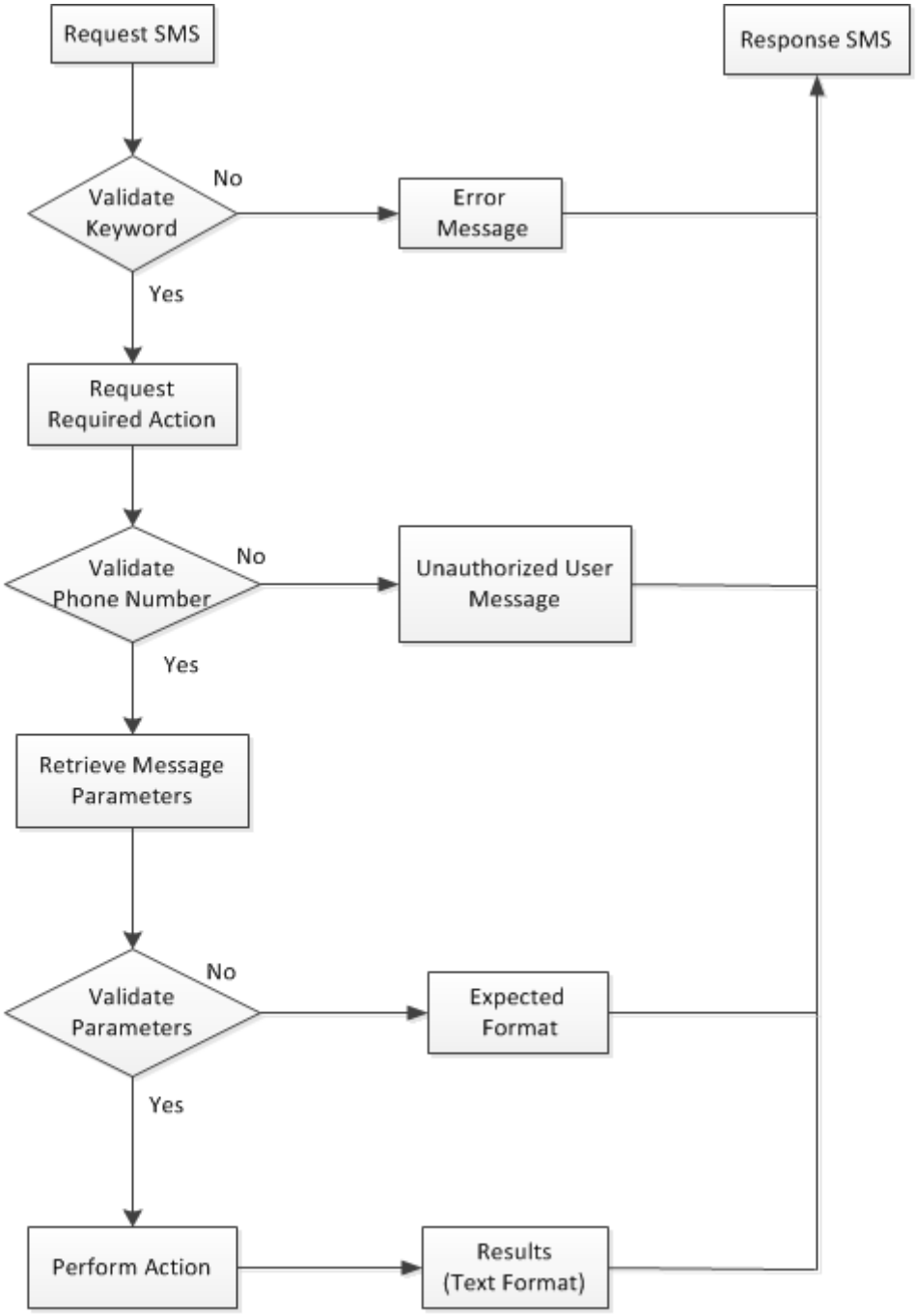

Figure 4.6: UI Model for a Two-Way SMS Application.

4.2.2 Mobile Web Application

The other user interaction method for the DRS is through a mobile web application. Section 2.4.3.2 highlighted that mobile web applications do not need to be installed or compiled on the target device as is the case with native applications. In addition, they allow control by a developer after the deployment of the application. This suits our working environment because our target clients include people of various ICT literacy levels. Those that have weak ICT literacy levels would find it hard to perfom the installation or the necessary upgrades when required to do so.

The other advantage in choosing a web application is the multiple device support for mobile web applications. A phone only needs to have a web browser, which comes preinstalled in most mobile phones. The main challenge would be to make the web application accessible by mobile phones with different specifications (from low end to high end phones). We considered a design that uses simple interfaces and the WAP technology, which optimizes web pages for wireless devices like mobile phones [30].

The user interactions with the DRS using the mobile web application starts with the login process whereby a username and password are required. After a successiful login, the user is provided with a main menu for a selection of the preferred role, either driver or hitch-hiker. Under each role, the relevant operations can be accessed. Figure 4.7 provides the UI model of the mobile web application for performing the DRS functions.

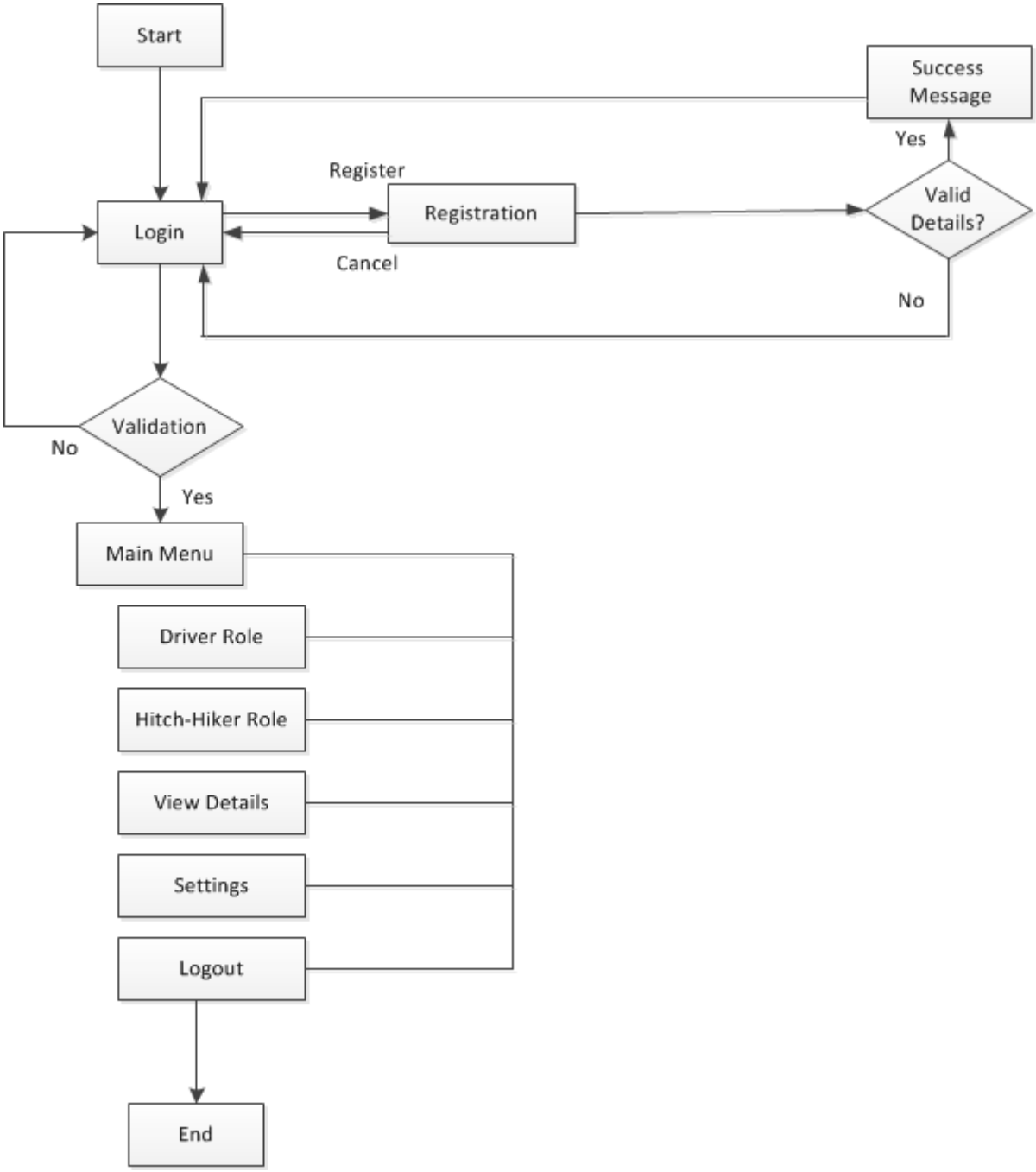

Figure 4.7: UI Model for the DRS Web Application.

4.3 System Architectural Design

The system has the architecture of a client-server design as presented in Figure 4.8. The server side runs the ridesharing service that performs the DRS's functions and persists all data in a relational database.

The client side has various mobile phone devices of diferent capabilities owned by drivers and hitch-hikers for arranging spontaneous rideshare trips. The users can access the DRS functions using either the SMS texts or mobile web browsers. The mobile phone devices operate in different mobile phone networks such as GSM, GPRS, and 3G and above.

The server side requires a communication link to be established with the mobile service provider network which hosts the mobile phone clients. The web server hosting the DRS's web pages needs to communicate with the mobile phone browsers through the Internet using the Hyper Text Transfer Protocol (HTTP).

The SMS application needs a GSM/GPRS modem to link the server hosting the DRS and the mobile phone networks. The user sent SMS messages must be converted to string format that the server understands and similarly, server messages must be structured in an SMS format understood by the mobile phones. This job is performed by an SMS gateway that operates between the mobile phone clients and the server.

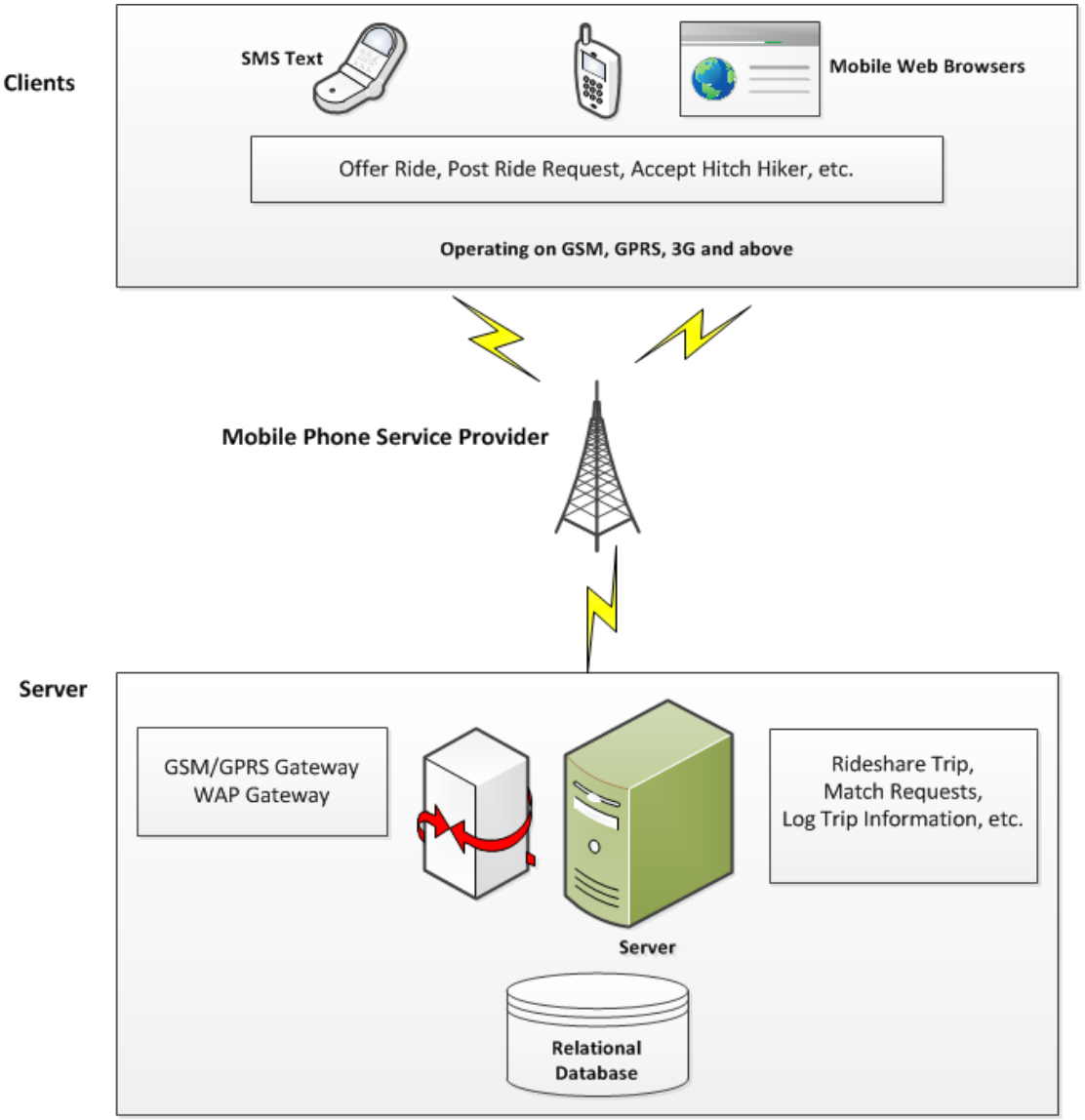

Figure 4.8: The System Architectural Design.

4.4 Summary

In this chapter, the design of the DRS has been discussed, including the requirements specification, the static structural design, the activity flow and the conceptual architecture. The choice of the mobile phone application types has been explained, taking into consideration the information about the type of mobile phones owned by the participants of hitch-hiking travels in South Africa. The next section follows up on the design stage with a discussion of the available free and open source solutions that were considered in the implementation of the DRS.

Chapter 5

Relevant Technologies

This chapter focuses on the various open source technologies that were used in the implementation of the DRS. It covers application development using the Java OSGi technology. TeleWeaver, an OSGi platform for e-services, will be discussed in terms of how services are created, deployed and consumed in the OSGi environment. The chapter concludes with a discussion of the open source solutions for WAP and SMS technologies using the Kannel system.

5.1 Modular Applications with Spring and OSGi

This section discusses the use of Spring Java and OSGi technologies to develop modular systems that use the benefits provided by both technologies.

5.1.1 Spring Framework

Spring is a lightweight Dependency Injection (DI), Aspect Oriented Programming (AOP) container and framework [106]. It promotes loose coupling of classes leading to modular programming. The Spring framework provides for all the needs encountered in the development life cycle of an enterprise Java project or server-side space [88].

Spring follows a software design pattern called Inversion of Control (IoC) in which the flow of the system is inverted with respect to a traditional sequence of procedural calls. The flow is instead delegated to an external entity which performs the sequence calls based on a predefined set of instructions. Spring's implemented type of IoC is called DI [88, 106].

The loose coupling in Spring applications is achieved through DI, whereby objects are passively given their dependencies instead of creating or looking for dependent objects for

themselves. The actual injection of objects takes place when an object of the corresponding class is instantiated, either via its constructor method or setter methods [88, 106].

Spring's core container provides the fundamental functionality of the Spring framework as shown in Figure 5.1. This module contains the BeanFactory, which is the fundamental Spring container and the basis on which Spring's DI is based.

The other modules, shown in Figure 5.1, such as AOP, DAO (Data Access Object), ORM (Object Relational Mapping), and MVC (Model-View-Controller), build on the core module to provide various functionalities to an application developed in Spring. For example, developers who prefer the use of ORM tools over straight Java Database Connectivity (JDBC) utilize the ORM module which builds on the DAO support module. The DAO module provides a convenient way to build DAOs for various ORM solutions e.g. Hibernate [44]. In this way, Spring does not provide a specific ORM solution, but provides the developer with hooks into the available solutions [106].

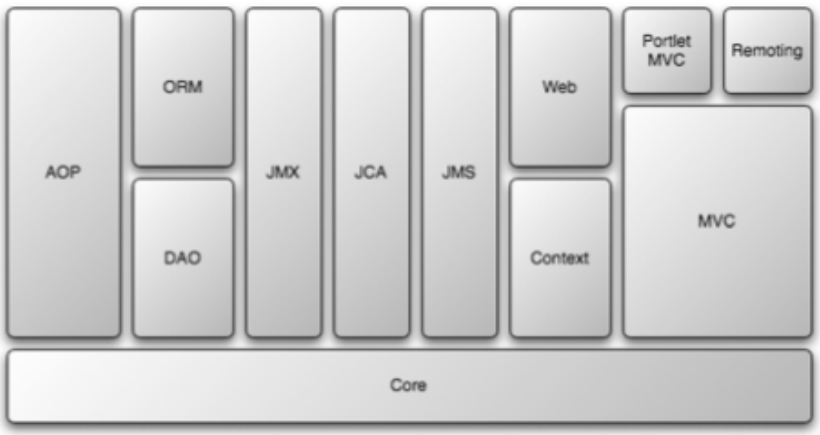

Figure 5.1: The Spring Framework. Taken from [106].

5.1.1.1 Spring Beans

In a Spring application objects are created, wired together and live within the Spring container. Though the components are called "Beans" or "Java Beans", they do not follow all Java Beans specifications. A Spring component can be any type of Plain Old Java Object (POJO) [88, 106].

5.1.1.2 Container Implementations

There is no single Spring container and container implementations can be categorized into two distinct types, namely bean factories and Application Contexts. A bean factory is a class whose responsibility is to create and dispense beans [106]. Apart from creating beans, it also knows about objects within an application and creating associations between collaborating objects as they are instantiated. Therefore, when a bean factory hands out objects, those objects are fully configured, aware of their collaborating objects and are ready to use. There are several implementations of *BeanFactory* in Spring, but the most commonly used is the *org.springframework.beans.factory.xml.XmlBean* factory which loads its beans based on the definitions contained in an XML file. Bean factories are the simplest of containers and they provide basic support for DI [106].

Application Contexts are Spring's more advanced containers as they take advantage of the full power of the Spring framework. Application contexts build on the notion of a bean factory. They provide application framework services, such as the ability to publish application events to interested event listeners and the ability to resolve textual messages from a properties file. An *ApplicationContex*t is preferred over a *BeanFactory* and the three most used implementations are [106]:

1. *ClassPathXmlApplicationContext*

   Loads a context definition from an XML file located in the classpath.

2. *FileSystemXmlApplicationContext*

   Loads a context definition from an XML file in the file system.

3. *XmlWebApplicationContext*

   Loads context definitions from an XML file contained within a web application.

5.1.2 OSGi Technology

Open Services Gateway Initiative (OSGi) emerged in 1999 to address the needs of the embedded device market [18]. OSGi technology is a set of specifications that defines a dynamic component system for Java. These specifications reduce software complexity by providing a modular architecture for large-scale distributed systems as well as small, embedded applications [5].

To understand why OSGi is an increasingly important Java technology, Hall et al [40] gives the following as Java's limitations with respect to creating modular applications:

1. Low level code visibility

   Java provides a fair complement of access modifiers to control visibility (such as public, protected, private, and package private), these tend to address low-level object-oriented encapsulation and not logical system partitioning. For code to be visible from one Java package to another, the code must be declared public or protected if using inheritance. Therefore, should the logical structure of an application call for specific code to belong in different packages then any dependencies among the packages must be exposed as public, which makes them accessible to everyone.

2. Error-prone class path concept

   Applications are generally composed of various versions of libraries and components. The class path pays no attention to code versions (it returns the first version it finds) as there is no way to explicitly specify dependencies. In addition, the process of setting up a class path is largely trial and error, where libraries are added until the Virtual Machine (VM) stops complaining about missing classes.

3. Limited deployment and management support

   There is no easy way in Java to deploy the proper transitive set of versioned code dependencies and execution of an application. The same is true for evolving an application and its components after deployment.

According to Cogoluegnes [18], OSGi addresses Java's limitations regarding modularity in four ways:

1. It defines exactly what a module is, as opposed to Java which defines only deployment units, like JAR (Java ARchive) or WAR (Web ARchive).

2. It provides ways to finely set visibility rules between modules.

3. It defines the life cycle of a module. A module can be installed, started and stopped.

4. It lets modules interact with each other via services.

The OSGi framework plays a central role when you create OSGi-based applications, because it is the application's execution environment [40]. The OSGi Alliance's [5] framework specification defines the proper behavior of the framework, which gives a well-defined API to program against. The specification also enables the creation of multiple implementations of the core framework providing a freedom of choice [40]. The well-known open source projects include Apache Felix [7], Eclipse Equinox [26] and Knopflerfish [57].

OSGi framework provides a standardized environment that is divided into different layers as shown in Figure 5.2. Every OSGi container is based on the Java Virtual Machine (JVM) which runs on an operating system.

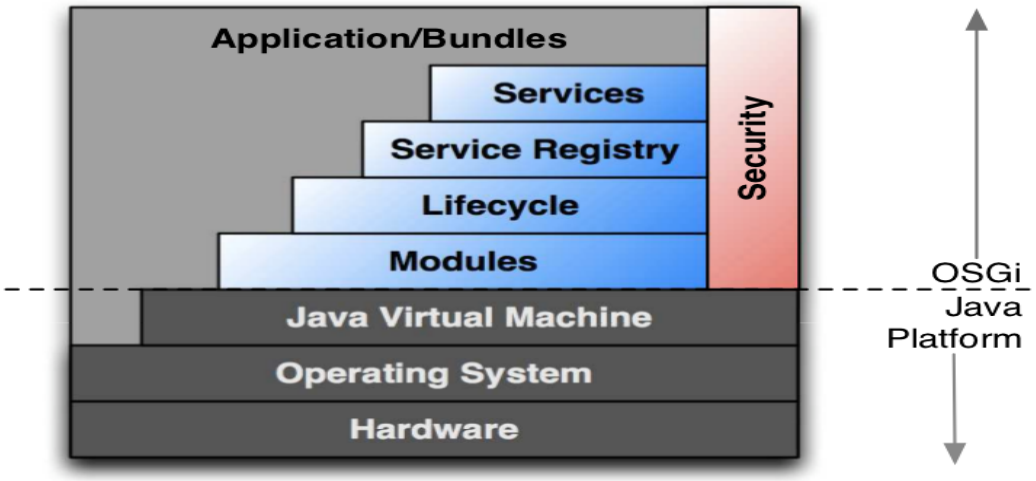

Figure 5.2: OSGi Layered Architecture. Taken from [105].

Referring to Figure 5.2, the three main conceptual layers defined in the OSGi specification are Modules, Lifecycle and Services. Each layer is dependent on the layers beneath it. Therefore, it is possible to use the lower OSGi layers without the upper layers, but not vice versa [40].

The OSGi layers are described as follows:

1. Modules layer

   The modules layer is responsible for handling components which in OSGi are called bundles. OSGi bundles are standard JAR files, with additional metadata in their manifest file that aims at (among other things) identifying them and configuring the visibility rules. The Metadata includes *Bundle-SymbolicName* (unique identifier for the bundle), *Bundle-Version* and *Import-Package* (lists one or more packages a bundle requires) [18, 105].

   Bundles are typically not an entire application packaged into a single JAR file. They are the logical modules that combine to form a given application [40]. The modules layer enforces visibility rules between modules which means that when using classes, the required packages must be specified as being visible before you can reference them from other bundles [18].

2. Life cycle layer

   The lifecycle layer defines how bundles are dynamically installed and managed in the OSGi framework. It relies on the module layer for classloading and provides a dynamic approach to bundle management, making it possible to update parts of an application without stopping it [40, 18].

   In the lifecycle, a component can be installed into the container without trying to resolve dependencies. The next phase is the resolution of dependencies which if successiful leaves the component in the resolved state. Once in resolved state, the component is available in the container for execution and its code executable by other components. The main state at this stage is the active state, which is reached after the start event. Figure 5.3 shows the different states, transitions and events in the lifecycle of an OSGi bundle.

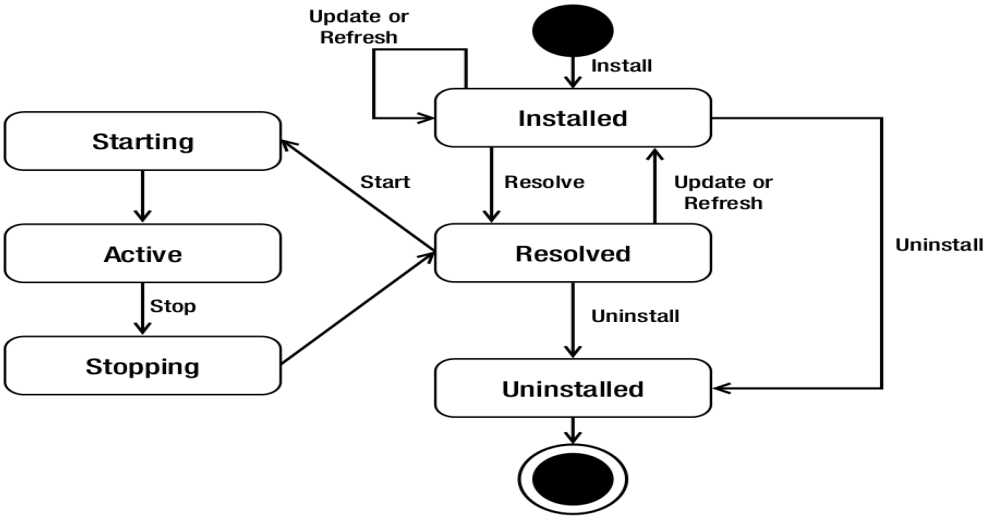

Figure 5.3: OSGi Bundle Life cycle. Taken from [88].

3. Service registry and services layer

   The service registry allows bundles to be used and to interact in a way that takes the dynamic nature of the system into account [18]. The registry enables bundles to publish and/or consume services, as shown in Figure 5.4. This enables a form of Service-Oriented Architecture (SOA). However, unlike many interpretations of SOA, which rely on web services for communication, OSGi services are published and consumed within the same Java Virtual Machine (JVM). Thus, OSGi is sometimes described as "SOA in a JVM" [105].

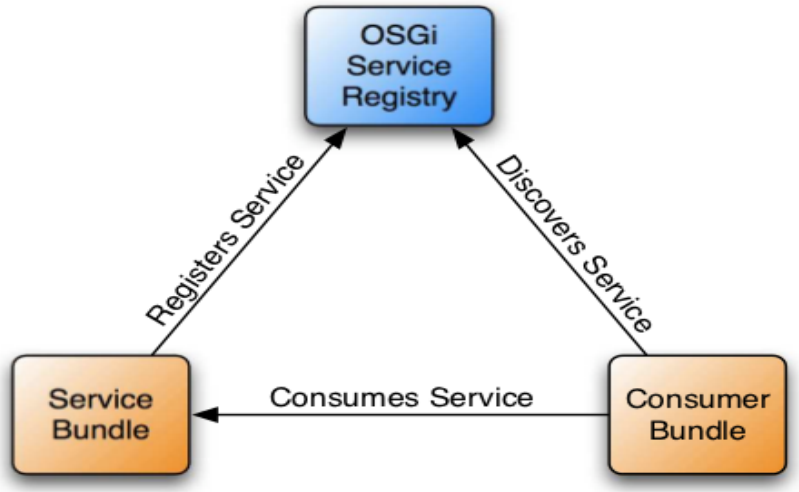

Figure 5.4: OSGi Service Registry. Taken from [105].

5.1.3 Spring with OSGi

In the previous sections, Spring and OSGi have been discussed as stand-alone technologies. This section will focus on how Java applications designed around Spring can benefit from the integration with OSGi technology through the use of the Spring Dynamic Modules (Spring DM).

5.1.3.1 Limitations in Spring and OSGi Technologies

Despite the benefits of both Spring and OSGi technologies discussed above, there are several limitations that exist in these technologies. The drawback of Spring framework occurs when trying to scale up to large and complex applications. In this case, a lot of beans need to be configured within the Spring container, making its configurations more difficult to maintain. Again, Spring suffers from a lack of modularity and no support to improve this, such that there is no way to limit the visibility of a bean in Spring [18].

In OSGi technology, the core specification does not provide any support for patterns or tools in the design and implementation of bundles. This leaves the developer free to choose a suitable architecture and framework (In this case, Spring can be the solution). In addition, using the service management API is tedious, and managing the dynamic nature of services in user code is error prone [18].

5.1.3.2 Benefits of Spring with OSGi

Rubio [88] suggests that OSGi is about the packaging, versioning, and dynamic loading of Java applications, while Spring is about pursuing a pragmatic approach to developing Java applications using POJOs and the IoC. He further states that the OSGi's strengths, presented in Section 5.1.2, can be greatly enhanced by leveraging the Spring framework using the following techniques:

1. OSGi's core services can make use of Spring's DI approach, making the development of OSGi-bound classes simpler and more amenable to testing.
2. OSGi can leverage Spring's ample foundation for delivering enterprise-grade applications.

Similarly, Spring can gain from adopting some of OSGi's features, yielding the following benefits:

1. The DI technique can be performed more intelligently with the help of OSGi's dynamic-loading capabilities.
2. Class libraries used in Spring applications can be shared more easily, with a lesser threat of class conflicts due to OSGi's awareness of other bundles (JARs) running in the system.
3. Spring applications can be designed to support class versioning, using OSGi's capabilities in this area.
4. Spring beans can be registered as OSGi services, making them readily available to other bundles (JARs) running in the system.

5.1.3.3 Spring Dynamic Modules

Spring Dynamic Modules (Spring DM) is a technology that bridges the gap between the Spring and the OSGi frameworks, combining the simplicity and power of Spring with the modularity, flexibility and dynamism of OSGi [18]. Figure 5.5 shows how Spring DM bridges between components (bundles), the service registry and embedded containers.

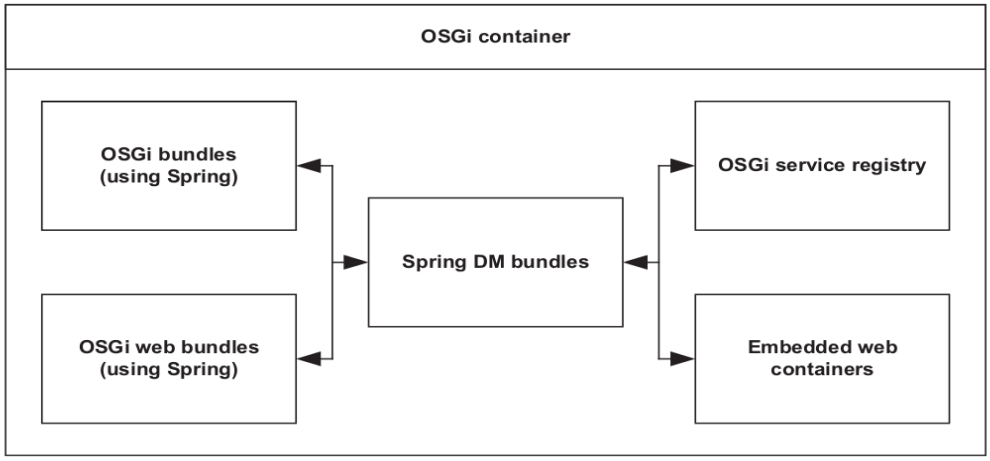

Figure 5.5: Spring DM in OSGi Container. Taken from [18].

Spring DM is composed of a set of OSGi bundles that become part of the infrastructure when installed in the OSGi container. They manage a Spring application context for bundles or start a web container inside the OSGi container.

An example of a Spring DM bundle is the Spring DM standard extender, which is a special bundle that listens for bundle installations in the OSGi container. It scans bundles looking for Spring configuration files and creates Spring application contexts on behalf of these Spring-powered bundles [105, 18].

In a classical Spring application, a dedicated Spring container is used, and the framework is configured from its enclosing contexts (the classpath or web environment). With Spring DM, a dedicated Spring container is associated with each Spring DM–powered bundle. Therefore, each container is unconnected with the rest and is internal to its enclosing bundle. The use of separate containers is mandatory because each bundle has its own lifecycle and can appear or disappear at any time [18]. Figure 5.6 shows the Spring-powered bundles that benefit Spring framework features such as DI, but also benefit from the OSGi environment like any other OSGi bundle. Spring DM helps them with tasks such as the interaction with the OSGi service registry.

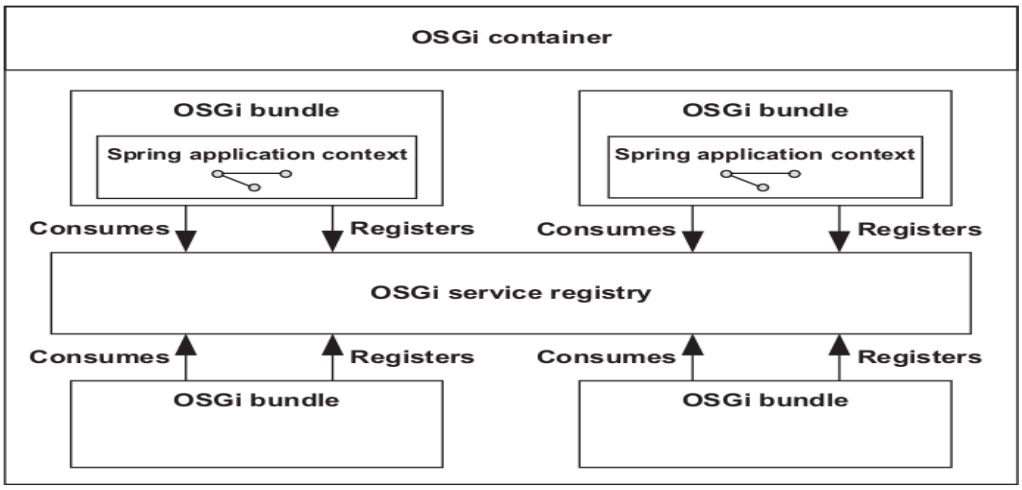

Figure 5.6: Spring-powered bundles. Taken from [18].

When registering services with Spring DM, the preferred procedure is to use the Spring context definitions which are executed at the time a bundle's Spring context is created. The definitions are taken from Spring DM's predefined or default trigger point locations. Spring DM automatically processes any XML file located under a bundle's */META-INF/spring/* directory. Therefore by using the OSGi namespace in the XML file, a bean is published in the OSGi service registry using the XML tag *osgi:service* from the OSGi namespace. Similarly, to retrieve a service from the OSGi registry, the tag *osgi:reference* from the OSGi namespace is used [88].

5.1.4 OSGi Development with Maven

The use of software project management and comprehension tools is important in software applications development. This section looks at the use of Apache Maven in OSGi application development.

Maven is a software project management and comprehension tool. Based on the concept of a Project Object Model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information [8].

The primary goal of Maven is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. There are several areas of concern that Maven attempts to deal with and they are as follows [8]:

- Making the build process easy.
- Providing a uniform build system.

- Providing quality project information.
- Providing guidelines for best practices development.
- Allowing transparent migration to new features.

5.1.4.1 How Maven Works

Maven provides archetypes (project templating toolkit) for creating the skeleton of a project [18]. There are different types of archetypes depending on the nature of the project. One example is the *maven-archetype-webapp* which creates a simple web application project.

Maven downloads dependencies from public repositories on the internet and stores them on the local file system, in the local Maven repository: in the *~/.m2* folder on Linux and *C:\Documents and Settings\username\.m2* on Windows [8].

A Maven project generates a POM file that contains two sections with information about the project [18, 8] and these are as follows:

1. The project identification with the coordinates such as project group id, artifact id, version and the packaging element (JAR, WAR, etc.).
2. The project dependencies with coordinates such as group id, artifact id, version, etc.

Maven does not only handle direct dependencies but a whole graph of dependencies. This notion is also referred to as transitive dependencies. For example, if a framework like Hibernate is specified as a dependency, Maven automatically provides all the dependencies that are required by Hibernate [18].

5.1.4.2 Maven Based OSGi Projects

OSGi-based software development is supported in Maven 2 and 3 versions [8]. There are several remote repositories that provide coordinates for dependencies that are OSGi compliant. These include MvnRepository [67] and SpringSource EBR (Enterprise Bundle Repository) [96].

When developing OSGi applications, the dependencies that are downloaded for the project must be OSGi-fied library versions. OSG-fied is a term that refers to a Java library with OSGi specifications [18]. The SpringSource EBR [96] is the recommended source for dependency coordinates of OSG-fied libraries for OSGi environments.

5.2 Data Access in OSGi

The previous sections have discussed Spring, OSGi and Spring DM in modular application development. The use of a database to store data is an important aspect of any enterprise application. This section explores how data access components are implemented in an OSGi environment.

5.2.1 Java Database Connectivity

The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases. The JDBC API provides a call-level API for SQL-based database access [79].

JDBC provides an abstraction layer above the vendor-specific database drivers. Although any database-based Java application must rely on JDBC, there is a structural mismatch between object-oriented applications, which are based on objects, and relational databases, which are based on rows, tables and relations [18].

Cogoluegnes [18] states that using JDBC within OSGi has particular constraints, especially with respect to the JDBC driver (*DriverManager*). The problem is that the *DriverManager* class, which is the central class when using JDBC, uses classloaders in a specific way that can have strange side effects (e.g. unexpected *Class-NotFoundException exceptions*) within an OSGi container. It is advised not to use the *DriverManager* class in OSGi, but instead to directly obtain a JDBC connection using the driver interface.

5.2.2 Object Relational Mapping

ORM enables applications to work with the object representation of the data in database tables, rather than dealing directly with that data. At the basic level, each ORM framework maps entity objects to JDBC statement parameters when the objects are persisted. It also maps the JDBC query results back to the object representation when they are retrieved [93]. Figure 5.7 illustrates how ORM interacts with a relational database on behalf of an application. A Java application persists objects of the mapped classes without concern about the JDBC implementation details in order to save the data in a relational database. An ORM technology operates between a Java application and a relational database to perform the required conversions of the application's objects to records of a database table.

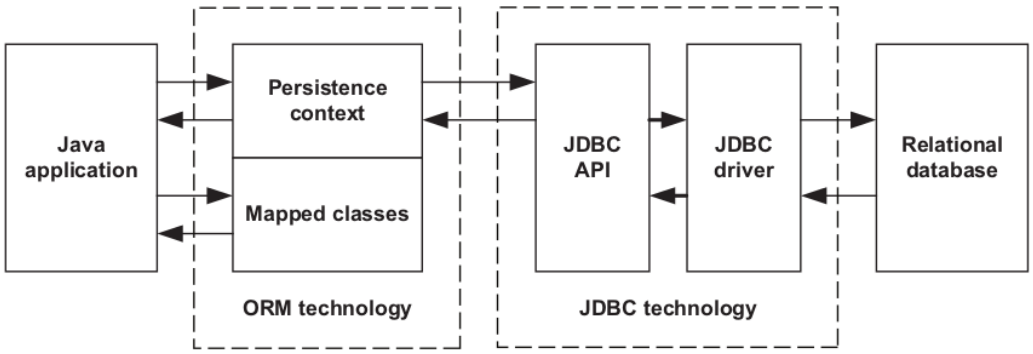

Figure 5.7: General ORM Architecture. Taken from [18].

5.2.2.1 Objectives

ORM tools aim to address issues that come up due to the structural mismatch between objects and the relational tables in a database which are as follows [18]:

- Handling specific SQL dialects since every database has its own extension of SQL specification.
- Execute a vastly different number of SQL requests when an application uses entity relationships.
- Handling object graphs which JDBC does not provide an easy way.
- Entity mapping with table and inheritance support.

5.2.2.2 Frameworks

Spring does not provide its own solution for persistence. It unifies pre-existing solutions under its consistent API and makes them easier to use [93]. The available persistence solutions include Java Persistence API (JPA) [78], Hibernate [44] and TopLink [80].

To use an ORM framework in the OSGi environment with Spring DM, Cogoluegnes[18] outlines the following steps:

1. Provision the OSGi container with bundles required for the chosen persistence API e.g. Hibernate. The OSGi-fied components can be found from the SpringSource EBR [96].
2. Configure JPA entities with Spring JPA support.

3. Configure the global packages in the OSGi configuration.

4. Implement the Data Access Object (DAO) classes.

5.3 OSGi Web Components

This section focuses on the development of web applications in OSGi environments using Spring DM.

5.3.1 Web Container in OSGi

In normal Java application development, a web application is deployed into a web container (e.g. Tomcat [33]) in a Java application server. In OSGi, instead of deploying an application into a web container, a web container is installed in an application. In this case, a web container is deployed alongside an application in the OSGi framework [105]. Two common choices of web containers in OSGi are Tomcat [33] and Jetty [32].

The OSGi framework does not know how to hand a WAR file off to the servlet container [105]. As such, web applications in OSGi are deployed using the Spring DM web extender. The web extender is a bundle that has an activator whose job is to watch for bundles to be installed into the OSGi framework. When it discovers a web bundle, it deploys that bundle to the web container which could either be Tomcat or Jetty [105].

5.3.2 Web Frameworks

Spring DM can integrate with various web technologies and frameworks to create web User Interfaces (UIs) and web services. We now look at how Spring DM works with different contexts of web frameworks.

5.3.2.1 Action-Based Frameworks

Action-based web frameworks are responsible for selecting the appropriate actions to handle requests. Actions are responsible for extracting parameters from requests, executing the requests and building responses [18].

Integrating Spring DM with action-based frameworks involves the use of OSGi services, which are configured through Spring DM, in action implementations. These actions use OSGi services to execute the business functions. Figure 5.8 shows the implementation design of action-based web frameworks with Spring DM in an OSGi environment.

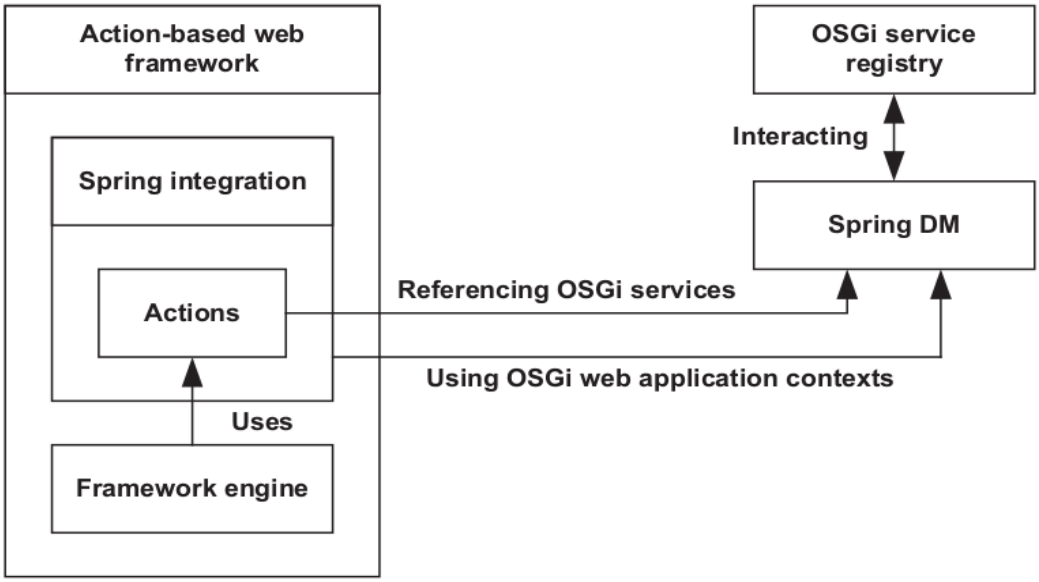

Figure 5.8: OSGi and Action-based Web Frameworks. Taken from [18].

Spring MVC is an action-based web framework shipped with Spring itself. In Spring MVC, Spring moves requests around a dispatcher servlet, handler mappings, controllers and view resolvers [106]. Figure 5.9 illustrates how a request is moved until a response in generated by the view component.

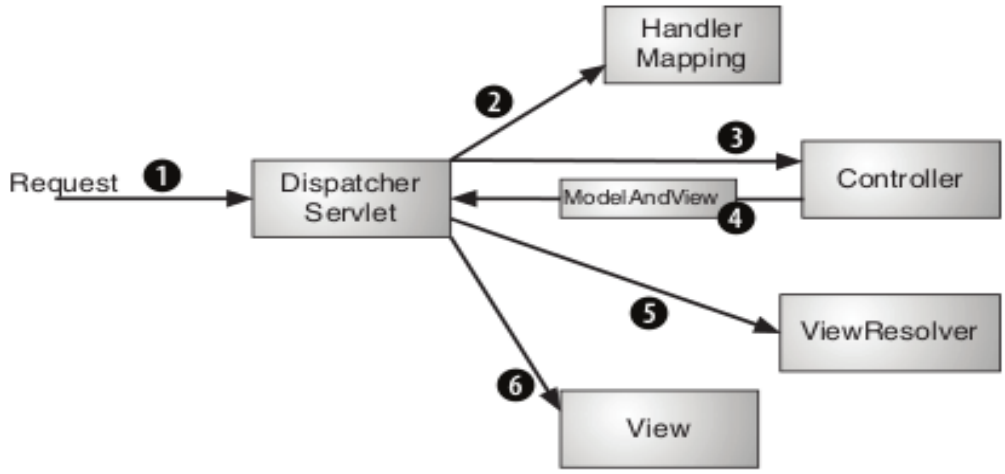

Figure 5.9: Spring MVC Design. Taken from [106].

The *DispatcherServlet* acts as the front controller servlet to funnel requests to a *Controller*.

A typical application may have several *Controllers* and the *DispatcherServlet* needs to decide which one to send a request to. In this case, the *DispatcherServlet* consults one or more *Handler* mappings, which check the URL carried by the request to find the next stop. Then the selected *Controller* processes the request by retrieving the payload (information submitted by a user) and performs the necessary actions. The logic performed by the *Controller* results in information, referred to as the *Model*, which must be sent back to the user. The *Model* can be presented in a *View* such as an HTML or JSP (Java Server Pages) page. The *Controller* packages the *Model* in a new object called *ModelAndView* which is sent to the *DispatcherServlet*. A *ModelAndView* object contains the *Model* and the information about a particular *View* that should render the results. In the end, the *DispatcherServlet* asks the *ViewResolver* to find the actual *View* (e.g. a JSP page) and use the data in a *Model* to render a page [106].

Spring MVC is available as OSGi components directly in the Spring distribution. Additional components need to be present in the OSGi container if JSP and JSTL (Java Standard Tag Library) are used.

In summary, when implementing an action-based web framework, an OSGi service can be configured with Spring DM within a Spring MVC *Controller* to display results of a web request in a *View* such as a JSP page [18]. In this case, the *Actions* component shown in Figure 5.8 would be handled by the *Controller* shown in Figure 5.9, and the business logic would be performed by referencing an OSGi service from the OSGi registry.

5.3.2.2 AJAX-Based Frameworks

AJAX (Asynchronous Javascript XML) frameworks are web frameworks that provide end-to-end support for implementing AJAX techniques from a JavaScript client on a web page to calling services on server side [18].

Integrating Spring DM with AJAX frameworks involves exporting OSGi services referenced through Spring DM as remote services with the exporting facilities of these frameworks. An exporter is a dedicated utility class that makes an existing service available remotely through a specific protocol [18]. Figure 5.10 shows the interaction between Spring DM and AJAX-based web framework.

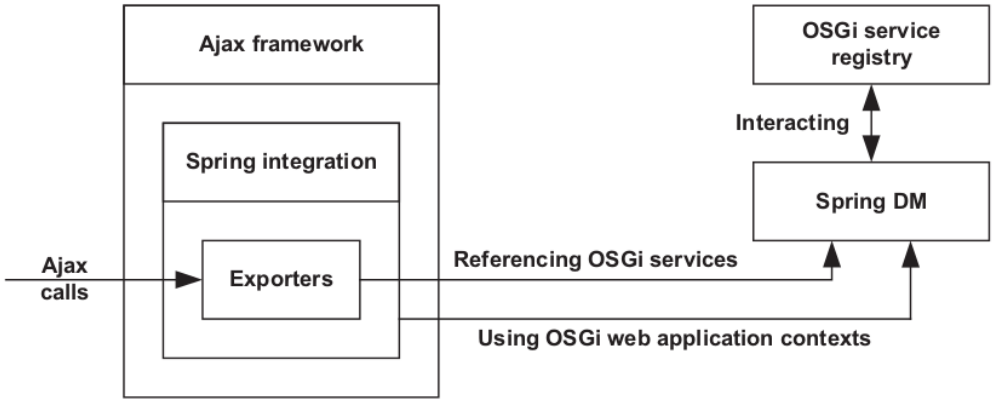

Figure 5.10: Spring DM and AJAX frameworks. Taken from [18].

An example of AJAX-based framework is the Google Web Toolkit (GWT) [35]. GWT is a development toolkit for building complex AJAX-based web applications using Java, which is then compiled to optimized JavaScript that runs across the major browsers [38].

Configuring GWT in an OSGi environment with Spring DM requires OSGi-fied versions of GWT and the GWT Widget library. The SpringSource EBR [96] does not provide the OSGi versions of GWT and GWT library. Instead, there is need to create the OSGi version using a utility tool such as Bndtools [11]. Bndtools is an open source utility that enables the creating and diagnosing of OSGi bundles.

In GWT applications, the exporter mechanism provided by the GWT Widget Server framework is used to expose Spring beans as remote GWT services based on Spring MVC. In general, GWT provides a Spring MVC controller for handling GWT Remote Procedure Calls (RPC). GWT RPC services are not web services. They are based on the Java servlet architecture and provide lightweight methods for transferring data between a server and the GWT application on the client [35].

In the OSGi environment, an OSGi service configured within Spring DM is exported as a remote GWT service and called from the client side through GWT remote interfaces. For the Javascript front end making the remote call, two interfaces are added with one corresponding to the remote interface of the service and the other being a remote asynchronous interface with a variant of the same methods exposed in the remote interface [18].

5.3.3 Web Services

Web service frameworks provide remote access to services using web technologies. According to Balani et al [10], a web service can be defined as a software component that provides a business function as a service over the web that can be accessed through a Uniform Resource Locator (URL).

Static HTML pages have been the web content that has evolved into more dynamic full featured web applications. A web service component is one step ahead of this web paradigm and provides only business service, usually in the form of raw XML data that can be digested by virtually all client systems [18]. In general, a web service can be thought of as a self contained, self describing, modular application that can be published, located and invoked across the web [10].

The key abstractions of web service frameworks are request handlers (Endpoints or Resources) [10]. These abstractions are responsible for handling requests and implementing functionality. Therefore, integrating Spring DM with these frameworks involves the use of OSGi services inside the action implementations [18]. Figure 5.11 shows how a web service framework integrates with Spring DM.

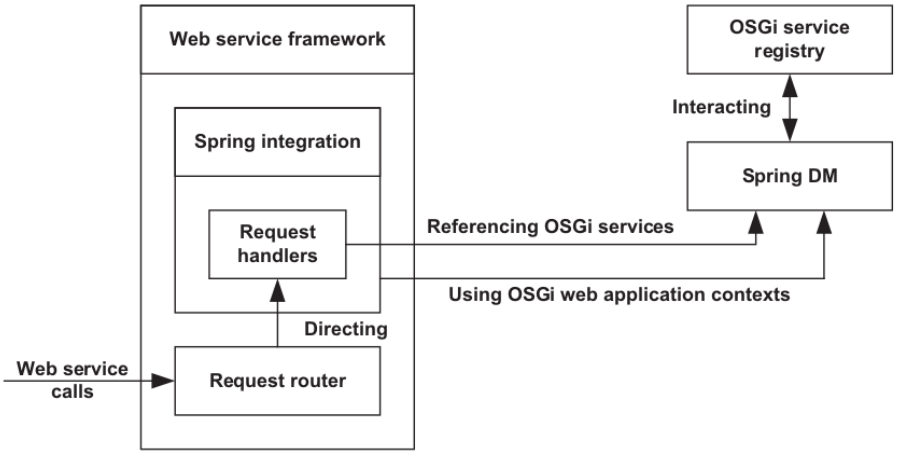

Figure 5.11: Spring DM and Web Service Frameworks. Taken from [18].

Two of the most widely used approaches for developing web services are SOAP (Simple Object Access Protocol) and the REST (Representational State Transfer) architecture style. The next sections explain the two architectures in detail.

5.3.3.1 SOAP Web Services

In SOAP-based web services, the service provider publishes the contract, Web Service Definition Language file (WSDL), of the service over the web where a consumer can access it directly or by looking it up in a service registry. The service consumer usually generates a web service client code from a WSDL file using the tools offered by the web service framework to interact with the web service [10].

In OSGi environment, there are several frameworks that can be used to implement web services using SOAP. One example is Spring WS [95] which is a Spring community project. Spring WS uses the contract-first approach, which consists of first implementing web service contracts using the WSDL, independent of the classes used to implement the services.

Another web service framework is Apache CXF [22]. CXF can be used to develop services using front end programming APIs like JAX-WS and JAX-RS [22]. These services can use a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, etc.

For SOAP web services in OSGi environments, CXF has a sub-project called Distributed OSGi (DOSGi) [22]. DOSGi implements the remote services functionality using web services, leveraging SOAP over HTTP and exposing the service over a WSDL contract .

5.3.3.2 RESTful Web Services

The term REST was first introduced in Roy Fielding's PhD dissertation [28], published in the year 2000, and it stands for REpresentational State Transfer [91]. The Abstractions that make a RESTful system are namely resources, representations, Universal Resource Identifiers (URIs) and the HTTP request types.

A RESTful resource is anything that is addressable over the Web. By addressable, it means that the resource can be accessed and transferred between clients and servers. Examples of resources can be a news story or a search result for a particular item in a web index, such as Google [91].

The representation of resources is what is sent back and forth between clients and servers. In general terms, it is a binary stream together with its metadata that describes how the stream is to be consumed by either the client or the server. A representation can take various forms, such as a text file, or an XML stream or a JSON stream, but has to be available through the same URI [91].

A URI in a RESTful web service is a hyperlink to a resource. The URIs use hyperlinks because the Web is used for creating the web services. Therefore, the hyperlinks (URIs) provide the only means for clients and servers to exchange representations [28, 91].

The HTTP methods of POST, GET, PUT, and DELETE are used to perform resource manipulations: creation, retrieval, update and deletion [91]. In this way, a resource such as a news story on the web can be read, edited or deleted using the HTTP methods.

There are several Java implementations of RESTful web services. Jersey is the reference implementation of Sun's Java API for RESTful web services which is commonly known as JAX-RS [91]. JAX-RS provides support in creating web services according to the REST architectural style [77]. The example open source frameworks that support JAX-RS are Restlet [86] and Apache CXF [22]. Both Restlet and Apache CXF provide OSGi versions of their distributions which can be downloaded from their respective websites.

5.4 The TeleWeaver Service Platform

In the previous sections, we have discussed application development and deployment in an OSGi environment through the creation of OSGi services and exposing them using the OSGi registry. We have also explored how web frameworks can be integrated in OSGi containers (using Spring DM) to develop web applications and web services. This section discusses TeleWeaver, an eServices platform based on OSGi technology which provides services that target marginalized communities in South Africa. In Chapter 2, TeleWeaver was introduced in a discussion of its business model and the targeted users and devices. We now focus on its technical details by discussing how the services are created, registered and consumed.

5.4.1 OSGi Container

TeleWeaver is a lightweight, custom built OSGi container which supports offline and online services access [72, 98]. It is based on Equinox OSGi framework [26]. The current release version of Teleweaver is 1.2.2 which uses Equinox version 3.6 (Release 3.6.0.v20100517).

5.4.1.1 Directory Structure

The directory for TeleWeaver contains an Equinox Java application file and folders named bundles, configuration, dropbox and logs (as shown in Figure 5.12). The Equinox application file, originally named *org.eclipse.osgi_3.6.0.v20100517.jar*, is renamed *Equinox.jar*. It is the main file that starts up the container.

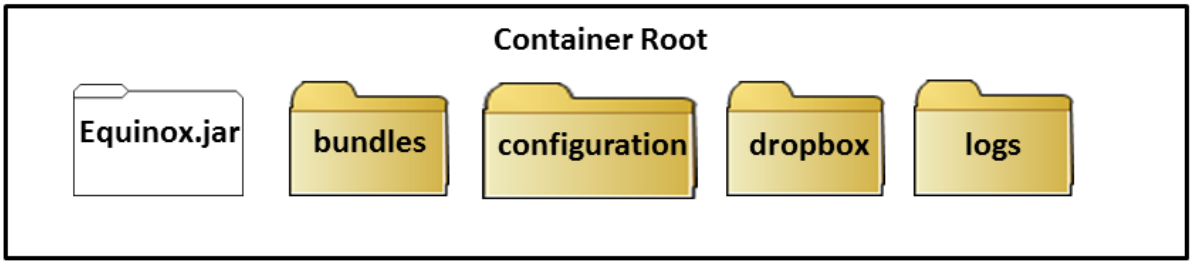

Figure 5.12: TeleWeaver Directory.

The *bundles* directory contains all OSGi bundles that are installed and started when the Equinox Container starts. The OSGi bundles are grouped in subfolders, for example, the *rhs* folder contains all custom made bundles by RHS, and the *Spring* folder contains all bundles for the Spring framework.

The *configuration* directory contains a configurations file called *config.ini* and a database configuration file *database.properties*. The *config.ini* has all the container settings such as the list of bundles that automatically start up when the container starts. The *database.properties* has the database connection settings which include the connection *driverClassName*.

The *dropbox* directory contains OSGi bundles that can be added dynamically to the container. In this directory, OSGi bundles can be un-installed (removed in the Container) and re-installed without requiring the restart of the container[99].

The *logs* directory stores all the log files with messages (e.g. error messages) generated by the container.

5.4.1.2 Starting the Container

To start TeleWeaver, a terminal session must be used to reach the location of the container. In TeleWeaver's root directory, issuing a command *java -jar Equinox.jar -console* starts the OSGi terminal and then prompts for a command to interact with Equinox. An example command is *ss,* which shows all bundles and their state as shown in Figure 5.13. Each bundle configured to start automatically (in the *config.ini*) resolves its dependencies and when successiful, ends up in *Active* or *Resolved* (if it is a supporting bundle or fragment) state.

Figure 5.13: Starting TeleWeaver.

5.4.2 Developing Services

To meet strict Java coding conventions and standards, RHS provides a standard guideline for TeleWeaver developers [99]. This section explains some of the guidelines that are followed when developing for TeleWeaver.

5.4.2.1 Project Structure

Maven is the recommended tool for creating projects which use a naming convention set by RHS. All project names are prefixed with *reedhousesystems-*, for example, a project on a tourism application is named as *reedhousesystems-tourism-api*. Packages are prefixed with *com.reedhousesystems-* (a service package uses *com.reedhousesystems.services.core-*). A web package uses the *com.reedhousesystems.web-* prefix. Figure 5.14 shows the project structure for TeleWeaver projects.

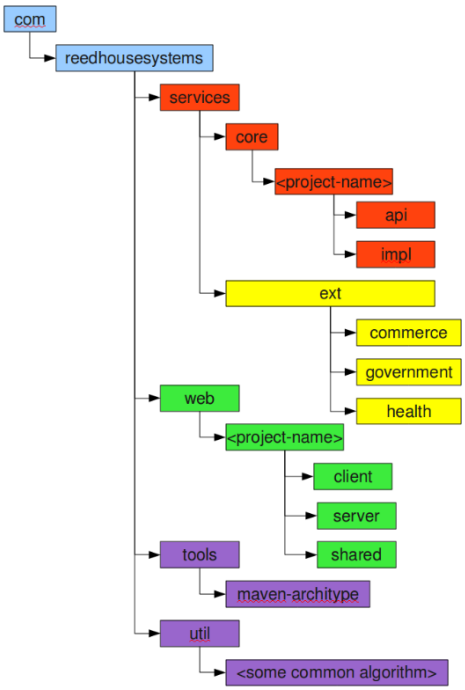

Figure 5.14: TeleWeaver Project Structure. Taken from [99].

There are custom made maven archetypes by the RHS which help in creating TeleWeaver projects with all relevant directories and default settings. An example of such archetypes is the *reedhousesystem-gwt-maven-archetype2* [99]. Running this archetype in Maven creates a TeleWeaver web project for a GWT application based on the structure shown in Figure 5.14.

5.4.2.2 The Service API Bundle

When developing OSGi services in Teleweaver, each service must have at least an API bundle and its implementation bundle. This design structure enables decoupling or a separation of concerns. In this way, client/web application developers are not concerned about the implementation of the APIs, but about the client applications they are developing [99].

An API bundle describes an interface for the interaction with a set of functions used by components of a software system. According to the RHS Wikidoc [99], an API can be language-independent and be called from several programming languages. This is a desirable feature for a service-oriented API that is not bound to a specific process or system and may be provided as remote procedure calls or web services.

The service API bundle can be created using the Maven quickstart archetype and specify

the project details (name, version, etc.) in the process. The generated POM file must specify the packaging type as bundle. The API bundle must have an interface class with methods that define the service functions.

5.4.2.3 The Service Implementation Bundle

A service implementation bundle can be created using the Maven quickstart archetype in the same way as has been explained for the API bundle. The bundle requires an import of its corresponding API bundle package by setting it as a dependency in the POM. The POM must have the coordinates for all dependencies (libraries) that are required by the classes inside the bundle. The packaging type of the project must be set as a bundle.

The service implementation bundle has a class (or classes) with code that perform the business logic for the methods listed in the API bundle's interface class.

5.4.3 Registering a Service

To add a service in the TeleWeaver service registry, the required settings must be defined in the *application-config.xml* file: located in the *META-INF/spring* folder of the service implementation bundle. A service is registered and exposed using the *osgi:service* element followed by an assigned service reference name and the interface class of the API bundle. An example is shown below:

<osgi:service ref="tourismService"

interface="com.reedhousesystems.services.core.tourism.TourismService" />.

5.4.4 Referencing a Service

To reference a service in TeleWeaver, a similar approach to the registration process is used by adding the settings in the *application-config.xml*. In this process, the *osgi:reference* tag is used. An example is shown below:

<osgi:reference ref="tourismService"

interface="com.reedhousesystems.services.core.tourism.TourismService" />.

We have looked at the TeleWeaver platform as an e-services enabler that is based on OSGi technology. The next section focuses on mobile phone communication technologies that can be integrated with TeleWeaver to provide the OSGi services to mobile phone users.

5.5 WAP and SMS Solutions

This section covers the WAP and SMS technologies, which are important to mobile phone subscribers on top of the normal voice communication.

5.5.1 The Wireless Application Protocol

The Wireless Application Protocol (WAP) is a collection of various languages, tools and an infrastructure for implementing services for mobile phones [30]. WAP makes it possible to implement services similar to the World Wide Web (WWW). The entry of smartphones removes the need for WAP, but this technology is still relevant especially in developing nations where feature phones are the dominant mobile phones as explained in Section 2.4.2.

WAP does not bring the existing content of the WWW directly to the phone. The main problem is that WWW content is mainly in the form of HTML pages: which require fast connections, fast processors, large memories, and often also fairly efficient input mechanisms [110, 30]. However, most feature phones have very slow processors, very little memory and intermittent bandwidth. With HTTP being heavy for wireless use, WAP defines a completely new markup language, the Wireless Markup Language (WML), which is simpler and much more strictly defined than HTML [30].

Figure 5.15 illustrates how content from a server is served to a mobile phone. The WAP gateway communicates with the phone using the WAP protocol stack, and translates the requests it recieves to HTTP. The WAP protocol stack transports requests for pages from the phone to the gateway, and the actual pages (possibly converted to a binary form) back to the phone. In this way, the content providers can use any HTTP server and utilize existing knowledge about HTTP service implementation and administration [110]. In addition to protocol translations; the gateway also compresses the WML pages to save bandwidth on the air and to further reduce the phone's processing requirements [30].

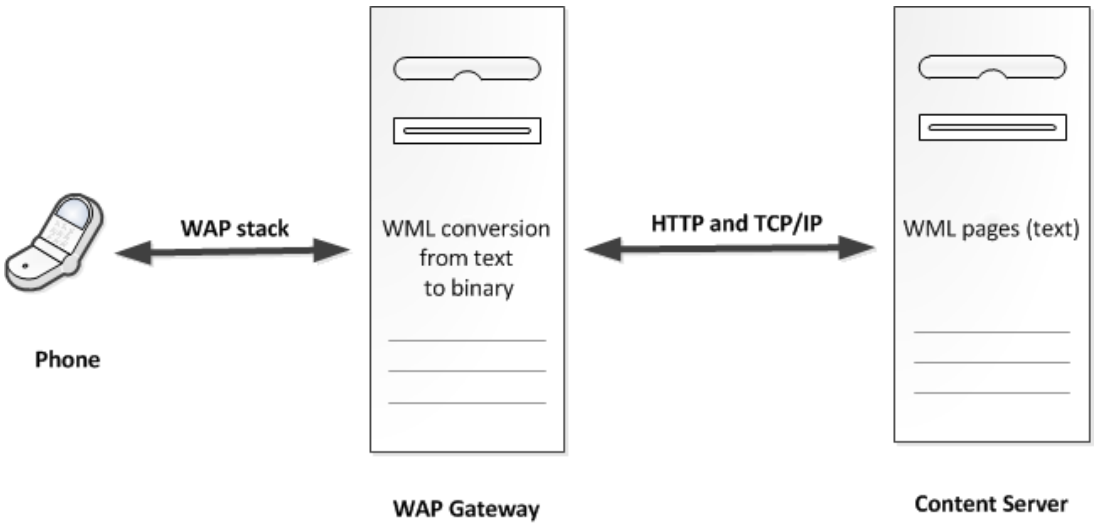

Figure 5.15: WAP Architecture. Adapted from [110].

WML is a simple markup language based on XML and is used to mark the contents of the file as actual text, title, hyperlinks, etc. A WML page is similar to a deck of cards with a single card displayed at a time by the phone. It is possible to switch between cards on the same deck quickly, since the whole deck is downloaded at once. A WAP application might fit onto one card, or be divided into several, depending on its size and how big a deck the phone can accept [110].

5.5.2 SMS

SMS is a way to send short (160 character) messages from one GSM phone to another [30]. It can also be used to send regular text as well as advanced content like operator logos, ringing tones, business cards and phone configurations [30].

SMS services are content services initiated by an SMS message to a phone number used by a content server to respond with requested content. When SMS services are used, the client (mobile terminal) sends an SMS message to a Short Message Service Center (SMSC)[30]. A SMSC is responsible for handling the SMS operations of a wireless network [42]. The SMSC then forwards the SMS message to the destination.

An SMS message may need to pass through more than one network entity (e.g. SMSC and SMS gateway) before reaching the destination [42]. The main duty of an SMSC is to route SMS messages and regulate the process. It uses the store and forward mechanism such that if the recipient is unavailable, the SMSC stores the message and forwards it once the recipient becomes available within the message's validity period [42].

Different SMSCs use specific protocols, for example, a Nokia SMSC uses CIMD protocol [30]. Therefore, an SMS gateway is used to handle connections with SMSC and to relay or intercept them onward in a unified form.

5.5.3 Kannel

Kannel is an open source WAP and SMS gateway [30]. It is made up of three different components:

1. Bearerbox - the direct interface to mobile phones which accepts SMS and WAP messages and sends them to the other two boxes.
2. Smsbox - handles the SMS gateway functionality.
3. Wapbox - handles the WAP gateway functionality.

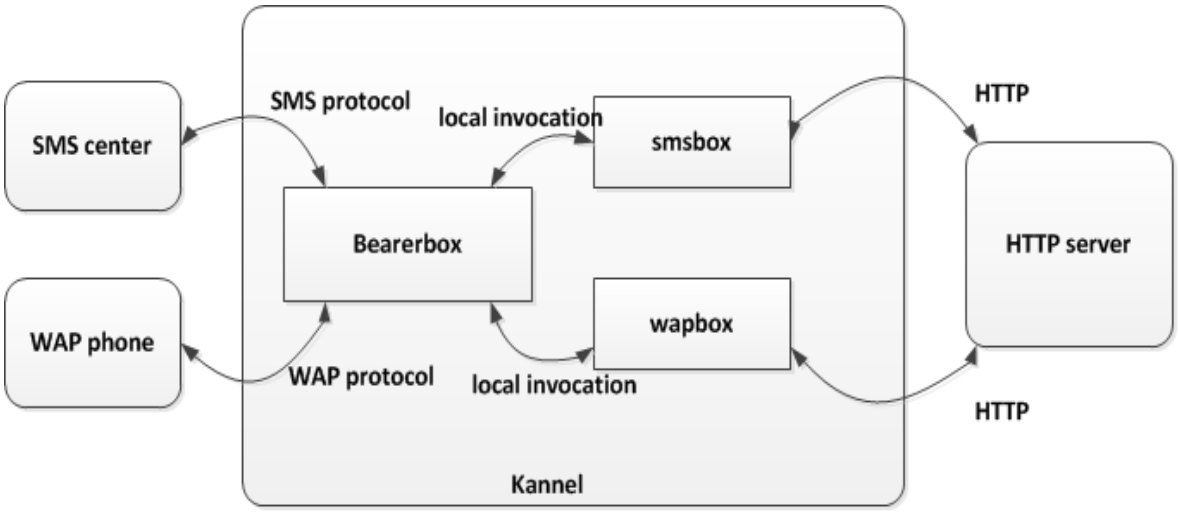

Figure 5.16: Bearerbox with Wapbox and Smsbox. Adapted from [110].

Figure 5.16 shows how the bearerbox works with both the smsbox and wapbox. There can be only one bearerbox, but any number of smsboxes and wapboxes. Having multiple smsboxes or wapboxes can be beneficial when the load is high [110, 30].

Kannel as a WAP gateway operates in between a phone and a content server, as shown in Figure 5.15, to perform the protocol translations (between WAP and HTTP) and compressions of the WML pages. The wapbox is configured to translate WML content to a binary format which is optimal for wireless communication. In this scenario, Kannel

acts as a proxy server by handling HTTP requests and responses on behalf of the mobile phone [110].

As an SMS gateway, Kannel is used to handle connections with SMS Centres (SMSCs) and to relay them onward in a unified manner [30]. Figure 5.17 shows Kannel as an SMS gateway providing content from a provider to an SMS client (mobile phone).

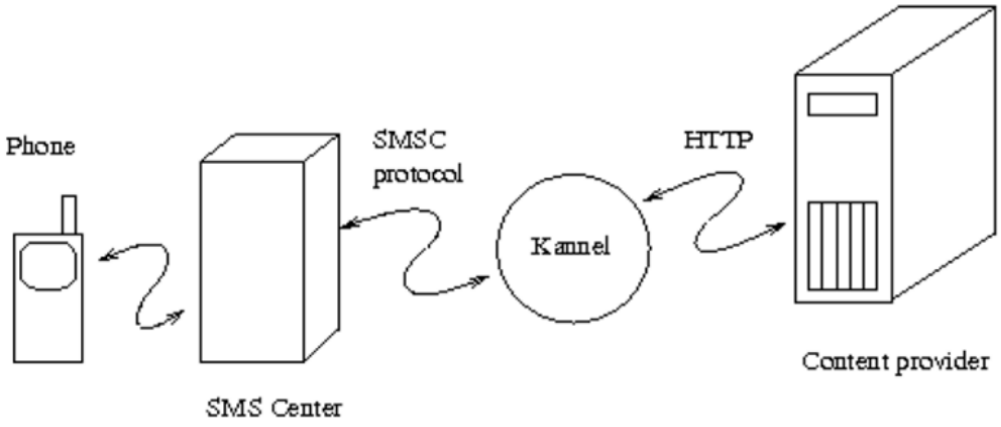

Figure 5.17: Kannel SMS Gateway. Taken from [30].

Kannel provides a mini-HTTP server that intercepts HTTP requests. Configuration settings in the smsbox section determine the hostname and port number it listens to.

In an SMS service, unique services are distinguishable by the first word in the message and by the number of arguments that follow it. If a particular message is matched against a service, Kannel fetches content through an HTTP request to the URL indicated in the service configuration [30].

5.6 Summary

In this Chapter, we have discussed various technologies that are relevant for the implementation of our proposed DRS. We explored how a service can be developed and deployed in an OSGi environment so that clients (e.g. web clients) can access it. Then, the TeleWeaver platform which is based on OSGi technology, has been discussed in terms of how e-services are developed and deployed. In conclusion, open source solutions for WAP and SMS technologies using the Kannel system have been explained in terms of how they operate and communicate with content servers and mobile phone clients. The next chapter covers how the DRS was implemented based on the technologies that have been presented in this chapter.

Chapter 6

System Implementation

The previous chapter explained the technologies that are relevant in the implementation of our DRS. This chapter dicusses how the different components of the system were implemented. It presents the details about the development of the DRS 's service bundles that make up the system in TeleWeaver. Thereafter, it explains how the service is exposed to mobile phone clients using the Kannel system.

6.1 The Development Environment

To implement our system in TeleWeaver, we followed the guidelines in developing for OSGi environments that are provided in the wiki document [99] by RHS. The wiki also provides the software and hardware requirements that assist in selecting the appropriate tools for the implementation process.

6.1.1 Hardware

The hardware that was used for the implementation of the system was as follows:

1. A desktop computer as a TeleWeaver developer machine installed with the necessary software for development. This machine also ran a local TeleWeaver test server (RHS Test Server) for tests that were performed during the development process.
2. A server machine to provide an environment for tests on the developed application.
3. GSM modem (Wavecom Fastrack Supreme 10). This is a GSM and GPRS (General Packet Radio Service) modem that comes with a slot for a SIM card and connects to a computer using the serial (console) port.

6.1.2 Software

The development machine had the following software installations:

1. Ubuntu desktop version 11.10 operating system [56].
2. SpringSource Tool Suite 2.9.2.RELEASE, for Spring and OSGi application development [20].
3. TeleWeaver container, RHSServer1.2.1 version [99].
4. Apache Maven version 2.2.1 [8].
5. MySQL community server version 5.1.63 [81].
6. Hibernate version 3.5.1-Final [19].

The server machine had the following software installations (relevant for our tests):

1. Ubuntu server 10.04.4 LTS operating system [56].
2. Kannel bearerbox version 1.4.3 [36].
3. TeleWeaver container, RHSTestServer1.2.2 version [99].
4. MySQL community server version 5.1.63 [81].

6.2 Developing the Service

The implementation process started with the development of the DRS service bundles (API and implementation) in TeleWeaver. This section discusses how the service bundles were implemented.

6.2.1 Overview of the Service Bundles

There are different design choices for implementing OSGi enterprise applications based on how bundles are used with the enterprise application layers (web, business or data access) [18]. An application can have one bundle for each layer that has only classes (e.g. domain and web) or it can be designed to use two bundles for each layer that has interfaces and class implementations (e.g. data access and business layers).

The important design practice is to separate static components from dynamic components [18]. Static components are bundles that define APIs by exporting interface-based Java

packages. These bundles do not contain a Spring application context (or a BundleActivator) and do not register or reference any OSGi services. Dynamic components are bundles that import Java packages from static bundles, provide implementations, and usually register OSGi services. These can also be called Spring-powered bundles [18].

We followed the design pattern used by RHS in TeleWeaver, as explained in chapter 5. Hence, the implementation design for the DRS was based on two bundles (for the API and the implementation) as shown in Figure 6.1.

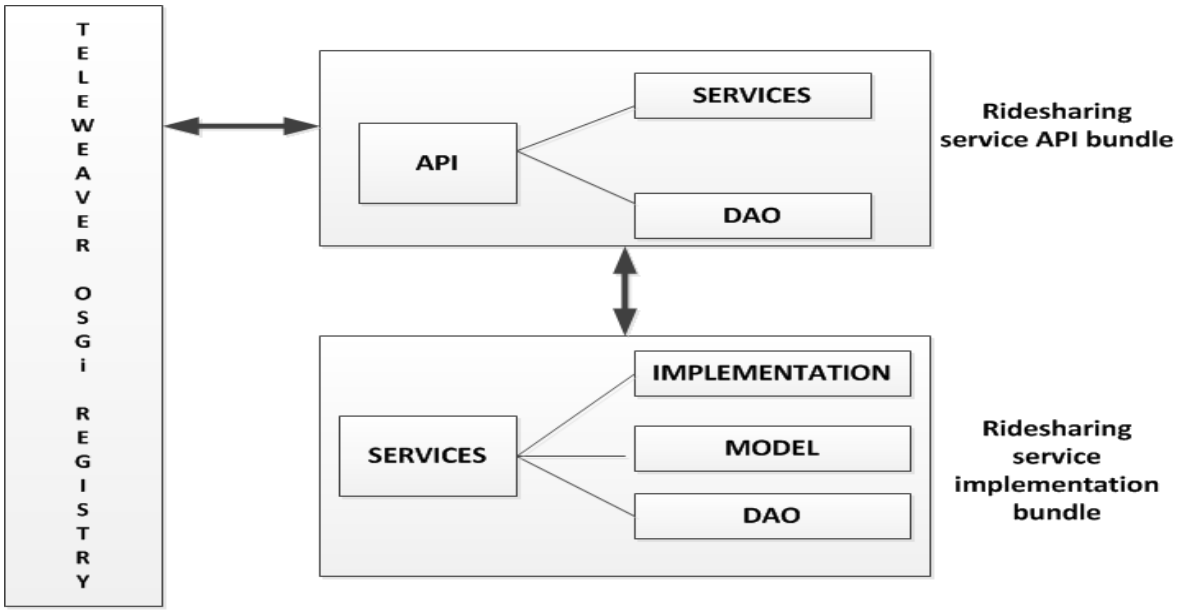

Figure 6.1: Overview of the Service Bundles.

This design ensures that the static components are stable (changing infrequently) whereas dynamic components can be frequently updated and benefit from OSGi dynamic features such as on-the-fly service updating [18].

The implementation design uses the spring framework's Data Access Object (DAO) pattern for data access from a database [105]. Hence, all persistence operations are performed using specific classes called DAO classes.

In Figure 6.2, the ridesharing service objects in the implementation bundle can only access the DAOs through interfaces in the ridesharing service API bundle. This separates the persistence code from the business logic, resulting in the service objects not being coupled to specific data access implementation (e.g. an ORM solution or JDBC connection details).

The next sections will discuss the two bundles that make up the DRS in TeleWeaver.

6.2.2 Service API Bundle

The API bundle was developed using the following steps:

1. Creating a Maven project in SpringSource Tool using the *maven-archetype-quickstart* archetype. Using the RHS naming conventions, explained in Section 5.4.2.1, the API bundle manifest file (*Manifest.mf*) was set as follows:

   - Bundle-Name: *reedhousesystems-ridesharing-api*
   - Bundle-Version: *1.2.2*
   - Export-Package-Name = *com.reedhousesystems.services.core.ridesharing*

2. Creating an interface class named *RideshareService* in the package called *com.reedhousesystems.services.core.ridesharing*. The interface class has the methods that define the ridesharing operations of the DRS.

3. In the same package, an interface class for the DAOs was created and named *RideshareDao*. This interface class has the methods for the data access operations.

4. Creating Data Transfer Objects (DTOs) that carry data across the application layers, for example, between the persistence and business logic layers. DTO is a simple Plain Old Java Object (POJO) only containing simple data fields to enable lightweight data transfers [38]. The DTOs in the API bundle were created based on the domain model entity objects which are explained in Section 6.2.3.4. More details about DTOs are provided in Section 6.2.3.5.

The POM file (pom.xml) of the project imports two dependencies: a custom RHS exceptions library (*reedhousesystems-exceptions*) and the J-Unit library for J-Unit tests. It includes a directive to export the bundle's package that contains the interface classes and DTOs. This ensures that other bundles that import the API bundle, as a dependency, can use the package contents.

After compiling and installing the project, an OSGi bundle called *reedhousesystems-ridesharing-api.jar* was generated. This bundle was added in the *rhs* folder of the TeleWeaver directory. The bundle file location details were added in the configuration file (*config.ini*) to ensure that the bundle starts in *Active* state whenever TeleWeaver starts up.

The API bundle is mainly concerned with the definition of the ridesharing service that is published on the OSGi registry. It is simple in structure and contents compared to the service implementation bundle, which is discussed in the following section.

6.2.3 Service Implementation Bundle

The service implementation bundle was created using similar steps as has been explained for the API bundle. The manifest file of this bundle has some of the properties which were set as follows:

- Bundle-Name: *reedhousesystems-ridesharing*
- Bundle-Version: *1.2.2*
- Import-Packages: *com.reedhousesystems.services.core.ridesharing,* ... (and a list of other imported packages)
- Bundle-Activator:

  com.reedhousesystems.services.core.ridesharing.impl.RideshareBundleActivator

The *Import-Packages* property contains a list of packages required by the classes in the bundle, and are specified through the import directive in the POM file.

6.2.3.1 Packages

The bundle has 3 packages as shown in the implementation design in Figure 6.1 above. The packages are set as follows:

1. *com.reedhousesystems.services.core.ridesharing.impl.*

   Contains the bundle activator class called *RideshareBundleActivator* and a service implementation class named *RideshareServiceImpl.* The *RideshareServiceImpl* class implements the methods in the *RideshareService* interface class of the service API bundle. The bundle activator class is used to initialize and finalize the bundle [18]. It has the *start* and *stop* methods that perform bundle-specific activities in the bundle's start and stop processes respectively. In our case, the *start* method is used to display information of a successiful start on the OSGi console whenever the bundle is started.

2. *com.reedhousesystems.services.core.ridesharing.dao.*

   Contains a DAO class named *HibernateRideshareDAO.* This class implements the methods of the *RideshareDao* interface class of the service API bundle.

3. *com.reedhousesystems.services.core.ridesharing.model.*

   Contains the domain model composed of entity classes that were created based on the class diagram that is presented in Section 4.1.4.

6.2.3.2 Spring Configurations

Chapter 5 explained how Spring DM bundles provision a Spring application context to an OSGi bundle for spring framework features such as DI. The implementation bundle of the DRS has a Spring configuration file called *application-config.xml,* which is located in the *src/main/resource/META/spring* folder. The Spring configurations can be separated in different files for easy management of the configurations. In this way, Spring beans can be configured in a separate file called *beans.xml* file, the datasource configured in a *datasource.xml* file and the hibernate configurations set in a *hibernate.xml* file. In our implementation, the *application-config.xml* file contains all the configurations in separate sections of the file. Hibernate was our selected ORM solution and the relational database was the MySQL database system. The configurations were made as follows:

1. Data source properties

   This part configures the reference to a *Datasource*. The *Datasource* uses a pool of connections using the *org.apache.commons.dbcp.BasicDataSource* class of Java. The database properties and their values (e.g. *driverClassName*, *username*, *password*, etc.) are provided in the *database.properties* file which is explained in Section 5.4.1.1.

2. SessionFactory

   SessionFactory is responsible for opening, closing and managing Hibernate sessions [106]. The *SessionFactory* in the DRS uses contextual sessions which, according to Walls et al [106], were introduced in Hibernate 3 as a way in which Hibernate itself manages one session per transaction. This decouples the DAO classes from the Spring API. The *SessionFactory* was configured using the *hibernate3.annotation.AnnotationSessionFactoryBean* class. Part of the *application-config.xml* file that contains our Hibernate settings for the *sessionFactory* is shown in Figure B.3 of Appendix B.2.

3. Bean declarations

   Spring beans are declared and wired together with other beans using their properties. This is part of the DI to enable an object to use another object in a Spring application. In our implementation, the *datasource* bean is declared and wired into a *sessionFactory* bean as its property. Then, the *sessionFactory* is wired into the *hibernateRideshareDao* bean*:* the DAO implementation class of the service implementation bundle *(reedhousesystems-ridesharing*). The bean declarations and their wiring can be seen in Figures B.2 and B.3 of Appendix B.2.

4. OSGi service registration

   The ridesharing service is registered on the TeleWeaver's OSGi registry using the *osgi:service* tag and the value of the interface class set as

   com.reedhousesystems.services.core.ridesharing.RideshareService. This is shown at the end of the configurations in Figure B.3 of Appendix B.2.

6.2.3.3 Domain Model Classes

Entity classes were created using the class diagram presented in Section 4.1.4. The entity classes are persisted using annotations based mapping by Hibernate [93]. The annotations enable mapping of collections and associations of objects in the domain model to their relational database representations. The entity classes in the domain model package include the *Person*, *Trip, RoutePoint, Vehicle* and *Hitchhiker* classes.

6.2.3.4 Business Logic

The application uses DTOs, as mentioned in Section 6.2.2, for the transfer of method parameters that involve domain model objects. The DTO classes located in the API bundle have their names derived from the domain model entity classes, located in the implementation bundle (e.g *personDTO* for *person* class). DTOs enable the transfer of subset properties of an object for a particular task. The transfer of lightweight objects in the form of DTOs helps to improve the performance of an application [35]. Figure 6.2 illustrates our implementation of DTOs to persist data in the MySQL database. The data transfers between the user interaction and business layers involve the DTOs, while between the business and persistence layers the actual objects are involved. The Java class *org.apache.commons.beanutils.BeanUtils* is used to copy properties of a DTO to its associated object and vice versa.

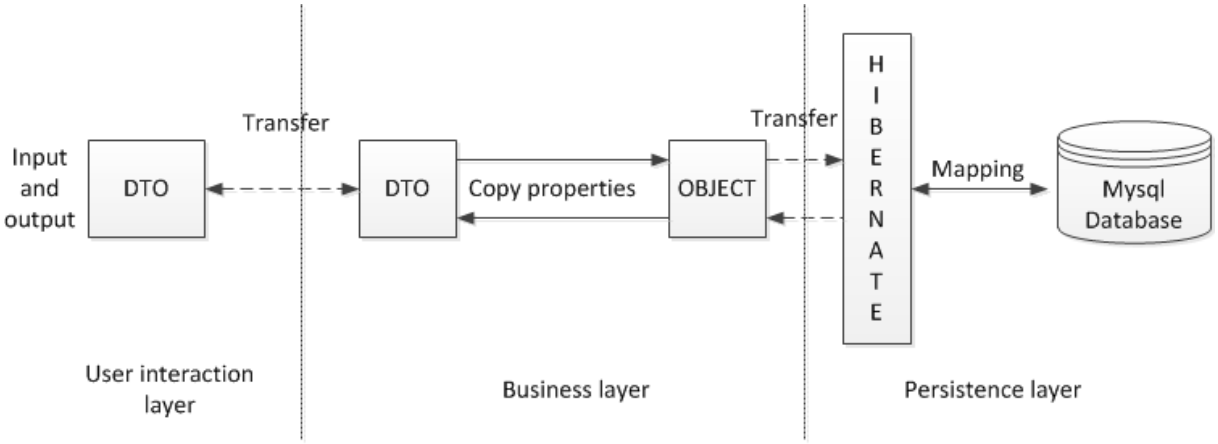

Figure 6.2: Data Transfer Objects.

The ridesharing functions of the DRS are implemented in the methods of the *RideshareServiceImpl* class of the implementation bundle. The main functions were implemented as follows:

1. Registration of users

   The *registerUser* method uses the *personDTO* as a parameter with a new user's details such as *username*, *password*, *phonenumber*, etc. The properties of the *personDTO* are assigned their values by the application front end interface. A new user is persisted by copying the *personDTO* properties to an instantiated *person* object which has Hibernate annotations to save it as a record in the database table called *Person*.

2. Register a vehicle

   The requirements specifications in Section 4.1.3.1 state that only a person in a driver role can create a trip. As such, the driver is required to register a vehicle under his/her profile. The *addVehicleToPerson* method uses the *vehicleDTO* parameter which has properties of a vehicle such as registration number, model, seating capacity and colour.

   The registration number of a vehicle must be unique and is checked for every new vehicle registration. A driver can register more than one vehicle under his/her profile, and a vehicle can be used by more than one user in different rideshare trips, as explained in the class diagram in Section 4.1.4.

3. Creating a trip

A trip is created using the *createTrip* method which accepts three parameters *username, vehRegNo and TripDTO.* A driver is required to provide a username, vehicle registration number and the new trip details. The trip details in a *TripDTO* are the final destination and the hitch-hiker's preferred pick-up point (*location name*, *street* and *nearest landmark*). The method also checks if the vehicle registration represents a vehicle that is registered under the driver's profile before creating a trip.

A location name or a short-code (used in hitch-hiking) can be used when setting the destination and pick-up location names. When a short-code is used, the system searches for the location name in a table of registered locations (*RoutePoint*). If the short-code entered does not match any registered location then the user is requested to register the new location details.

The *createTrip* method generates a security code (*TripCode*) using a private method called *generateTripCode.* The *generateTripCode* uses a trip's destination short-code and a random number generator function to create a secret code, *TripCode.* An example of a *TripCode* is *PE283461*, generated for a trip with the destination set as Port Elizabeth. Therefore, all trips to Port Elizabeth are identified by the prefix *PE-* followed by a random six digit number. The *TripCode* can be used by a driver to verify unknown ride partners at the pick-up point.

4. Offering a trip to hitch-hikers

   The *rideOffer* method enables a driver to offer a trip to hitch-hikers with a set of conditions. The conditions are: available seats, rideshare cost, ride offer validity and any additional conditions (optional).

   The method gets the conditions in the form of parameters: seats (integer), *TripCode* (string), *Cost* (double), *Time_Window* (interger) and *Conditions* (string). The most recently created *Trip* object by the driver is offered at a *RoutePoint* (pick-up point of the *Trip*). As a result, the *Trip* is available to any hitch-hiker available at the *RoutePoint* and seeking a ride to a same destination as that of the offered *Trip.* The implementation of the matching algorithm is discussed in point number six of this section.

   The timestamp of a ride offer is recorded and the *Time_Window* (integer), which represents the offer duration in minutes (e.g 10 = 10 minutes), is added to calculate the expiry timestamp of the ride offer. Figure 6.3 shows how ride offers are logged with a validity period calculated from a specified *Time_Window*, in the MySQL database. Any ride request by a hitch-hiker that arrives outside the validity period

of a ride offer is not considered for matching.

```
mysql> select * from TRIP_ROUTEPOINT;
+-------------+---------------------+--------------+---------------------+-------------+------+---------+------
| TripRtPt_Id | Offer_Time          | Offer_Window | PickupTimeMax       | Offer_Seats | Cost | Trip_Id | Point
+-------------+---------------------+--------------+---------------------+-------------+------+---------+------
|         109 | 2012-09-16 20:18:52 |           30 | 2012-09-16 20:48:52 |           3 |   80 |      82 |
|         110 | 2012-09-18 10:56:19 |           30 | 2012-09-18 11:26:19 |           0 |   80 |      83 |
|         111 | 2012-09-18 15:48:04 |           10 | 2012-09-18 15:58:04 |           4 |   80 |      84 |
|         112 | 2012-09-18 16:34:53 |           10 | 2012-09-18 16:44:53 |           3 |   80 |      85 |
|         113 | 2012-09-19 11:09:06 |           10 | 2012-09-19 11:19:06 |           3 |   50 |      86 |
|         114 | 2012-09-19 13:31:31 |           20 | 2012-09-19 13:51:31 |           5 |   85 |      87 |
|         115 | 2012-09-19 20:36:20 |           10 | 2012-09-19 20:46:20 |           5 |   65 |      88 |
```

Figure 6.3: Registering a Ride Offer at a Route Point.

5. Ride request by a hitch-hiker

   A ride request by a hitch-hiker is handled by the *seekRide* method which gets a *username* and a *HitchhikerDTO* parameters. The *HitchHikerDTO* carries the details of a hitch-hiker such as destination and the hitch-hiking spot where he/she is standing. The destination and hitch-hiking spot location names can use short-codes and are handled as explained in point number three, for the *createTrip* method.

   The ride request is matched against ride offers available at the hitch-hiker's specified hitch-hiking spot *(RoutePoint)*. Figures 6.4 and 6.5 show how records of hitch-hikers are saved in the database and ride requests at a hitch-hiking spot (*RoutePoint*) respectively.

```
mysql> select * from HITCHHIKER;
+----------+-----------+--------------------+--------------+--------------+------------+----------------+----------
| Hiker_Id | Person_Id | Hitchhiking_Dest   | Transit_Dest | Request_Seats | Travel_Day | Curr_Loc_Name  | Curr_Loc_Street
+----------+-----------+--------------------+--------------+--------------+------------+----------------+----------
|       85 |        44 | Port Elizabeth     | not available |            1 | 2012-09-16 | Grahamstown    | Beaufort
|       86 |        45 | King Williams Town | not available |            3 | 2012-09-18 | Port Elizabeth | albany
|       87 |        44 | Port Elizabeth     | not available |            1 | 2012-09-18 | Grahamstown    | beaufort
|       88 |        46 | Port Elizabeth     | not available |            2 | 2012-09-18 | Grahamstown    | Beaufort
|       89 |        45 | Port Elizabeth     | not available |            1 | 2012-09-19 | Grahamstown    | beaufort
|       90 |        45 | King Williams Town | not available |            2 | 2012-09-20 | Grahamstown    | beaufort
|       91 |        45 | King Williams Town | not available |            1 | 2012-09-21 | Grahamstown    | African
```

Figure 6.4: Hitch-Hiker Record.

```
mysql> select * from HITCHHIKER_POINT;
+---------------+---------------------+----------+----------+
| HikerPoint_Id | CheckInTime         | Hiker_Id | Point_Id |
+---------------+---------------------+----------+----------+
|            82 | 2012-09-16 20:29:42 |       85 |       47 |
|            83 | 2012-09-18 10:51:58 |       86 |       58 |
|            84 | 2012-09-18 15:10:13 |       87 |       47 |
|            85 | 2012-09-18 16:29:01 |       88 |       59 |
|            86 | 2012-09-19 11:06:56 |       89 |       60 |
|            87 | 2012-09-20 21:26:04 |       90 |       65 |
|            88 | 2012-09-21 13:33:38 |       91 |       72 |
|            89 | 2012-09-25 13:14:39 |       92 |       83 |
|            90 | 2012-09-27 15:06:10 |       93 |       85 |
|            91 | 2012-09-27 15:11:43 |       94 |       84 |
```

Figure 6.5: Hitch-Hiker Request at a Route Point.

6. The matching algorithm

   The matching algorithm considers both scenarios of a driver ride offer and a hitch-hiker ride request arriving at different times (ride offer before ride request or vice versa) as discussed in the design of the DRS in Section 4.1.3.2.

   When matching a ride offer, the search for hitch-hikers is performed on the objects of the *HitchhikerPoint* class (representing hitch-hikers at a *RoutePoint*). The matching algorithm puts a priority on hitch-hiker requests that have the same pick-up point (location, street and nearest land mark) as specified by the driver (these are referred to as the *BestMatch* records). If the *BestMatch* results do not meet the number of offered seats, then a second matching is performed for any hitch-hiker along the street of the driver's specified pick-up location.

   The list of matching hitch-hikers is sorted by the timestamp of their ride request (The *CheckInTime* at the hitch-hiking spot shown in Figure 6.5). The earliest ride request ranks at the top of the list as explained in the design of the DRS.

   When matching a ride request, the search for a ride offer by a driver is performed on the objects of the *TripRoutePoint* class (representing offered trips at a *RoutePoint*). A single ride offer is selected from a possible list of valid driver ride offers. A valid ride offer with the closest departure time, based on its *PickupTimeMax* value, is selected for the ride request.

   If no ride offer is available, the hitch-hiker request waits until it is cancelled by the owner or by the end of the day (by default). If a *BestMatch* for a ride request is not found, the matching algorithm targets ride offers (to the same destination) with a pick-up point specified along the same street as the ride request. In this scenario,

the DRS suggests a possible ride partner to a driver at a particular point along the street of the specified pick-up point. Then, the driver can decide whether or not to include the hitch-hiker in the trip.

7. Message notifications

   The message alerts sent by the DRS to the users use SMS. The *SMSClient* class of the implementation bundle has a method called *sendSMS,* which gets a phone number (or a list of phone numbers) and the information for the SMS message. The *sendSMS* method uses the Apache Commons client *(HTTPClient)* to send an SMS message delivery request to the SMS gateway. The implementation of the SMS gateway and its communication with the OSGi bundles in TeleWeaver are discussed in Sections 6.3 and 6.5 respectively.

We have discussed the implementation of an OSGi bundle that performs the required operations of the DRS. The next section discusses how the ridesharing service of the DRS was made to be accessible by mobile phone clients using web browsers and SMS.

6.3 Kannel SMS and WAP Gateway

This section explains how the Kannel system was configured as an SMS and WAP gateway for the accessibility of the ridesharing service by mobile phones.

6.3.1 Installation

Kannel can be installed either from a source code package or by using a pre-compiled binary version [30]. For our installation, the Ubuntu software management primitives were used, using the command *sudo apt-get install kannel kannel-dev.*

The installation has a default configuration file, called *modems.conf,* which defines all types of modems used by Kannel to connect to an SMSC.

6.3.2 Configurations

We configured the bearerbox, smsbox and wapbox in a single file, *kannel.config.* The configurations consist of groups of configuration variables and are explained in the next few sections.

6.3.2.1 Bearerbox

We configured the *core* group for general bearerbox settings as shown in Figure 6.6:

```
group = core
admin-port = 13000
admin-password = k@nn3l
smsbox-port = 13002
wapbox-port = 13005
wdp-interface-name = "*"
log-file = "/tmp/kannel.log"
log-level = 2
box-deny-ip = "*.*.*.*"
box-allow-ip = "127.0.0.1;192.168.10.11;146.231.121.165;146.231.122.16"
#store-location = "/var/log/kannel/kannel.store"
http-proxy-host = ""
http-proxy-port = ""
http-proxy-username = ""
http-proxy-password = ""
http-proxy-exceptions = "localhost,192.168.10.11"
http-proxy-exceptions-regex = "146.231.*.*"
```

Figure 6.6: Bearerbox Configuration Settings.

The *core* group instantiates the bearerbox with global settings for the system. The smsbox and wapbox are assigned port numbers (can be any number) 13002 and 13005 respectively to which they can be connected. The admin-port key identifies the port that the bearerbox binds to for HTTP administration commands. A list of allowed IP addresses can be set using the box-deny-ip and box-allow-ip for controlled access to the system.

6.3.2.2 SMS Center

For the kannel system to access the SMSC for SMS messages, a group called *smsc* was added and configured as shown in Figure 6.7. The smsc key is given the value *at*, however any string is acceptable. The smsc-id key is an optional id. This 'id' is written into log files and can be used to route SMS messages, and to specify the used SMS-service [30]. The *modemtype* key value is defined in the included file *modems.conf.* Instead of using a specific value, the value of this key was set to *auto* for Kannel to automatically detect the type of modem that is attached.

```
group = smsc
smsc = at
smsc-id = AT
modemtype = auto
device = /dev/ttyS0
validityperiod = 167
#for mtn
sms-center = +27831000113
# for vodacom
#sms-center = +27829129
include = "/etc/kannel/modems.conf"
```

Figure 6.7: SMSC Configuration Settings.

6.3.2.3 Smsbox

The Kannel system can have as many smsboxes in a distributed setup. We defined a single group (*smsbox*) for the smsbox configurations as shown in Figure 6.8. The *bearerbox-host* key identifies the IP address of the machine that the bearerbox is running on. Therefore, to run a distributed deployment of the Kannel system, smsbox can be run on a separate machine from the machine running bearerbox. The *smsbox-id* key provides a unique identifier for a specific smsbox instance. The *sendsms-port* key sets the port number used by the smsbox to send out HTTP requests.

```
group = smsbox
bearerbox-host = 192.168.10.11
smsbox-id = mysmsc
sendsms-port = 13013
global-sender = 13013
log-file = "/var/log/kannel/smsbox.log"
log-level = 2
access-log = "/var/log/kannel/access.log"
sendsms-chars = "0123456789-,.:() "
```

Figure 6.8: Smsbox Configurations Settings.

6.3.2.4 Wapbox

The wapbox configurations were set as shown in Figure 6.9. The *map-url* entry uses the format of "http://source/* http://destination/*". Hence, an incoming URL of

"http://source/some/path" is replaced with "http://destination/some/path" [30].

```
group = wapbox
bearerbox-host = 192.168.10.11
log-file = "/var/log/kannel/wapbox.log"
syslog-level = none
access-log = "/var/log/kannel/wapaccess.log"
map-url = "http://wapprideshare/* http://192.168.10.11:8090/*"
```

Figure 6.9: Wapbox Configuration Settings.

6.3.2.5 SMS Push

Kannel provides a mini-HTTP server [30] that intercepts HTTP requests formatted in the following manner:
“http://hostname:port/cgi-bin/sendsms?user=user&pwd=pwd&to=num&text=text”.
The mini-HTTP server configurations were put in a group called *sendsms-user,* and are shown in Figure 6.10. The mini-HTTP server recieves HTTP client requests from the service implementation bundle in TeleWeaver for SMS messages that must be sent to the users.

```
group = sendsms-user
username = k@nn3l
password = k@nn3l
user-deny-ip = "*.*.*.*"
#user-allow-ip = "146.231.122.16,146.231.123.86"
user-allow-ip = "127.0.0.1;146.231.122.16;146.231.123.86"
max-messages = 3
concatenation = true

#------------------------------------------
```

Figure 6.10: Sendsms Configuration Settings.

6.3.2.6 SMS Service

To implement our SMS application, we created a group for SMS services, called *sms-service* (mandatory variable), that is used by the smsbox. An SMS service defines the

expected behaviour when an SMS is received. Unique services are distinguishable by the first word in the message and by the number of arguments that follow it. Figure 6.11 shows an example of an SMS service configuration for the registration of users in our system.

```
group = sms-service
keyword = reg
max-messages = 3
post-xml = "http://127.0.0.1:9090/rideshare/users"
send-sender = true
```

Figure 6.11: SMS Service Configuration Settings.

The *keyword* specifies the string that matches a request for a service to be triggered. In this case, the triggered action is an XML post defined by the value of the *post-xml* key (*http://127.0.0.1:9090/rideshare/users)*. Therefore, when an SMS with a keyword *reg* is sent, Kannel sends a request to the specified address as an HTTP POST of the SMS message (in XML format). If an HTTP GET request is required, then *get-url* is used in place of the *post-xml*. The response from the HTTP request is sent to the SMS sender as an SMS message. The *max-messages* key specifies the maximum parts permitted for a multi-part message.

Following the configurations of SMS services in Kannel, a RESTful web service API (see Appendix C) was created and the next section explains how an OSGi bundle for RESTful web services was developed to complete the actions for the SMS application.

6.4 Implementing Restful Web Services

Restful web services can be implemented with frameworks such as Restlet and Apache CXF in OSGi environments as discussed in Section 5.3.3.2. TeleWeaver is provisioned with Apache CXF bundles, hence a RESTful web service was implemented using Apache CXF.

This section presents the tasks that were performed to implement a bundle that exposes the ridesharing service through RESTful webservices. The RESTful web service completes the development of an SMS application by ensuring that each keyword in an SMS message has a corresponding URI of an HTTP request for a particular operation of the DRS.

6.4.1 RESTful Services Bundle

To develop an OSGi bundle for the RESTful web service, the following tasks were performed:

1. Creating an OSGi bundle using the Maven archetype *maven-archetype-quickstar*t. The properties in the manifest file were set as follows:

   - Bundle-Name: *reedhousesystems-ridesharing-ws*
   - Bundle-Version: *1.2.2*
   - Bundle-Activator: *com.reedhousesystems.services.core.ridesharing.ws.RideshareBundleActivator*
   - Import-Package: *com.reedhousesystems.services.core.ridesharing, javax.ws.rs*, ... (a list of other imported packages)

2. Creating a bundle activator class named *RideshareBundleActivator.* This class initializes the bundle with RESTful web service settings that are specified in the *start* method. The *start* method sets the web service address as the host machine's localhost address on port 9090. The bundle is registered in the OSGi registry through the bundle context's *registerService* method. The settings can be seen in Figure B.4 of Appendix B.3.1.

3. Creating an implementation class named *RideshareRestService:* annotated for RESTful web services with JAX-RS annotations. According to Balani et al [10], in JAX-RS terminology, a class which has the JAX-RS annotations defined is termed as a resource class. Therefore, the resource class *(RideshareRestService)* has the URI as *http://localhost:9090/rideshare/*, following the settings in the *RideshareBundleActivator* class. In this way, each method in the resource class has a unique URI relative to the URI of the resource class. For example, the *addUser* method, annotated with *@Path("/users"),* has a URI as *http://localhost:9090/rideshare/users.*

   The *RideshareRestService* class uses the *@Consumes* annotation to indicate the content type that a method accepts. All the methods in the class have the content type set as text/xml since the expected content from Kannel is in XML format, as explained in Section 6.3.2.6.

   The *@Produces* annotation defines the content type that a method or resource class can produce and send back to the client. All the methods in the *RideshareRestService* class were set to produce plain text, which is delivered in an SMS message.

The ridesharing service is injected in the *RideshareRestService* class using the constructor *setRideService,* and is assigned the variable name *rideService*. The injected service (*rideService*) is referenced by the methods of the class when they perform actions to respond to requests sent by Kannel.

4. Creating a Java data object for Kannel's XML request data. Apache CXF uses Java Architecture for XML Binding (JAXB) as the default data binding component. JAXB is used to serialize request and response data objects in different formats such as Java Script Object Notation (JSON) and XML. In our case, the XML data from a Kannel request must be converted into a Java object understandable by the methods of the *RideshareRestService* class.

   A *Message* class with properties that represent the format of a Kannel XML post was created. The XML request data from Kannel is serialized into a *Message* object by JAXB. The *Message* object's *ud* (user data) property contains the SMS message contents from Kannel. To retrieve the parameters in an SMS message (separated by a single space character), the *StringTokenizer* class (Java utility class) breaks the message into tokens representing each parameter. For example, an SMS message sent as "profile user", results in two string tokens, "profile" and "user", which can be set as separate parameters for a method of the ridesharing service class.

5. Creating methods in the *RideshareRestService* class which reference the ridesharing service for the operations of the DRS and setting their URIs. For example, an HTTP POST request to the URI address *http://localhost:9090/rideshare/trips* creates a new trip. By creating methods and setting their corresponding URIs for the ridesharing operations, a RESTful web service API was completed. This is used by the Kannel requests for actions in the SMS application of the DRS. The RESTful web service API details are available in Appendix C of this document.

6.5 SMS Application Architecture

The architecture of the SMS application is presented in Figure 6.12. It shows how different components in Kannel and TeleWeaver interact in the implementation of a two-way SMS application, which follows a design discussed in Section 4.2.1. A mobile phone user sends an SMS message (with a keyword and parameters) to the phone number of the Kannel SMS gateway. Kannel uses its HTTP client to make an HTTP request to the RESTful web service running in bundle 1 (*reedhousesystems-ridesharing-ws*) inside TeleWeaver. Then, bundle 1 performs the necessary action for the request by referencing the ridesharing

service in bundle 2 (*reedhousesystems-ridesharing*). The response from bundle 1 is sent back to Kannel, which structures it as an SMS message response to the mobile phone user.

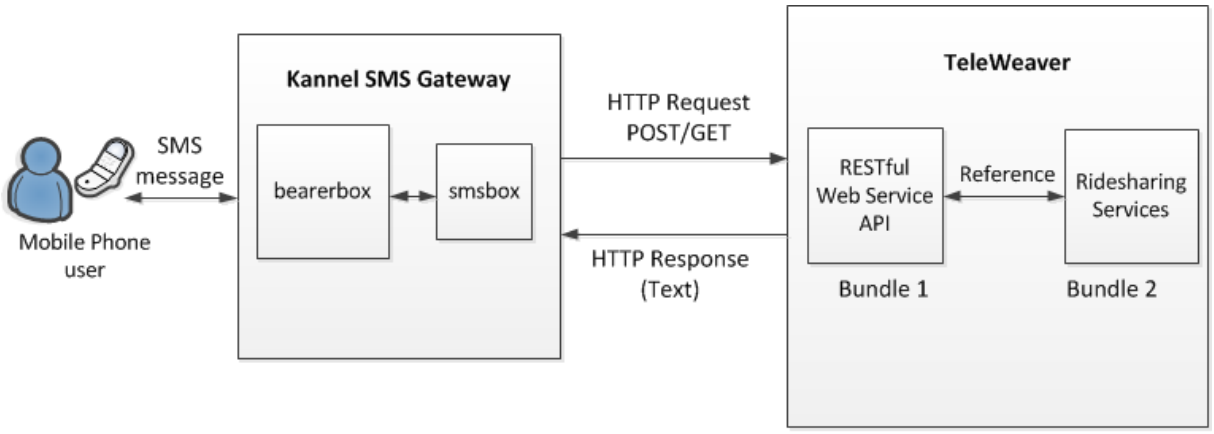

Figure 6.12: SMS Application with Kannel and TeleWeaver.

We have discussed how the ridesharing services are made available to mobile phone clients using the SMS technology provided by the Kannel SMS gateway. The next section discusses how mobile web clients access the ridesharing services using the WAP technology.

6.6 WAP Solution

Following the configurations of a WAP gateway as explained earlier in Section 6.3.2.4, the next step was to develop a web bundle that serves dynamic WML content. This section presents the tasks that were performed to develop an OSGi web bundle that targets mobile phone clients using the WAP technology.

6.6.1 Action-Based Web Framework

An action-based framework web bundle was implemented to render JSP pages with WML content. The Spring MVC framework was used in the implementation since its OSGi components are directly available in the Spring distribution, as mentioned in Section 5.3.2.1.

Following the understanding of how the Spring MVC works, as explained in Section 5.3.2.1, a dispatcher servlet and a controller were implemented to process web requests. In the implementation, the controller retrieves a request's parameters and references the

ridesharing service to create the *model* data. Then, the *model* is presented in a *view* that is appropriate for mobile phone browsers: a JSP page with WML tags.

6.6.2 OSGi Web Bundle

The steps that were performed to create an OSGi web bundle were as follows:

1. Creating a bundle using the *maven-archetype-quickstart,* and setting the manifest file properties as follows :

   - Bundle-Name: *reedhousesystems-ridesharing-mvc*
   - Bundle-Version: *1.2.2*
   - Import-Package: *com.reedhousesystems.services.core.ridesharing, javax.servlet.jsp, javax.servlet.jsp.jstl.core,...* (list of other import packages)

   Being a web bundle, a *web.xml* file was created in the *src/main/webapp/WEB-INF* folder. A *web.xml* file is used to specify properties such as the servlet and filter mappings, and the context paths that indicates an application's deployment URL [88]. In our *web.xml*, the dispatcher servlet class was set as

   org.springframework.web.servlet.DispatcherServlet. In addition, the web context was specified as *OsgiBundleXmlWebApplicationContext* to enable SpringDM support.

   Two Spring configuration files were added in the *src/main/resources/conf* folder: *application-context.xml* file for OSGi service reference and *servlet-context.xml* file for the *InternalResourceViewResolver* bean configuration. The *InternalResourceViewResolver* is recommended by Walls et al[106], for *views* that are rendered by JSP. The *InternalResourceViewResolver* was configured to use files with the extension *.jsp,* located in the */WEB-INF/jsp* folder.

2. Creating the controller class *RideshareController* with annotations for Spring MVC framework.

   The *RideshareController* class is annotated with the *@Controller* annotation to identify it as the controller. The handler methods in the *RideshareController* class are annotated with the *@RequestMapping* annotations and they include the URL address value. Part of the *RideshareController* class, which shows the annotations, is available in Figure B.6 of Appendix B.4.1. The web context path was configured as "ridesharing", therefore the URLs to the methods of the *RideshareController* class use the prefix as *http://localhost/ridesharing/.*

The ridesharing service provided by the *reedhousesystems-ridesharing* bundle is referenced in the *application-config.xml* file of the *reedhousesystems-ridesharing-mvc* bundle. The service is autowired in the *RideshareController* class and is used by the methods when processing web requests for the ridesharing operations.

3. Developing dynamic WAP pages.

   The dynamic WAP pages were developed using JSP considering that the system development was Java based. The JSP pages are placed in a folder named *jsp* inside the WEB-INF folder. For a web server to provide a service for WAP applications, the required MIME (Multipurpose Internet Mail Extension) type for the responses must be declared in the web page [46]. Therefore, the MIME type of the JSP pages was set to WML, using the JSP directive *<% response.setContentType("text/vnd.wap.wml"); %>*.

   The JSP pages use a combination of the WML and JSP Expression Language (EL). The WML tags structure the static page layout while the JSP EL inserts the dynamic content of the page. An example of how the JSP pages were implemented is shown in Figure B.7 in Appendix B.4.2.

4. Creating *ModelAndView* objects in the methods of the *RideshareController* class.

   The *ModelAndView* objects package the *model* data and the name of a *view*. A *ModelAndView* object is constructed with three parameters, logical name of a *view* and a name-value pair representing the *model* object. For example, instantiating a new *ModelAndView* object using the code "new ModelAndView("enquiry", "PersonDetails", Person)", the *ViewResolver* looks for a JSP page named *enquiry.jsp* (in the *jsp* folder under WEB-INF) and passes it to the *Person* object named *PersonDetails*. Then, the *ModelAndView* object (*PersonDetails*) properties can be accessed on the JSP page (*enquiry.jsp*) using the JSP EL.

6.7 WAP Implementation Architecture

The architecture of the WAP application for the DRS is shown in Figure 6.13. The Kannel WAP gateway, located between the mobile phone WAP browser and the content provider (TeleWeaver), translates WAP requests and responses (for the mobile phone) to normal HTTP protocol which is used by web servers (in our case, Jetty). It also compresses the WML pages, located in bundle 1, to improve the page processing by the phone as discussed in Section 5.5.1.

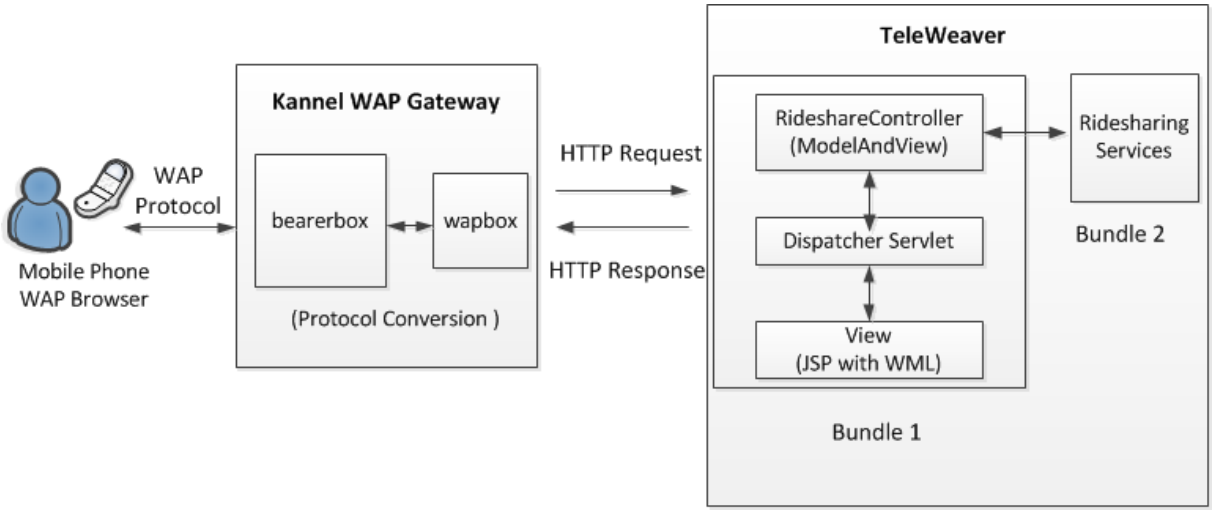

Figure 6.13: WAP Implementation with Kannel and TeleWeaver.

According to Figure 6.13, bundle 1 in TeleWeaver implements the Spring MVC framework and refers to bundle 2 when processing web requests for ridesharing operations. The response data from bundle 2 is packaged together with an appropriate *View* in a *ModelAndView* object to render the information on a JSP page as WAP content.

6.8 Summary

This chapter has presented the work that was undertaken to develop the DRS using OSGi bundles deployed in TeleWeaver. The DRS use the SMS and WAP Gateways provided by the Kannel system to make the services in TeleWeaver accessible by mobile phone clients. The next chapter covers the various system tests that were performed on the DRS to verify that its objectives were implemented as required.

Chapter 7

System Testing and Evaluation

The previous chapter discussed the implementation of the DRS using TeleWeaver and the Kannel system as an SMS and WAP gateway. This chapter focuses on the tests that were performed on the DRS and their results. It concludes with a discussion of the results which provide the assessment of the DRS against the requirements specifications that are presented in Chapter 4.

7.1 The Experimental Set-up

The experimental set-up for the tests on the DRS was arranged as shown in Figure 7.1.

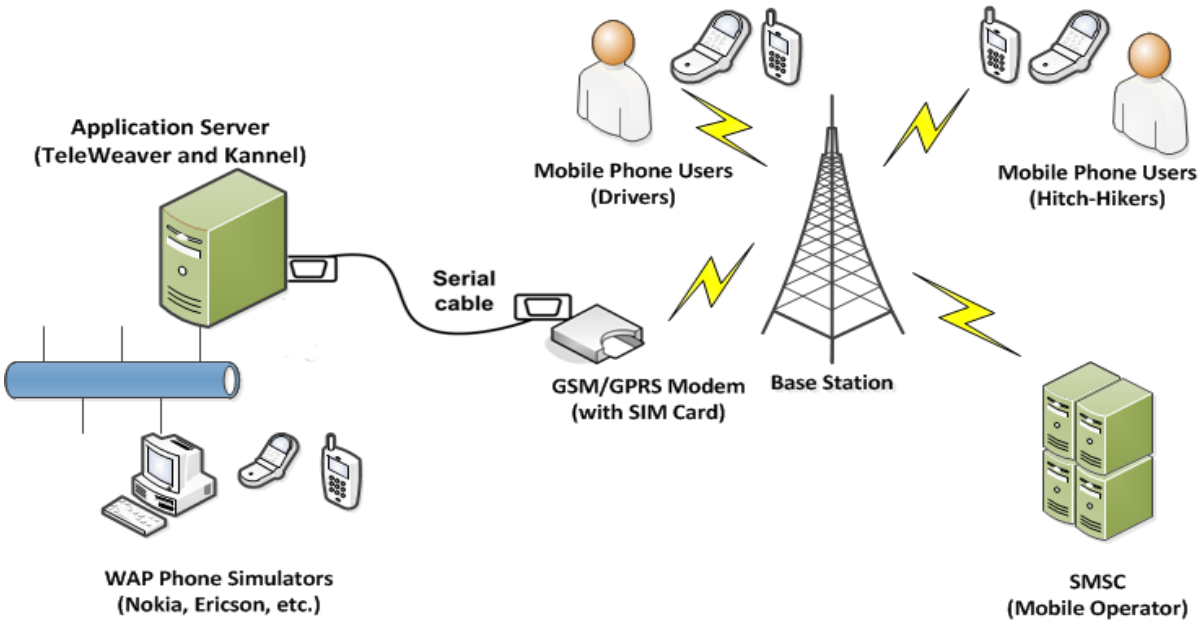

Figure 7.1: The Experimental Set-up.

Figure 7.1 shows how the server side, made up of TeleWeaver and Kannel, interacts with the mobile phone clients in the simulated and live environments. The SMS application of the DRS communicates with mobile phone users via the SMSC that is configured in the GSM/GPRS modem. During the tests, an MTN SIM card was used (mobile number +27785677238). Hence, all the tests for the SMS application involved actual mobile phones.

In conducting tests on the WAP application, WAP emulator softwares such as the Nokia S40 Emulator, WinWAP, Mozilla web browser WAP extension and WAPProof [107] were used. These applications provided emulators for different mobile phones which represent those owned by the target users of the DRS.

Nokia S40 is the world's most widely used mobile phone platform, with an extensive range of phone models in use [70]. The platform has been built into over 150 phone models. According to Nokia [70], it includes phones that offer budget conscious consumers options ranging from their first internet capable mobile phone to a phone with smartphone-like features such as accelerometers and touch screens. Therefore, the series 40 simulator provided a test platform for phone types that are owned by a significant population of feature phone users in South Africa (as highlighted in Section 2.4.1.1).

WAPProof provides 21 different mobile devices that are emulated [107]. These include Nokia, Samsung, Siemens and Motorola brands. The availability of different types of mobile phones using WAPProof enabled a wide test on the behaviour of user interactions on the different screen designs and user input methods that are provided.

The WinWAP and Mozillaweb browser WAP extension were used for tests on the WAP page designs on the desktop computers. The tests provided initial results on the interface design of the WAP pages and their behaviour (e.g. navigations) before carrying out tests in the mobile phone environment (small screens, input mechanism, etc.).

7.2 The System Test Methodology

The following aspects of the system were tested in the process:

1. Configurations of the server applications

   By carrying out tests on the deployment environment (TeleWeaver) and checking the configurations in Kannel for the SMS and WAP gateway services.

2. Functional tests

Performing tests on the DRS to verify that the functional requirements, set in Section 4.1.3.1, and the matching algorithm, discussed in Section 4.1.3.2, are implemented as expected.

3. Usability tests

   Conducting usability tests and using a sample of users to get feedback from their assessment of the DRS.

7.3 Testing the Server Applications

The components in the TeleWeaver and Kannel systems were tested to verify that the configurations were correct for the operations of the DRS. This section provides the results of the tests on the two systems.

7.3.1 TeleWeaver

TeleWeaver hosts the OSGi bundles for the ridesharing service of the DRS. The OSGi bundles must be in *active* state (all dependencies resolved) when providing the ridesharing service, based on the OSGi bundle life cycle that is discussed in Section 5.1.2. The console command "*ss*" can be issued in TeleWeaver to check the status of each bundle in the container. Figure 7.2 shows the "*ACTIVE*" status of the bundles for the DRS implying that they are registered in TeleWeaver's OSGi registry. Therefore, the DRS bundles can perform OSGi services (register, discover and consume) in the TeleWeaver container.

```
174     ACTIVE      com.reedhousesystems.services.core.reedhousesystems-ridesharing-api_1.2.2
175     ACTIVE      com.reedhousesystems.services.core.reedhousesystems-ridesharing_1.2.2
176     ACTIVE      com.reedhousesystems.services.core.reedhousesystems-ridesharing-mvc_1.2.2
177     ACTIVE      com.reedhousesystems.services.core.reedhousesystems-ridesharing-ws_1.2.2

osgi>
```

Figure 7.2: DRS Bundles on TeleWeaver's OSGi Console.

Each bundle can be managed independently such that changes (e.g. adding an upgrade version of a bundle) can be made without restarting the container. In addition, more than one version of a bundle can be deployed in the container. Such server management abilities provide the stability of the DRS, meaning that clients will not experience service disruptions when new DRS bundles are tested in TeleWeaver.

7.3.2 Kannel System

The Kannel System provides the DRS with the SMS and WAP gateway functionality. The configurations of the gateways were tested to ensure the successful operation of the SMS application and the WAP web pages.

7.3.2.1 Testing the BearerBox

All configurations in Kannel must always include a group for general bearerbox configuration [30]. The *core* group (explained in Section 6.3.2.1) is used to instantiate the bearerbox. Therefore, it must be configured properly for the SMS and WAP gateways to operate.

The status of the bearerbox can be checked by issuing the command *"bearerbox -v l kannel.conf"* on the *kannel.config* configuration file. The results from our test on the bearerbox are shown in the console display of Figure 7.3. The last line on the console output shows that the bearerbox is configured properly to communicate with the SMSC. This confirms that the bearerbox is ready to work with the smsbox and wapbox.

```
2012-11-29 12:24:19 [7005] [6] DEBUG: AT2[AT]: --> AT+CSMS=?^M
2012-11-29 12:24:20 [7005] [6] DEBUG: AT2[AT]: <-- +CSMS: (0,1)
2012-11-29 12:24:20 [7005] [6] DEBUG: AT2[AT]: <-- OK
2012-11-29 12:24:20 [7005] [6] INFO: AT2[AT]: Phase 2+ is supported
2012-11-29 12:24:20 [7005] [6] DEBUG: AT2[AT]: --> AT+CSMS=1^M
2012-11-29 12:24:20 [7005] [6] DEBUG: AT2[AT]: <-- +CSMS: 1,1,1
2012-11-29 12:24:20 [7005] [6] DEBUG: AT2[AT]: <-- OK
2012-11-29 12:24:20 [7005] [6] DEBUG: AT2[AT]: --> AT+CNMI=1,2,0,1,0^M
2012-11-29 12:24:20 [7005] [6] DEBUG: AT2[AT]: <-- OK
2012-11-29 12:24:20 [7005] [6] INFO: AT2[AT]: AT SMSC successfully opened.
```

Figure 7.3: Bearerbox at Work.

7.3.2.2 Testing the Smsbox

When the bearerbox is in operation, the smsbox can be started using the command *"smsbox kannel.conf"*. The smsbox confirms the established connection with the bearerbox in its message display as shown in Figure 7.4.

```
2012-11-29 12:40:00 [7174] [0] DEBUG:     <username> = <k@nn3l>
2012-11-29 12:40:00 [7174] [0] DEBUG:     <concatenation> = <true>
2012-11-29 12:40:00 [7174] [0] DEBUG:     <password> = <k@nn3l>
2012-11-29 12:40:00 [7174] [0] DEBUG: Started thread 4 (gw/smsbox.c:obey_request_thread)
2012-11-29 12:40:00 [7174] [4] DEBUG: Thread 4 (gw/smsbox.c:obey_request_thread) maps to pid 7174.
2012-11-29 12:40:00 [7174] [0] DEBUG: Started thread 5 (gw/smsbox.c:url_result_thread)
2012-11-29 12:40:00 [7174] [5] DEBUG: Thread 5 (gw/smsbox.c:url_result_thread) maps to pid 7174.
2012-11-29 12:40:00 [7174] [0] DEBUG: Started thread 6 (gw/smsbox.c:http_queue_thread)
2012-11-29 12:40:00 [7174] [6] DEBUG: Thread 6 (gw/smsbox.c:http_queue_thread) maps to pid 7174.
2012-11-29 12:40:00 [7174] [0] INFO: Connected to bearerbox at 192.168.10.11 port 13002.
2012-11-29 12:40:00 [7174] [0] DEBUG: Started thread 7 (gw/heartbeat.c:heartbeat_thread)
2012-11-29 12:40:00 [7174] [7] DEBUG: Thread 7 (gw/heartbeat.c:heartbeat_thread) maps to pid 7174.
```

Figure 7.4: Smsbox at Work.

7.3.2.3 Testing the Wapbox

The wapbox is launched using the command "*wapbox kannel.conf*". Figure 7.5 shows the console output that confirms the launch of the wapbox and the connection established to the bearerbox. The successiful launch of the wapbox completes the startup of the important components of the Kannel system (bearerbox, smsbox and wapbox). This confirms that the Kannel system is ready to work with TeleWeaver for the SMS and WAP requirements of the DRS.

```
2012-11-29 13:00:10 [7312] [0] INFO: Kannel wapbox version 1.4.3 starting up.
2012-11-29 13:00:10 [7312] [0] DEBUG: Started thread 1 (wap/wsp_session.c:main_thread)
2012-11-29 13:00:10 [7312] [1] DEBUG: Thread 1 (wap/wsp_session.c:main_thread) maps to pid 7312.
2012-11-29 13:00:10 [7312] [0] DEBUG: Started thread 2 (wap/wsp_unit.c:main_thread)
2012-11-29 13:00:10 [7312] [2] DEBUG: Thread 2 (wap/wsp_unit.c:main_thread) maps to pid 7312.
2012-11-29 13:00:10 [7312] [0] DEBUG: Started thread 3 (wap/wsp_push_client.c:main_thread)
2012-11-29 13:00:10 [7312] [3] DEBUG: Thread 3 (wap/wsp_push_client.c:main_thread) maps to pid 7312.
2012-11-29 13:00:10 [7312] [0] DEBUG: Started thread 4 (wap/timers.c:watch_timers)
2012-11-29 13:00:10 [7312] [4] DEBUG: Thread 4 (wap/timers.c:watch_timers) maps to pid 7312.
2012-11-29 13:00:10 [7312] [0] DEBUG: Started thread 5 (wap/wtp_resp.c:main_thread)
2012-11-29 13:00:10 [7312] [5] DEBUG: Thread 5 (wap/wtp_resp.c:main_thread) maps to pid 7312.
2012-11-29 13:00:10 [7312] [0] DEBUG: Started thread 6 (gw/wap-appl.c:main_thread)
2012-11-29 13:00:10 [7312] [6] DEBUG: Thread 6 (gw/wap-appl.c:main_thread) maps to pid 7312.
2012-11-29 13:00:10 [7312] [0] DEBUG: Started thread 7 (gw/wap-appl.c:return_replies_thread)
2012-11-29 13:00:10 [7312] [7] DEBUG: Thread 7 (gw/wap-appl.c:return_replies_thread) maps to pid 7312.
2012-11-29 13:00:10 [7312] [0] INFO: Connected to bearerbox at 192.168.10.11 port 13005.
2012-11-29 13:00:10 [7312] [0] DEBUG: Started thread 8 (gw/heartbeat.c:heartbeat_thread)
2012-11-29 13:00:10 [7312] [8] DEBUG: Thread 8 (gw/heartbeat.c:heartbeat_thread) maps to pid 7312.
```

Figure 7.5: Wapbox at Work.

We have presented the results of the various tests which show that the DRS components in TeleWeaver and Kannel were correctly configured to provide ridesharing operations for hitch-hiking to mobile phone clients. The next section presents the functional tests that were performed to verify the operations of the DRS in organizing hitch-hiking trips.

7.4 Functional Test Results

This section presents the tests that were conducted to verify the functional specifications stated in Section 4.1.3.1. The tests on the WAP application were done on desktop computers using web browsers with WML support and mobile phone emulators. The SMS application was tested using actual mobile phones. However, some of the screenshots for the SMS application use a mobile phone emulator to provide clear information. This section explains how the tests were conducted and the results that were obtained.

7.4.1 Testing the RESTful Web Service API

To carry out tests on the DRS using the two-way SMS application, the RESTful web service API (Appendix C) which provides the communication link between Kannel and TeleWeaver was tested. Tests were conducted to verify that all the URI requests are processed by the appropriate JAX-RS annotated methods in the *RideshareRestService* class of the *reedhousesystems-ridesharing-ws* bundle, following the implementation discussed in Section 6.4.1. In addition, the data transfers were tested to ensure that correct formats are used in the operation of the RESTful web service: Kannel requests consumed in XML format, and the TeleWeaver responses produced in plain text format.

An external REST client application was used to test the RESTful web service of the DRS in TeleWeaver. There are free REST client applications which run in web browsers and are installed as plugins. Examples include RESTClient [63] for Mozilla Firefox browser and Simple REST Client [97] for Chrome browser. The tests on the RESTful web service involved the use of a REST client application to send XML data requests (using HTTP POST and GET methods) in a similar format to Kannel requests. Figure 7.6 shows an example of a test request that was performed on the RESTful web service (in TeleWeaver) using the Simple REST client application.

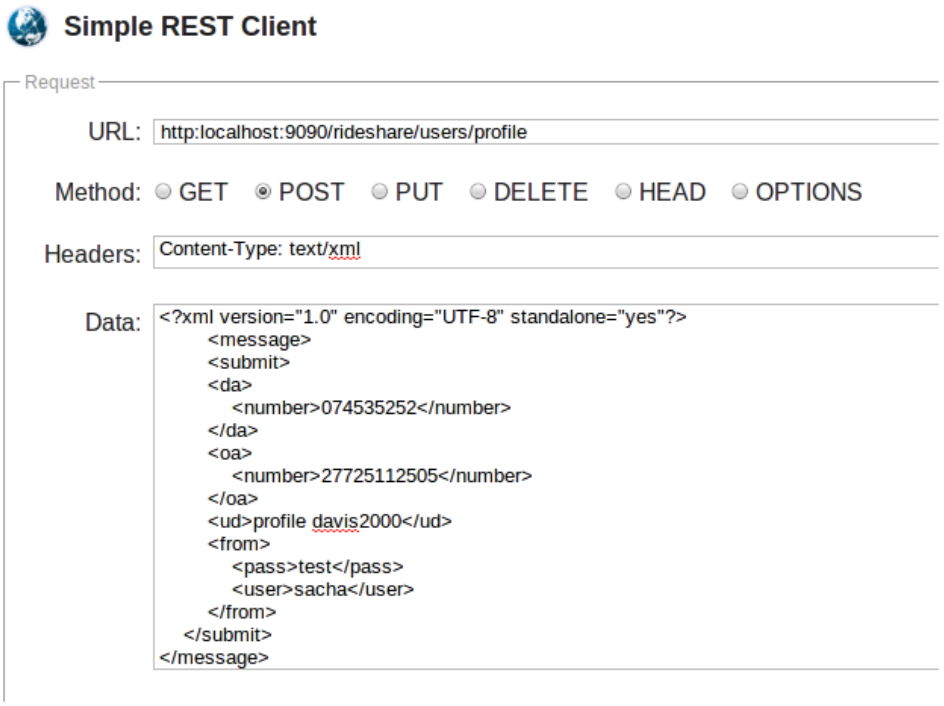

Figure 7.6: Testing the RESTful Web Service using the Simple Rest Client.

In the example above, the *URL* field takes the URI address that Kannel would use to make an HTTP request using the *Method* of HTTP POST. The *Headers* field sets the content type of the response to *text/xml*; which is also the setting on the produced content for the methods of the *RideshareRestService* class. The *Data* field gets the XML data in a structure that Kannel uses when sending HTTP requests. In our case, the important data in a Kannel request are the value of the *oa* element (the sender's phone number) and the *ud* element which has the SMS message contents.

The response from TeleWeaver comes in plain text and Figure 7.7 shows the reply (*Data* field) that Kannel is supposed to send as an SMS message following the request shown in Figure 7.6. The response contains personal details for a particular username (davis2000) fetched from the database. The status value of "200 OK" confirms that the service was executed without any error.

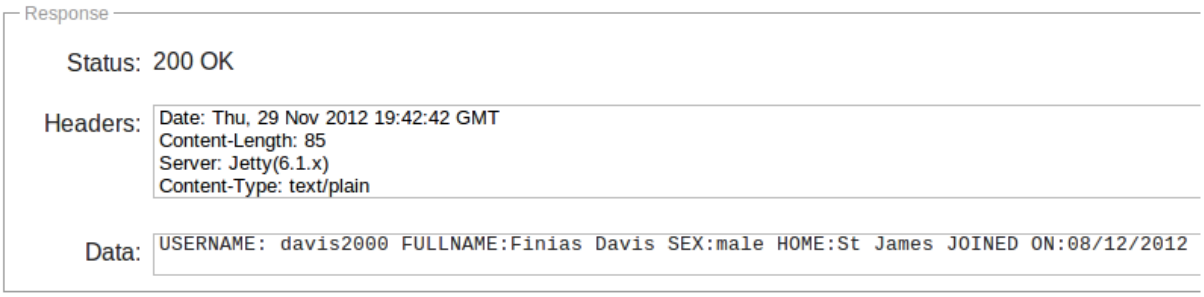

Figure 7.7: Response from a RESTful Web Service.

The test results above demostrate how the RESTful web service in TeleWeaver responds to HTTP requests sent by Kannel. The tests of this kind were performed on all the URIs defined for the methods of the *RideshareRestService* class in the RESTful web service bundle. The RESTful web service API for the SMS application is available in Appendix C of this document. The next section will present the tests that were performed on the core functions of the DRS to check if the functional requirement specifications were met.

7.4.2 User Registration

User registration is a requirement before using the DRS as outlined in the requirements specifications provided in Section 4.1.3. The SMS and WAP applications provide the registration function as shown in Figures 7.8. The SMS application uses the keyword "Reg" (not case sensitive) and subsequent parameters of the new user's details. The user details are defined in the following order: firstname, surname, username, password, gender, language (default is english) and home location. The WAP application provides a registration form that has the input fields of the new user details.

In the SMS registration, the phone number of a new user is extracted from the value of the *oa* element in the Kannel XML message as explained earlier in Section 7.4.1. After a successful registration, the DRS responds with a confirmation SMS message which includes keywords for further instructions on how to use the system (for driver and hitch-hiker roles).

User registration through the WAP application requires a new user to submit a username and phone number. After registering, a password is auto generated and sent in an SMS message to the phone number provided by the user. This verifies that the registered phone number belongs to the new user. The user can change the assigned password after logging into the system. Figure 7.9 shows the records of users in a MySQL database table called *PERSON*, after successfully registering in the DRS.

Figure 7.8: User Registration

```
mysql> select * from PERSON;
+-----------+------------+------------+----------+--------------+-----------+------------+--------+-----------+----------+
| Person_Id | Surname    | Firstname  | Username | Passcode     | Phone_Num | Home_Loc   | Gender | Day_Of_Reg | Language |
+-----------+------------+------------+----------+--------------+-----------+------------+--------+-----------+----------+
|        43 | Honye      | Silvester  | sly      | 45gs9m       | 0782153181 | Grahamstown | Male   | 2012-08-30 | English  |
|        44 | Miteche    | Sacha      | smiteche | 12345        | 0725112505 | Grahamstown | Male   | 2012-08-31 | Xhosa    |
```

Figure 7.9: Registered User Records in MySQL Database.

7.4.3 Organizing Rideshare Trips

When organizing a rideshare trip, a driver provides information which includes vehicle registration, destination and the preferred pick-up point. Figure 7.10 shows how a trip can be created using an SMS message and a WAP web page. For the SMS message, the keyword “Trip” is used and is followed by the required trip details. In the example shown in Figure 7.10, “test123” is the registration number of the vehicle to be used in the trip. The destination is set as “pe” which is the short-code for Port Elizabeth. A user can enter the full name of a location (as a single word) or use a short-code. This is fine for full location names that are one word, such as Grahamstown. Therefore, a short-code must

be used for location names that have more than one word (e.g. King Williams Town, East London, Port Elizabeth, etc.) to ensure that the correct location name is used. Similarly, the location name of the pickup point, in the SMS message example, can use "gt" or "Grahamstown" and should use short-codes for location names with more than one word. The street name of the pickup point has been set as "beaufort" and the nearest landmark as "shoprite".

Figure 7.10: Creating a Trip.

Once a trip is created, a confirmation message, which includes a secret code *TripCode,* is sent to the driver. Figure 7.11 shows a confirmation message after creating a trip. The same message is sent to a user as a response SMS message when using the SMS application.

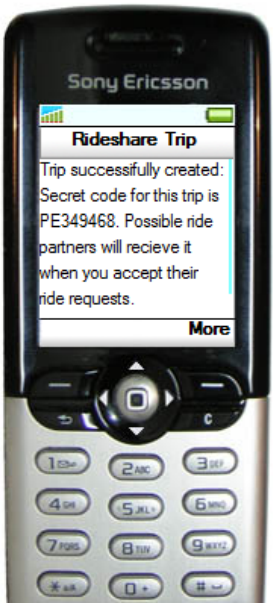

Figure 7.11: A Trip Confirmation Message to the Driver.

7.4.4 Ride Offer to Hitch-Hikers

The SMS application uses the keyword "Offer" followed by the ride offer conditions. The conditions are in the following order: seats, cost, offer duration and additional condition. The additional condition is optional and if included, a maximum of five words are accepted. An example of a ride offer to hitch-hikers is shown in Figure 7.12. In the SMS message example, the conditions of a recently created trip are set as follows: a maximum of four hitch-hikers only, a rideshare cost of R100 and the ride offer is valid for 30 minutes.

Figure 7.12: Ride Offer by Driver.

The results for the ride offer are immediately sent to the driver as shown in Figure 7.13. If no match is found, a message for the unsuccessiful matches is sent, as shown in part A. Otherwise, a list of hitch-hikers (limited by the offered seats) is provided as shown in part B. The same messages are also provided in an SMS message when using the SMS application. If the list of hitch-hiker details does not fit in a single SMS message (160 characters) then a multi-part SMS (limited to three) is sent to the driver.

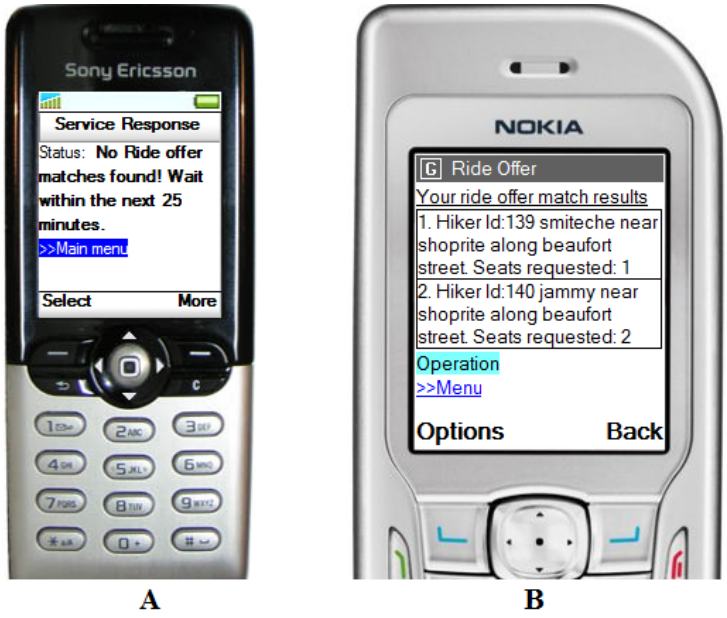

Figure 7.13: Ride Offer Request Results.

7.4.5 Ride Request by a Hitch-Hiker

A ride request by a hitch-hiker is performed in a similar way as explained above for a ride offer by a driver. The SMS message uses the keyword “Hike” followed by the hitch-hiking details: destination name, current location name, street and nearest landmark. An optional parameter is the number of seats requested, in the case of a hitch-hiker travelling with a partner. The screen shots for a ride request operation are shown in Figure 7.14.

Figure 7.14: Ride Request by Hitch-Hiker.

After sending a ride request, a message with the result is sent to the user indicating whether or not a match has been found, as shown in Figure 7.15. Both messages shown in part A (match not found) and part B (match found) have a hitch-hiker identification number which the DRS uses to refer to the user's hitch-hiker role. This number is used when modifying or cancelling the associated hitch-hiking record.

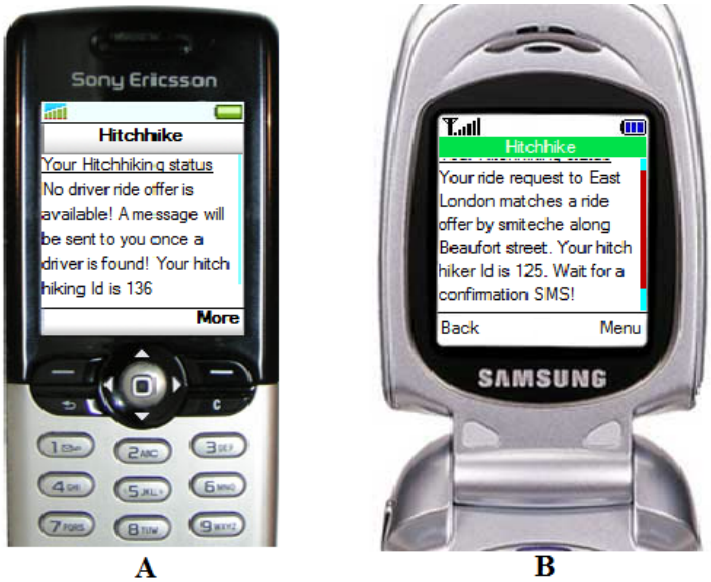

Figure 7.15: Ride Request Results.

When a matching ride request by a driver is found, as indicated in the message shown in part B of Figure 7.15, an SMS message alert for an available hitch-hiker is sent to the driver. The driver can add a single hitch-hiker as explained in the next section.

7.4.6 Adding a Hitch-Hiker to a Trip

A driver confirms the intention to pick-up a single hitch-hiker or a list of hitch-hikers by using the add hitch-hikers operation. A list of matched hitch-hikers is available when a ride offer request is made, as shown earlier in part B of Figure 7.13. The driver can decide to accept all hitch-hikers using a single action. In the SMS application, all matched hitch-hikers can be accepted in a trip by sending an SMS message "Accept all", which uses the keyword "Accept". The WAP application provides a button to accept all.

If the driver prefers to add a single hitch-hiker from a list of hitch-hikers then a hitch-hiker identity number must be used as shown in Figure 7.16. After adding a hitch-hiker (or hitch-hikers) to a trip, an SMS message is sent to a hitch-hiker to confirm the driver's acceptance as a trip partner. The message includes the trip secret code (*TripCode*), vehicle details and the estimated pickup time limit, as shown in Figure 7.17.

Figure 7.16: Accept Hitch-Hiker for a Trip

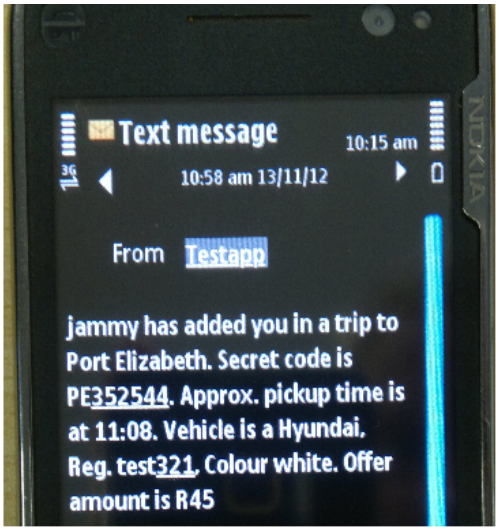

Figure 7.17: Trip Confirmation to a Hitch-Hiker

7.4.7 Informing a Next Of Kin

The DRS provides an optional operation that notifies a next of kin of a user who is participating in a hitch-hiking trip. This operation can be performed using an SMS message with the keyword "Inform", and followed by the secret code of a trip (*TripCode*). Figure 7.18 shows the screenshots for this operation. The DRS generates an SMS message with the trip details that the user would like to share to the next of kin, as shown in Figure 7.19. The next of kin, who is registered under the user's profile, can receive further information (e.g. other passengers) by sending an SMS message with the keyword "RideInfo", followed by the secret code provided (*PE352544*). The full trip information that can be viewed is shown in Figure E2 of Appendix E.

Figure 7.18: Inform Next Of Kin.

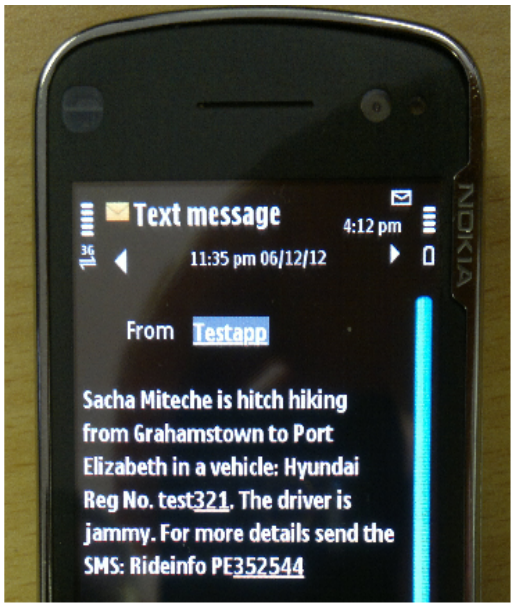

Figure 7.19: An SMS Sent To A Next Of Kin.

7.4.8 Multilingual Support

To localize the DRS, the isiXhosa language was added to the system considering that the majority of people in the Eastern Cape speak isiXhosa. The language of the messages when performing the DRS operations can be set to isiXhosa by sending an SMS with the keyword “Lang”, followed by “x” or “xhosa” as shown in Figure 7.20. The message in

isiXhosa, shown in Figure 7.20, is sent to a user after creating a trip and is a translation of a similar message shown in Figure 7.11. The equivalent isiXhosa translations for the SMS keywords were found difficult to express in simple words. As such, they remained unchanged as recommended by the users who took part in the usability tests of the DRS (covered in Section 7.5). To return to English language, an SMS of either "Lang e" or "Lang english" must be used.

Figure 7.20: Changing Language to IsiXhosa.

The full instructions on how to operate the DRS using the SMS application can be found in Appendix D of this document. Selected instructions are available when an SMS with a keyword "Instruct" is sent to the DRS. The next section will discuss the usability tests that were conducted to gather the assessment views on the DRS, from a sample of drivers and commuters who participate in hitch-hiking travels.

7.5 Usability Tests

The user feedback on the functions of the DRS was gathered through a usability test process. The usability tests measured the effectiveness, efficiency and satisfaction of the operations of the DRS, as explained in the non functional requirements provided in Section 4.1.3.3.

The usability tests for the DRS were conducted using the SMS application. The SMS application was selected to take advantage of the availability of SMS in mobile phone handsets owned by the target users as well as the users' familiarity with SMS messaging. SMS services are widely available in urban and rural areas in South Africa through the GSM network. According to MTN, a mobile phone service provider, SMS services reach

93% of the population [66]. This provided the freedom to carry out the usability tests at almost any location point in the urban and rural places of the study area.

The usability tests targeted only people (in urban and rural areas) who hitch-hike or pick-up hitch-hikers. The tasks that were performed were as follows:

1. Each participant responded to a user background questionnaire (see Appendix A.2.1).
2. A demonstration of how the system works to provide an overview understanding.
3. Each participant performed the DRS operations to organize a rideshare trip using a personal mobile phone. The tests involved playing both the role of a driver and a hitch-hiker in separate runs. When conducting the test with more than one participant, the driver and hitch-hiker roles were assigned to the participants and exchanged in the second run of the test.
4. Each participant responded to a Systems Usability Scale (SUS) questionnaire after performing the tests. The SUS is a technology independent questionnaire for measuring perceptions of usability and is used for many types of systems which include consumer software, websites, mobile phones, etc [92]. We adapted some of the questions of the SUS, which we found relevant for our system, to establish our questionnaire for the usability tests. The SUS questionnaire for the usability tests on the DRS is available in Appendix A.2.2.

7.5.1 Urban Context

Usability tests were conducted in the Grahamstown area and a total of nine people participated in the tests. The background of the participants is shown in the results of the user background questionnaire, in Figure 7.21. The results indicate that two of the participants had real-life experience in both roles of a driver and a hitch-hiker. All the participants stated that they owned a mobile phone and had background knowledge in using SMS and computers.

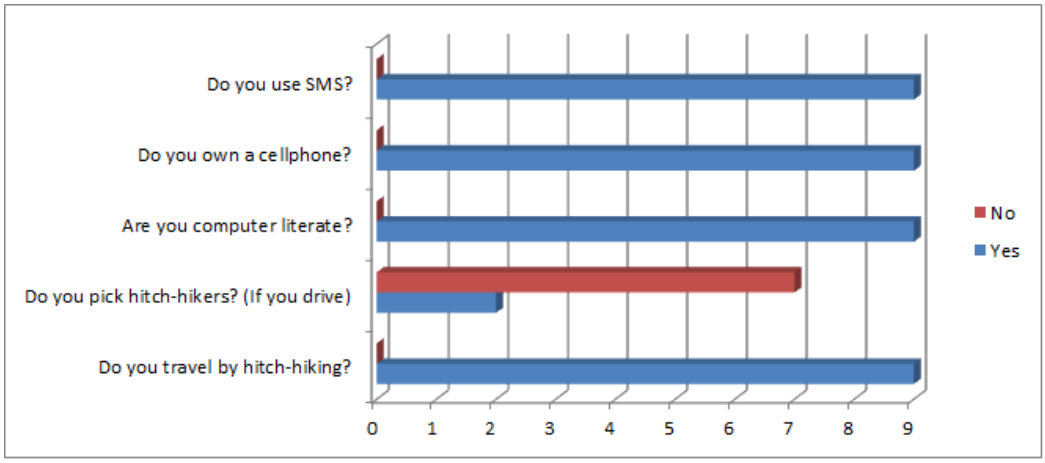

Figure 7.21: SUS: User Background Information.

After testing the system, each participant responded to the SUS questionnaire and the results were as follows:

1. Relevance of the DRS to hitch-hiking travel arrangements

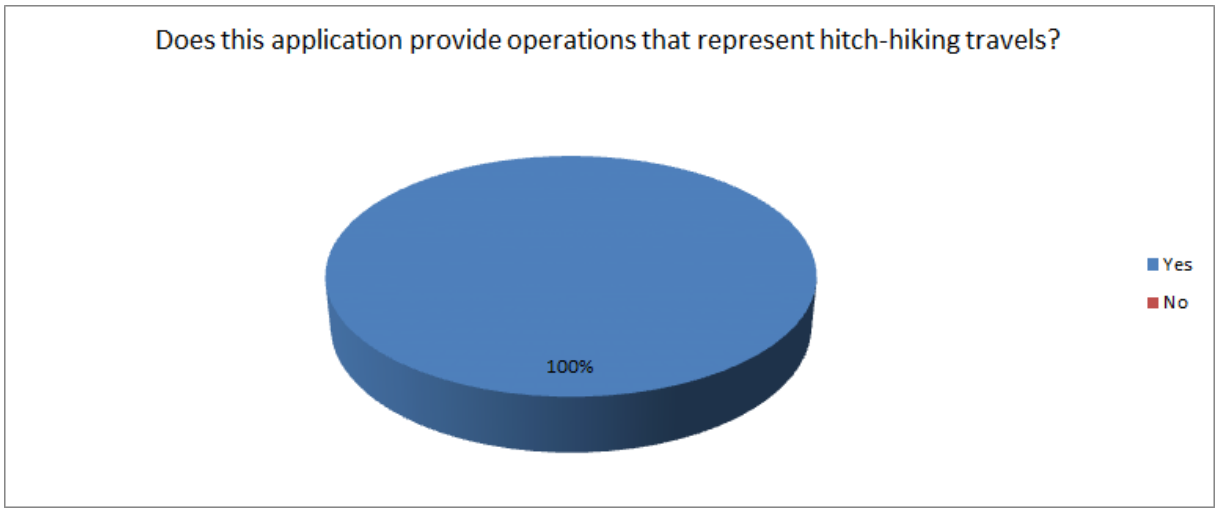

Figure 7.22: SUS: Relevance to Hitch-Hiking.

Figure 7.22 shows that all nine participants agreed that the DRS provides operations that relate to hitch-hiking travels. By performing the tests in both roles, as a driver and a hitch-hiker, each participant had the experience of how to organize a rideshare trip and also how to search for a ride offer using the DRS. Therefore, the response by the participants provides a strong indication that the operations of the DRS represent the actions that are performed in reality.

2. Ease of use in performing the DRS operations

   During the usability tests, the user instructions for the SMS application was provided to the participants. This was done to remove the extra challenge of memorizing the structure of SMS message requests. The main interest in the tests was to measure how easy it is to perform the operations if the user understands what information is required; and how to send requests using correct formats.

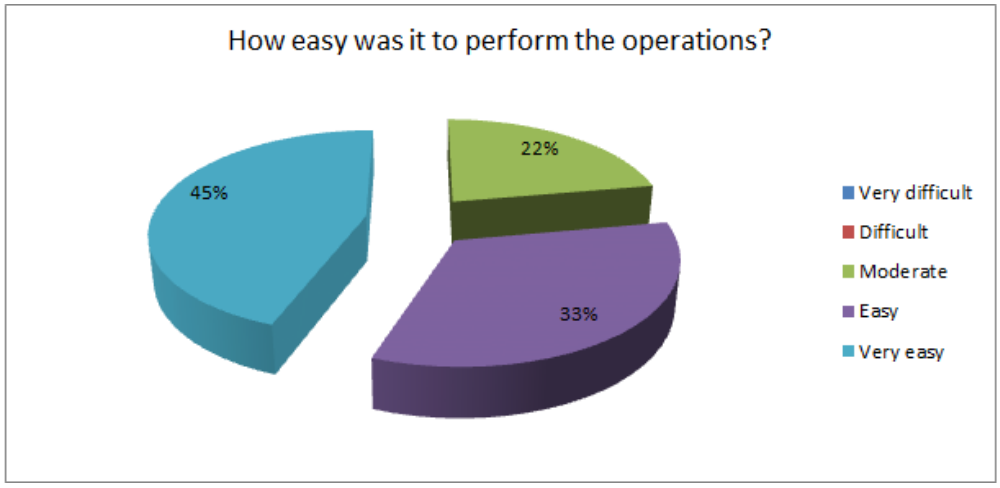

Figure 7.23: SUS: Ease of Use.

   Figure 7.23 shows that none of the respondents indicated having difficulty in performing the operations. All responses show that the operations are achievable without much effort.

3. The anticipated learning curve for other people

   Based on the experience in performing the operations, the participants provided their opinion on how quickly they thought other people could learn how to use the system.

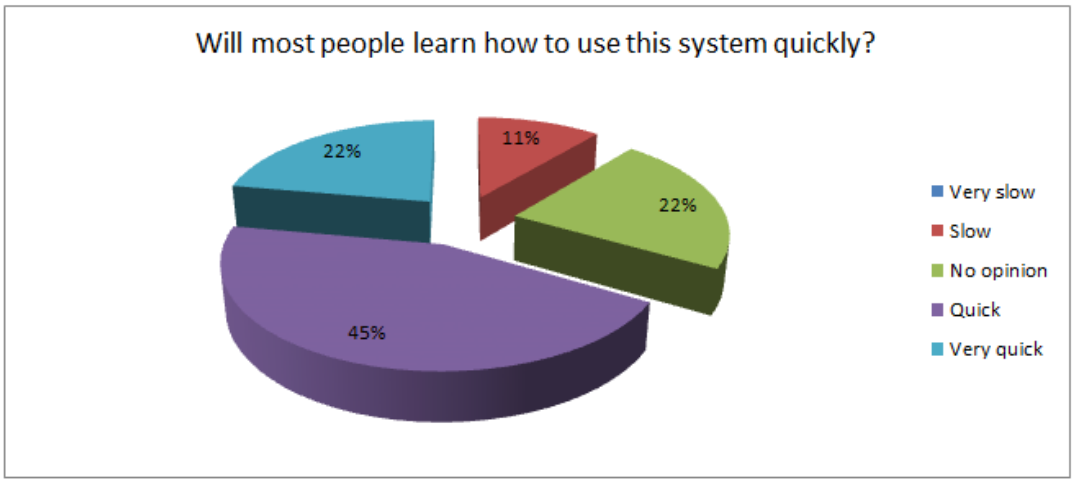

Figure 7.24: SUS: How Easy to Learn.

The responses were divided, as shown in Figure 7.24. The majority suggested that it would be quick to learn the system, while others indicated that they did not have an opinion.

4. Overall rating of the DRS

   The overall rating shown in Figure 7.25 indicates that the participants were at least satisfied with what they experienced in testing the system.

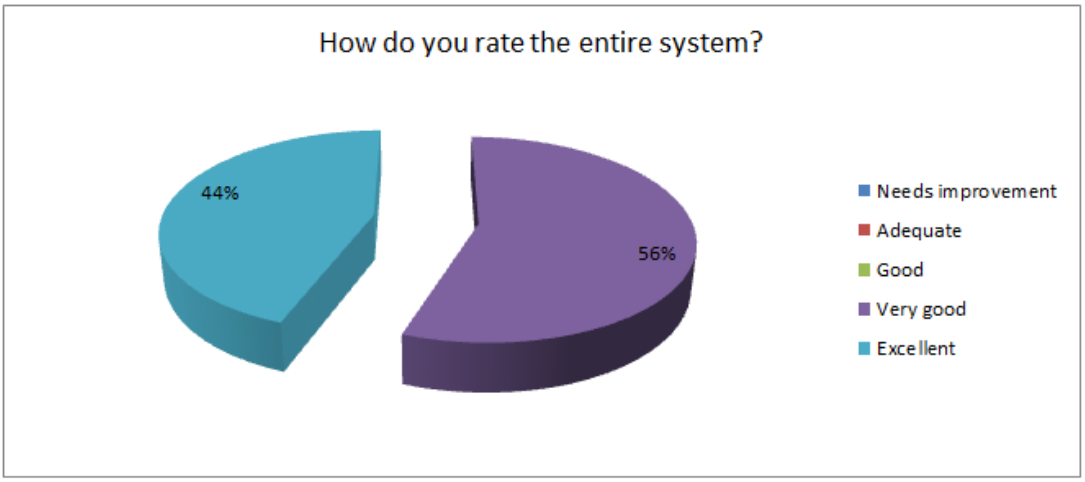

Figure 7.25: SUS: Rating for the System.

7.5.2 Rural Context

For the rural context, the usability tests were conducted in the rural areas of Dwesa. A total of 10 people participated in the tests in Nqabara (four people) and Ngwane (six

people). The usability test included a test on the isiXhosa language setting of the DRS. The participants were allowed to select a language of choice (english or isiXhosa) during the tests. The results of the user background questionnaire are provided in Figure 7.26.

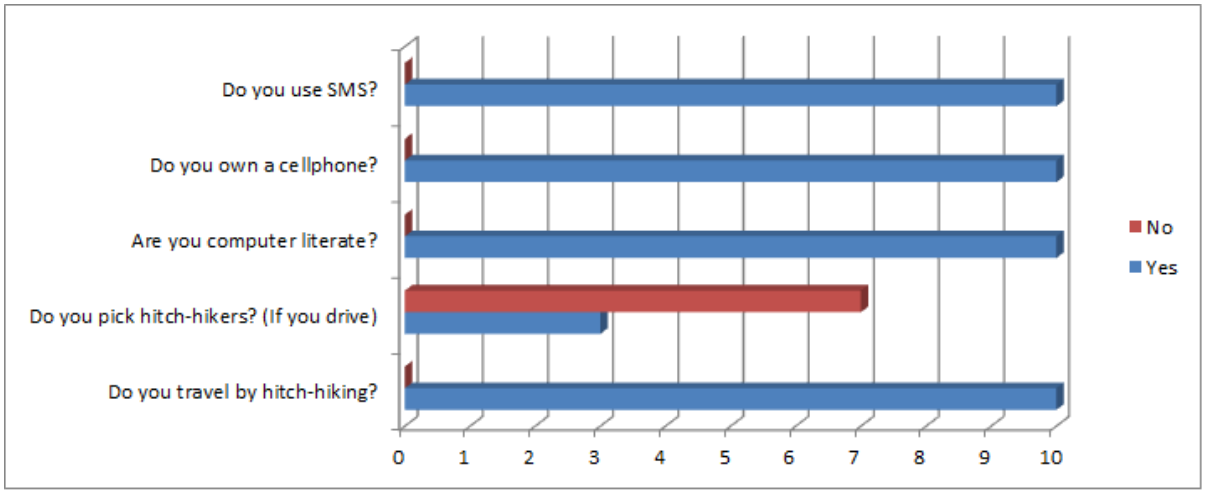

Figure 7.26: SUS: User Background Information.

The responses for the SUS questionnaire after performing the tests were as follows:

1. The relevance of the DRS to hitch-hiking

   All the 10 participants agreed that the system provided operations that are relevant to hitch-hiking, as shown in Figure 7.27.

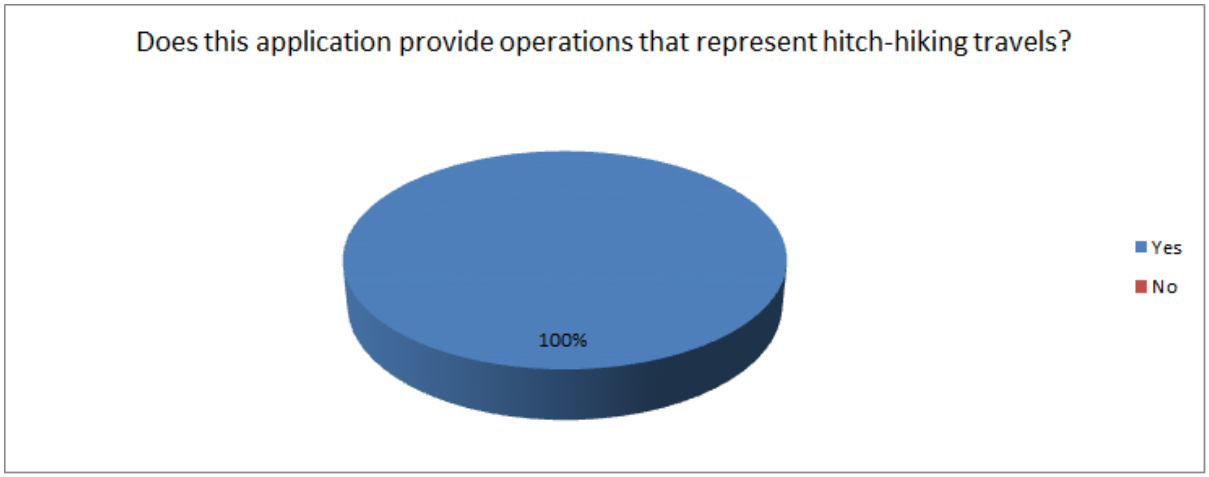

Figure 7.27: SUS: Relevance to Hitch-Hiking.

2. Ease of use in performing the DRS operations

Most participants (80%) responded that it was easy to perform the ridesharing operations, as shown by the results in Figure 7.28. This indicates the advantage of using SMS service which is familiar to most mobile phone users in this context.

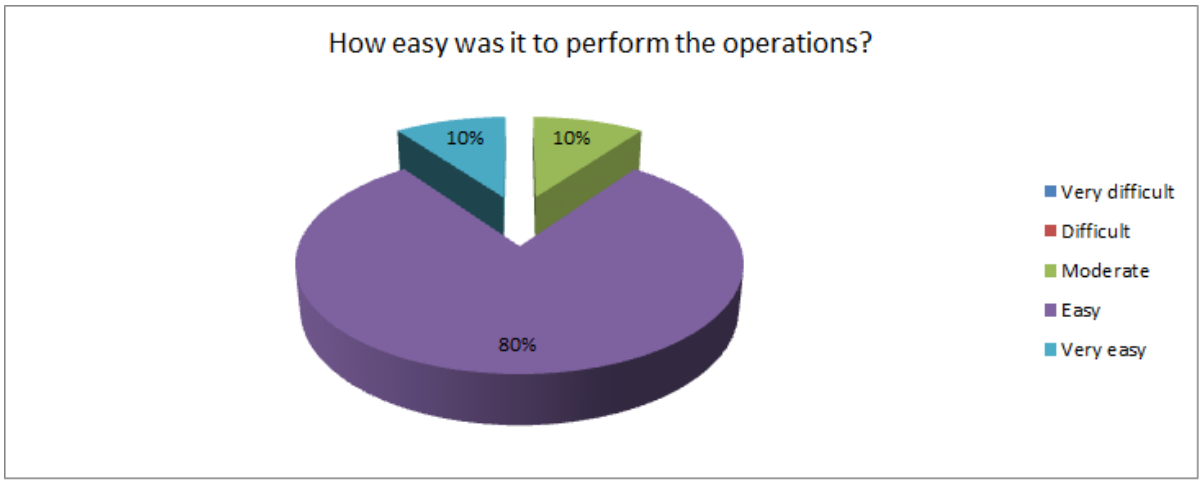

Figure 7.28: SUS: How Easy To Use.

3. The anticipated learning curve for other people

   Based on the results in Figure 7.29, there was a strong indication that it would be quick for other people to learn how to use the system. The majority (40%) indicated that it would not take much time to learn how to use the system. The remaining participants suggested that it would be very quick (30%) to learn while others thought that it would be slow (30%).

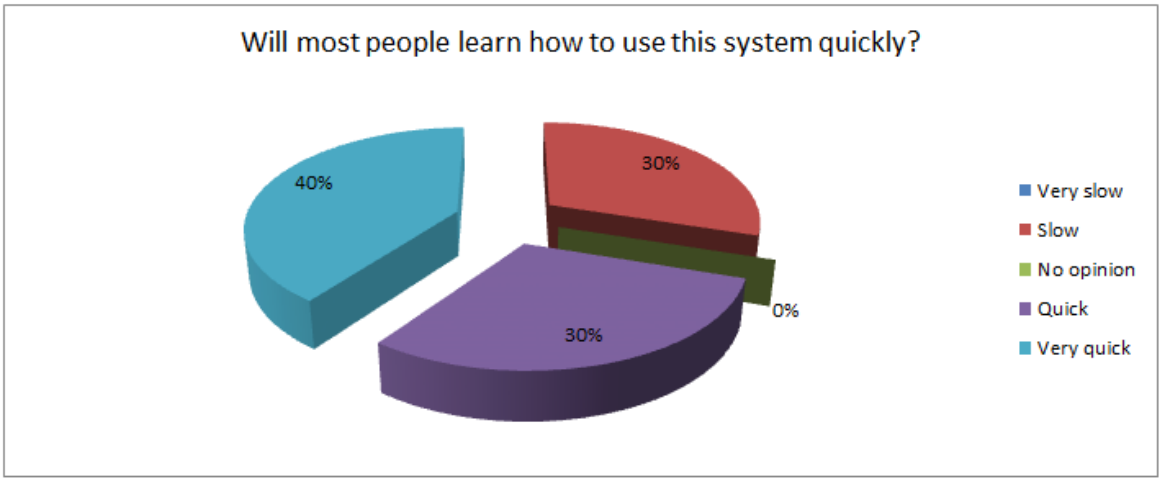

Figure 7.29: SUS: Learning How to Use.

4. Overall rating of the DRS

Figure 7.30 shows the results of the overall rating of the DRS. The results show the ratings of the satisfaction with the system: excellent (40%); very good (20%); and good (40%).

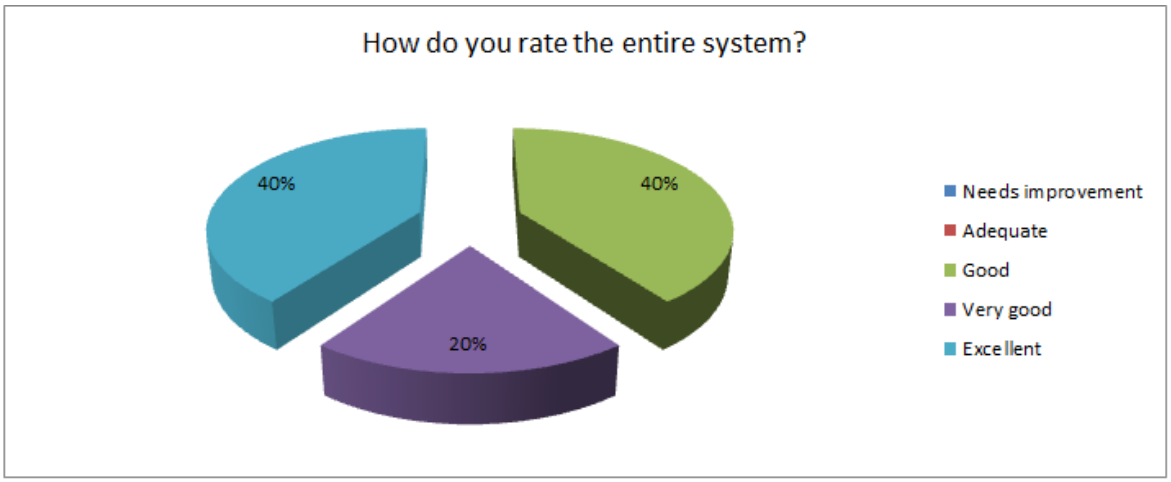

Figure 7.30: SUS: Rating for the System.

7.6 Discussion

The outcome of the functional and non functional tests, presented above, provide the assessment of the requirements specifications of the DRS presented in Section 4.1.3. The functional test results in Section 7.3 show the successful operation of the server application components of the DRS. The OSGi bundles of the DRS meet the requirements of OSGi application development in order to run in TeleWeaver: a service hosting environment designed for the context of South Africa, as highlighted in Section 2.5.3.2. The results of the configurations on the Kannel SMS and WAP gateways show how the ridesharing service (in TeleWeaver) is accessible to mobile phones using SMS and WAP technologies. The functional test results for the DRS, provided in Section 7.4, demonstrate the features of dynamic ridesharing that were highlighted in Section 2.2.2.1. As such, the DRS provides the dynamic ridesharing service in a customized way that suits the hitch-hiking culture in the Eastern Cape. In addition, the development of the DRS using the guidelines under SLL ensures the use of appropriate software solutions that are sustainable in the context of South Africa.

The non functional test through a usability study has provided the assessment results by a sample of users. The test results provide a measure of the effectiveness, efficiency and satisfaction of the DRS, which were explained in Section 4.1.3.3. The relevance of the DRS as a tool for organizing hitch-hiking has been emphasized by the 100% response (in

both the urban and rural contexts) that the system provides operations that represent the hitch-hiking arrangements. The DRS enables the arrangement of rideshare trips by people without acquaintances who share the same destination, as is the case with other example DRSs (explained in Section 2.2.2.7) and in the hitch-hiking travels studied in the Eastern Cape (covered in Chapter 3). This indicates a positive step in developing an appropriate ridesharing system for hitch-hiking in South Africa.

During the usability tests, all participants were able to complete their designated tasks. The results showed that people in both the urban (78%) and the rural (90%) found the operations using SMS messages to be easy to use. Performing the ridesharing operations using SMS messages provided familiar operations due to the participants' background knowledge of SMS. The main challenge was remembering the keywords and properly structuring the requests, as mentioned in the disadvantages of SMS application in Section 2.4.3.2. However, participants had a user manual which they could reference when composing the SMS messages. This could be the reason why some participants felt that the learning process of the system would be slow (11% in urban area and 30% in rural area) or failed to provide an opinion (22% in the urban area). Nevertheless, this problem can be solved through frequent use of the application which would allow a user to get acquainted with the message structures for the different operations; or through the use of menu based operations as in USSD and WAP applications.

The overall rating of the DRS, in Figures 7.25 and 7.30, show that all the participants were satisfied with the operations they carried out in the tests. This indicates that people can adopt the system when they are given time to test it. Further improvements on the current version of the DRS could also be made to increase this adoption.

The use of SMS technology to deliver the ridesharing service makes the DRS compatible with a variety of mobile phones owned by the targeted drivers and hitch-hikers in this research. During the usability tests, all participants owned a mobile phone, which they used when carrying out the tests. This satisfied the requirement for the DRS to reach many mobile phone types (old and new models), which is part of the non functional requirements presented in Section 4.1.3.3.

7.7 Summary

This chapter has discussed the tests that were conducted to verify whether or not the developed DRS meets the objective of this research. The functional test results show how the DRS works by including the user interaction screens. The usability test results provide the feedback from a sample of users who understand and participate in hitch-hiking in the

study area. In conclusion, a discussion of the results provided an analysis of the obtained results after performing tests in both the simulated and live environments.

Chapter 8

Conclusion

This thesis has presented work that was performed to develop a DRS for hitch-hiking travels in South Africa. It begun by providing the details of the available DRSs that are used in most developed nations and their primary objectives. By consulting the related literature on hitch-hiking and carrying out a study on how it is done in selected urban and rural areas of the Eastern Cape, the requirements of the DRS were identified. This was followed by a discussion on how the DRS was designed and its implementation using the appropriate open source solutions. Thereafter, the functional and non-functional test results which verify the achievement of the objective of this research were presented. This chapter concludes the thesis by an assessment of the objective of this research, highlighting the contributions, challenges and outlining suggestions for future work.

8.1 Assessment of Research Objective

The objective of this research is revisited to discuss the degree to which the work presented in this thesis achieved the main goal. This is presented in a discussion of the theoretical and practical contributions of this thesis, followed by the challenges that were encountered in the process.

8.1.1 Theoretical Contributions

The thesis has presented literature on vehicle ridesharing and DRSs that assist in organizing ad-hoc ridesharing using mobile phones, mostly belonging in the context of developed nations. This provided background knowledge on the objectives and general features of DRSs, which formed the basis for developing a relevant DRS to the context of South Africa (a developing nation). The availability of technical documentation of free DRSs such as

OpenRide (an open source project) provided the in-depth understanding of how the various components are designed in modern ridesharing systems. Based on the understanding of the hitch-hiking culture in the Eastern Cape, a DRS was designed to provide familiar methods of ad-hoc trip arrangements. This includes the use of a matching algorithm that uses location short-codes (for origin and destination names) and also a time window for ride offers, which simplifies the departure time arrangements by drivers. The functional and non functional test results indicate a positive step in an effort to provide a design of an appropriate DRS (for hitch-hiking) that is accessible by mobile phones owned by people in urban and rural contexts of South africa, which include marginalized areas.

8.1.2 Practical Contributions

The functional test results have demonstrated how a driver and commuter can arrange a rideshare trip without any acquaintaces, at any time of the day, by performing the required operations using a mobile phone (through SMS or the WAP application). The information of users, vehicles and organized trips is recorded and made available to registered users. As such, enquiries can be made about a particular rideshare trip to get details such as the driver, passengers and the vehicle particulars. In addition, a registered next of kin for a user can be informed about a hitch-hiking trip that they are participating in by sending a request to the DRS. This solves the problem of the lack of information to a relative or friend when a person is travelling by hitch-hiking. These are some of the requirements that people who use hitch-hiking travels indicated as important in order to improve on the current hitch-hiking process, as we found in the study presented in Chapter 3. The usability tests on the DRS made drivers and hitch-hikers aware of the possibility to arrange hitch-hiking trips and enquire details of a particular rideshare trip using their mobile phones. This provided them an opportunity to learn about the use of a DRS in ad-hoc ridesharing, which improve on the shortfalls observed in the ordinary methods such as hitch-hiking.

The existing free DRSs for mobile phones target primarily smartphone users as found in Section 2.2.2. This limits the number of people who can access the ridesharing services as indicated in the statistics of the mobile phones that are owned globally (in Section 2.4.2). Therefore, most people who participate in hitch-hiking (in South Africa) cannot access the available DRSs for their benefits. The developed DRS supports mobile phones of different capabilities through the methods of SMS and WAP. These access methods suit the targeted drivers and hitch-hikers in the urban and rural areas of South Africa, who are already familiar with SMS and web browsing technologies on their mobile phones. In this way, the developed DRS addresses the gap created by the modern DRSs, which focus

on smartphones and the operating environments that have high speed mobile internet services.

8.1.3 Challenges

In Chapter 2, we found that trust is a major issue in implementing DRSs since they involve users who are not acquainted. In our DRS, the identification details provided by the user (e.g. names) must be trustworthy to bring confidence among users when they request for travel partners. This would require extra verification on the personal details provided during registration. In Section 4.1.1, RICA [65] was explained as a system which keeps the personal identification information of mobile phone users in South Africa. Linking the DRS with RICA could provide a solution for the verification of user details by compairing with information on documents which certify identity (e.g. National ID or Passport). The user verification could be further enhanced by conducting a search for any criminal record of the user from relevant authorities (e.g. the Police). In this way, the issues of trust and security can be minimized.

Providing origin and destination location names in an SMS message is error prone as names can be misspelled, which could result in missing out on a matching ride partner. Some users may struggle to provide correct location names considering that the target population of users (in this research) consists of some people who are illiterate. Therefore, an extra function would be required to handle incorrect location names by auto-correcting or suggesting location names after verifying the location information in external applications, such as Google maps. In smartphones, input errors of location names are minimized through the capabilities of auto-detection of locations (using GPS and map data) and auto-complete for the location names.

The costs that are incurred when using SMS messages could restrict the number of people using the system. Section 2.4.3.3 highlighted the high cost of SMS messaging in South Africa through the observed increase in use of available cheaper alternative instant messaging services such as WhatsApp [109] and Mxit [68]. Therefore, the challenge would be on negotiating with the mobile phone operators to provide discounted SMS messages for the DRS. This would help most drivers and hitch-hikers afford to use the ridesharing service espencially in marginalized areas such as the Dwesa communities: where hitch-hiking is a major alternative mode of transportation due to limited affordable public transport services.

The effects of mobile phone device fragmentation were experienced during the tests on the WAP application, using different mobile phone brands. It was observed that the user

interfaces were not consistent across the different mobile phone brands; and in some cases within old and latest models of a particular brand. The notable differences were observed in the input mechanism and the screen sizes which provides different layouts of web pages. This can confuse some users when they migrate to a different mobile phone handset due to the different experience provided when performing the same operations of the DRS.

The existing disagreements between the taxi operators and private car owners, who pick-up hitch-hikers, could have effects on the active participation of users in the DRS. As found in the study on hitch-hiking by Sicwetsha [94], presented in Section 2.3.3.3, private car drivers risk attacks by taxi operators when they pick-up hitch-hikers. Therefore, it cannot be predicted how the general public (particularly the taxi operators, private car drivers and commuters) would accept the DRS once it is deployed for public use.

8.2 Future Work

The work that this thesis has presented provides the initial step in providing a system that is tailor-made for the dynamic ridesharing of hitch-hiking in South Africa. The modular design of the system using OSGi technology provides the opportunity for the expansion of the technical capabilities of the system.

The DRS should be refined by conducting further usability tests that include field tests involving drivers and hitch-hikers. Thereafter, additional functionalities such as GPS and location based services must be added to make the system relevant for smartphones in order to provide the extra features discussed in Section 2.2.2.2. In South Africa, the number of smartphone users continue to grow each year, indicating that people are slowly moving away from feature phones [84]. This should help to provide simplified arrangement of hitch-hiking trips and also improve the security aspect (e.g. logging the trip progress in real-time), as is the case in similar DRSs such as Avego [9].

8.3 Concluding Remarks

The thesis has outlined the design and implementation process of a DRS for drivers and commuters who participate in hitch-hiking travels. Some of the reasons for the choice to hitch-hike or pick-up hitch-hikers in South Africa have been found to be different to those in developed nations. The hitch-hiking culture and the operating environment are unique to this context such that the available free DRSs do not satisfy all the requirements to enable many people to use their ad-hoc ridesharing services. The thesis has presented how the developed DRS is accessible by mobile phones of different capabilities and the results

of a usability study which indicate an initial step forward in implementing an appropriate system for ad-hoc ridesharing of hitch-hiking.

References

[1] Statistics South Africa. Census 2011. Online: http://www.statssa.gov.za/Publications/P03014/P030142011.pdf, [Last accessed on 25/07/13], October 2012.

[2] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang. Sustainable Passenger Transportation: Dynamic Ride-Sharing. Online: http://www2.isye.gatech.edu/people/faculty/Alan_Erera/pubs/aesw09.pdf, [Last accessed on 10/12/12], 2009.

[3] Arrive Alive. Minibus Taxis and Road Safety. Online: http://www.arrivealive.co.za/pages.aspx?i=2850, [Last accessed on 7/10/12].

[4] Global Intelligence Alliance. Native or Web Application. Online: http://www.globalintelligence.com/insights-analysis/white-papers/native-or-web-application-how-best-to-deliver-cont/, [Last accessed on 22/12/12], 2010.

[5] OSGi Alliance. OSGi Technology. Online: http://www.osgi.org/Main/HomePage, [Last accessed on 10/11/12].

[6] A. Amey. Real-Time Ridesharing: Exploring the Opportunities and Challenges of Designing a Technology-based Rideshare Trial for the MIT Community. Master's thesis, Massachusetts Institute of Technology, 2010.

[7] Apache.org. Apache Felix. Online: http://felix.apache.org/site/index.html, [Last accessed on 10/11/12].

[8] Apache.org. Apache Maven. Online: http://maven.apache.org/, [Last accessed on 11/11/12].

[9] Avego. Avego Real-time Ridesharing. Online: http://www.avego.com/products/real-time-ridesharing/, [Last accessed on 30/05/12].

[10] N. Balani and R. Hathi. *Apache CXF Web Service Development*. Packt Publishing, 2009.

[11] N. Bartlett. Bndtools. Online: http://bndtools.org/, [Last accessed on 12/11/12], 2011.

[12] G. Beger and A. Sinha. South African Mobile Generation. Online: http://www.unicef.org/southafrica/SAF_resources_mobilegeneration.pdf, [Last accessed on 15/10/12]., May 2012.

[13] Carpoolmates. Carpool Matching System. Online: http://www.carpoolmates.co.za/, [Last accessed on 7/10/12].

[14] Washington State Transportation Center. Bellevue Smart Traveler: Design, Demonstration, and Assessment. Online: http://www.wsdot.wa.gov/Research/Reports/300/376.1.htm, [Last accessed on 24/10/12].

[15] N. Chan and S. Shaheen. Ridesharing in North America: Past, Present and Future. Online: http://tsrc.berkeley.edu/ridesharingNA, [Last accessed on 10/12/12]., 2011.

[16] V. Chaube. Understanding and Designing for Perceptions of Trust in Rideshare Programs. Masters thesis, Virginia Polytechnic Institute and State University, July 2010.

[17] N1 Lift Club. N1 Lift Club Gauteng. Online: http://www.n1liftclub.co.za/, [Last accessed on 7/10/12].

[18] A. Cogoluegnes, T. Templier, and A. Piper. *Spring Dynamic Modules in Action*. Manning Pubs Co Series. Manning Publications, 2010.

[19] JBoss Community. Hibernate. Online: http://www.hibernate.org/downloads, [Last accessed on 11/12/12].

[20] SpringSource Community. SpringSource Tool Suite. Online: http://www.springsource.org/spring-tool-suite-download, [Last accessed on 11/12/12].

[21] G. Correia and J. Viegas. Advanced OR and AI Methods in Transportation Car Pooling Clubs: Solution For The Affiliation Problem In Traditional/Dynamic Ridesharing System. Online: http://www.iasi.cnr.it/ewgt/16conference/ID92.pdf, [Last accessed on 10/12/12].

[22] Apache CXF. RESTful Services, JAX-RS. Online: http://cxf.apache.org/docs/jax-rs.html, [Last Accessed on 23/08/12].

[23] E. Deakin, K. Frick, and K. Shively. Markets for Dynamic Ridesharing? Case of Berkeley, California. University of California Transportation Center, Working Papers, University of California Transportation Center, 2011.

[24] South Africa Department of Transport. The First South African National Household Travel Survey 2003. Online: http://www.arrivealive.co.za/document/household.pdf, [Last accessed on 7/10/12], August 2005.

[25] eBay. URL: http://www.ebay.com/, [Last accessed on 09/01/13].

[26] Eclipse. Equinox OSGi. Online: www.eclipse.org/equinox/, [Last accessed on 10/11/12].

[27] Rhodes University Environment. Travel Wise. Online: http://www.ru.ac.za/environment/action/travelwise/, [Last accessed on 3/10/12].

[28] Roy Thomas Fielding. *Architectural styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000. AAI9980887.

[29] FindALift.co.za. Find a lift. Online: http://www.findalift.co.za, [Last accessed on 8/10/12].

[30] A. Fink, B. Rodrigues, S. Tolj, A. Syvänen, A. Malysh, L. Wirzenius, and K. Marjola. Kannel 1.5.0 User Guide, Open Source WAP and SMS gateway. Online: http://www.kannel.org/download/1.5.0/userguide-1.5.0/userguide.pdf, [Last accessed on 9/05/12].

[31] B. Fling. *Mobile Design and Development: Practical Concepts and Techniques for Creating Mobile Sites and Web Apps - Animal Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[32] Codehaus Foundation. Jetty. Online: http://jetty.codehaus.org/jetty/, [Last accessed on 15/11/12].

[33] The Apache Foundation. Apache Tomcat. Online: http://tomcat.apache.org/, [Last accessed on 15/11/12].

[34] A. Goldstuck. Internet Matters: The Quiet Engine of the South African Economy. Online: www.internetmatters.co.za, [Last accessed on 12/10/12].

[35] Google. Google Web Toolkit. Online: https://developers.google.com/web-toolkit/, [Last access on 4/05/12].

[36] Kannel Group. Kannel: Download Pages. Online: http://www.kannel.org/download.shtml, [Last accessed on 11/12/12].

[37] GSMA. African Mobile Observatory 2011. Online: http://www.gsmworld.com/our-work/public-policy/, [Last accessed on 14/06/12].

[38] D. Guermeur and A. Unruh. *Google App Engine Java and Gwt Application Development.* Packt Publishing, Limited, 2010.

[39] Gumtree. South Africa Carpool and Rideshare. Online: http://www.gumtree.co.za/f-Community-carpool-rideshare-W0QQAdTypeZ1QQCatIdZ9035QQisSearchFormZtrue, [Last accessed on 7/10/12].

[40] R.S. Hall, K. Pauls, S. McCulloch, and D. Savage. *OSGi in Action: Creating Modular Applications in Java.* Manning Pubs Co Series. Manning, 2011.

[41] S. Heinrich. Implementing Real-time Ridesharing in the San Francisco Bay Area. Master's thesis, Mineta Transportation Institute San Jose State University, 2010.

[42] Developer's Home. Example Applications of SMS Messaging. Online: http://www.developershome.com/sms/sms_tutorial.asp?page=egApps, [Last Accessed on 14/12/12].

[43] M. Kaigwa. Why You Must Never Forget About the Humble Feature Phone in Africa. Online: http://afrinnovator.com/blog/2012/02/02/why-you-must-never-forget-about-the-humble-feature-phone-in-africa, [Last accessed on 17/10/12], February 2012.

[44] G. King, C. Bauer, E. Bernard, and S. Ebersole. Hibernate 3.6.10.Final, Community Documentation. Online: http://docs.jboss.org/hibernate/core/3.6/quickstart/en-US/html/, [Last Accesed on 12/09/12].

[45] A. Kothari. Genghis - A Multiagent Carpooling System. Bsc in Computer Science Dessertation work, May 2004. University of Bath.

[46] K. Kurbel and A. Dabkowski. Dynamic WAP Content Generation with the Use of Java Server Pages. Online:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.3195&rep=rep1&type=pdf, [Last accesed on 14/01/13].

[47] D. Kurka. GWT PhoneGap library. Online: http://code.google.com/p/mgwt/wiki/GettingStarted, [Last accessed on 21/09/12].

[48] Helsinki Living Lab. Helsinki Living Lab. Online: www.helsinkilivinglab.fi, [Last accessed on 30/09/12], 2012.

[49] MIT Media Lab. William J. Mitchell. Online. http://web.media.mit.edu/ wjm/ [Last accessed on 21/10/12].

[50] Sekhukhune Living Lab. Sekhukhune Living Lab. Online: http://www.c-rural.eu/Southafrica-LivingLab/, [Last accessed on 10/12/12].

[51] Siyakhula Living Lab. The Siyakhula Living Lab. Online: http://siyakhulall.org/, [Last accessed on 30/09/12].

[52] Siyakhula Living Lab. The Siyakhula Living Lab: An Important Step for South Africa and Africa. Online: http://siyakhulall.org/sites/default/files/SiyakhulaLL_Booklet_2012.pdf, [Last accessed on 02/02/13].

[53] A. Levofsky and A. Greenberg. Organized Dynamic Ride Sharing: The Potential Environmental Benefits and the Opportunity for Advancing the Concept. Report, Transportation Research Board, January 2001.

[54] LLiSA. Living Labs in Southern Africa. Online: http://llisa.meraka.org.za/index.php/Moutse_LL:Project_Overview, [Last Accessed on 10/12/12].

[55] S. Loucks. Report for Haliburton County's Rural Transportation Options Committee of Environment Haliburton. Online: http://www.haliburtoncooperative.on.ca, [Last accessed on 21/10/12].

[56] Canonical Ltd. Get Ubuntu. Online: http://www.ubuntu.com/download, [Last accessed on 11/12/12], 2012.

[57] Makewave. Knopflerfish- Open Source OSGi Service Platform. Online: http://www.knopflerfish.org/, [Last accessed on 10/11/12].

[58] D. Massaro, B. Chaney, S. Bigler, J. Lancaster, S. Iyer, M. Gawade, M. Eccleston, E. Gurrola, and A. Lopez. Carpoolnow - Just-in-Time Carpooling without Elaborate Pre-planning. In Joaquim Filipe and José Cordeiro, editors, *WEBIST 2009 - Proceedings of the Fifth International Conference on Web Information Systems and Technologies, Lisbon, Portugal, March 23-26, 2009*, pages 219–224. INSTICC Press, 2009.

[59] South Africa Ministry of Transport. National Land Transport Act 2009. Online: http://www.transport.gov.za/PublicTransport.aspx, [Last accessed on 30/09/12].

[60] S. Miteche, A. Terzoli, and H. Thinyane. A Mobile Phone Solution to Improve Geographic Mobility. In *SATNAC 2011: Southern African Telecommunications Networks and Applications Conference*, 2011.

[61] S. Miteche, A. Terzoli, and H. Thinyane. A Mobile Phone Solution to Improve Geographic Mobility. In *SATNAC 2011: Southern African Telecommunications Networks and Applications Conference*, 2012.

[62] Vision Mobile. Mobile Platforms: The Clash of Ecosystems. Online: http://www.visionmobile.com/product/clash-of-ecosystems, [Last accessed on 23/10/12], November 2011.

[63] Mozilla. RESTClient a Debugger for RESTful Web Services. Online: https://addons.mozilla.org/en-us/firefox/addon/restclient/, [Last accessed on 29/11/12].

[64] H. Mpofu. Development Of An M-Payment System Prototype For A Marginalized Region (Dwesa Case Study). Master's thesis, University Of Forthare, 2011.

[65] MTN. RICA - Regulation of Interception of Communication Act. Online: http://www.mtnsp.co.za/MTNServices/RICA/Pages/Overview.aspx, [Last accessed on 18/11/12].

[66] MTN. Visiting South Africa. Online: http://www.mtn.co.za/Travel/Pages/TravellingToSA.asp [Last Accessed on 8/12/12].

[67] MvnRepository. MVN Repository. Online: http://mvnrepository.com/, [Last accessed on 15/11/12].

[68] Mxit. Mixit for Your Feature Phone. Online: http://site.mxit.com/, [Last accessed on 21/09/13].

[69] United Nations. Environmental Indicators Greenhouse Gas Emissions. Online: http://unstats.un.org/unsd/environment, [Last accessed on 12/10/12]., July 2010.

[70] Nokia. Series 40. Online: http://www.developer.nokia.com/Develop/Series_40/Platform/, [Last accessed on 1/12/12].

[71] Ride Now. Pilot Tests of Dynamic Ridesharing. Online: http://www.ridenow.org, [Last accessed on 24/10/12].

[72] L. Ntshinga, M. Thinyane, and A. Terzoli. Blueprints for Software Components within Teleweaver, a Custom OSGi Container. In *SATNAC 11: Southern African Telecommunications Networks and Applications Conference*, 2011.

[73] B. Nussbaum. African Culture and Ubuntu. Online: http://barbaranussbaum.com/downloads/perspectives.pdf, [Last accessed on 10/12/12], 2003.

[74] Massachusetts Institute of Technology. MIT "Real-Time" Rideshare Research. Online: http://ridesharechoices.scripts.mit.edu/home/real-time, [Last accessed on 04/10/12], 2009.

[75] South Africa Online. South Africa Transport. Online: http://www.southafrica.co.za/about-south-africa/transport/, [Last accessed on 7/10/12].

[76] OpenRide. OpenRide Ridessharing 2.0. Online: http://www.open-ride.com/, Online [Last accessed on 25/04/12].

[77] Oracle. Java API for RESTful Services JAX-RS. Online: http://jax-rs-spec.java.net/, [Last accessed on 13/11/12].

[78] Oracle. Java Persistence API. Online: http://www.oracle.com/technetwork/java/javaee/tech/p
jsp-140049.html, [Last accessed on 13/11/12].

[79] Oracle. Java SE Technologies- Database. Online: http://www.oracle.com/technetwork/java/javase/jdbc/index.html, [Last accessed on 11/11/12].

[80] Oracle. Oracle Top Link. Online: http://www.oracle.com/technetwork/middleware/toplink/ov
[Last accessed on 13/11/12].

[81] Oracle. MySQL Download. Online: http://dev.mysql.com/downloads/, [Last accessed on 11/12/12], 2012.

[82] PayPal. URL: https://www.paypal.com/za/webapps/mpp/home, [Last accessed on 03/02/13].

[83] Pickuppal. Pickuppal on Your Way. Online: http://www.pickuppal.com/pup/intro.html, [Last accessed on 22/10/12].

[84] Our Mobile Planet. Mobile Behaviour. Online: http://www.thinkwithgoogle.com/mobileplanet/en/, [Last accessed on 10/09/12].

[85] M. Rao. Mobile Africa Report 2012. Sustainable Innovation Ecosystems. Technical report, http://www.mobilemonday.net/reports/MobileAfrica_2012.pdf, 2012.

[86] Restlet. The Leading Web API Framework for Java. Online: http://www.restlet.org/, [Last accessed on 13/11/12].

[87] RLabs. LLiSA Living Labs in Southern Africa. Online. http://llisa.net/, [Last accessed on 21/10/12].

[88] Daniel Rubio. *Pro Spring Dynamic Modules for OSGi Service Platforms.* Apress, Berkely, CA, USA, 2009.

[89] Brand SA. About South Africa. Online: http://www.southafrica.info, [Last accessed on 28/10/12].

[90] J. Samalenge and M. Thinyane. Developing SOA Wrappers for Communication Purposes in Rural Areas. Master's thesis, University of Fort Hare, 2010.

[91] J. Sandoval. *RESTful Java Web Services: Master Core REST Concepts and Create RESTful Web Services in Java.* From technologies to solutions. Packt Publishing, Limited, 2009.

[92] J. Sauro. Measuring Usability. Online: https://www.measuringusability.com/sus.php, [Last Accessed on 7/12/12].

[93] A. Seddighi. *Spring Persistence with Hibernate: Build Robust and Reliable Persistence Solutions for Your Enterprise Java Application.* From technologies to solutions. Packt Publishing, Limited, 2009.

[94] M. Sicwetsha. Hitch-Hiking Research Report, Reasons behind hitch-hiking in the Eastern Cape. Online: http://www.ectransport.gov.za/uploads/Reports/reasons-behind-hitch-hiking-in-the-eastern-cape1.pdf, [Last accessed on 12/04/12].

[95] SpringSource. Spring Web Service. Online: http://static.springsource.org/spring-ws/sites/2.0/ [Last accessed on 13/11/12].

[96] SpringSource. SpringSource Enterprise Bundle Repository. Online: http://ebr.springsource.com/repository/app/, [Last accessed on 11/11/12].

[97] Chrome Web Store. Developer Tools Simple Rest Client. Online: https://chrome.google.com/webstore/category/ext/11-web-development, [Last accessed on 29/11/12].

[98] Reedhouse Systems. Reedhouse systems. Online: http://reedhousesystems.com/, [Last accessed on 2/04/12].

[99] Reedhouse Systems. Reedhouse Systems Wikidoc. Online: http://reedhousesystems.com/wikidoc, [Last accessed on 13/11/12].

[100] Techcrunch. It's Still A Feature Phone World: Global Smartphone Penetration at 27 Online: http://techcrunch.com/2011/11/28/its-still-a-feature-phone-world-global-smartphone-penetration-at-27/, [Last accessed on 22/10/12], November 2011.

[101] M. Thinyane. *A knowledge-oriented, context-sensitive architectural framework for service deployment in marginalized rural communities.* PhD thesis, Rhodes University, 2009.

[102] H.G. Timmermans. Rural Livelihoods at Dwesa/Cwebe: Poverty, Development and Natural Resource Users on the Wild Coast, South Africa. Master's thesis, Rhodes University, 2004.

[103] W. Trochim. Nonprobability Sampling. Online: http://www.socialresearchmethods.net/kb/sampnon.php, [Last accessed on 1/11/12].

[104] D. Ungemah, G. Goodin, and C. Dusza. Examining Incentives and Preferential Treatment of Carpools on Managed Lane Facilities. *Journal of Public Transportation*, 10, 2007.

[105] C. Walls. *Modular Java: Creating Flexible Applications with OSGi and Spring.* Pragmatic Bookshelf Series. Pragmatic Bookshelf, 2009.

[106] C. Walls and R. Breidenbach. *Spring in Action.* Manning Publications, 2008.

[107] WAPProof. WAPProof WAP Browser for Developers of Mobile Content. Online: http://www.wap-proof.com/, [Last accessed on 1/12/12].

[108] R. Wessels. Combining Ridesharing & Social Networks. Online: www.utwente.nl/ctw/aida/education/ITS2-RW-Pooll.pdf, [Last accessed on 2/10/12].

[109] WhatsApp. Simple. Personal. Real Time Messaging. Online: http://www.whatsapp.com/, [Last accessed on 21/10/12].

[110] L. Wirzenius. Kannel Architecture and Design. Online: rt-lab.cs.nthu.edu.tw/course/cs5241/papers/wap_arch.pdf, [Last accessed on 14/11/12].

[111] Worldwideworx. Leaders in Technology Market Research in South Africa. Online: http://www.worldwideworx.com/, [Last accessed on 20/10/12]., 2012.

[112] Worldwideworx. The Mobile Internet in South Africa 2012. Technical report, www.worldwideworx.com, 2012.

[113] Zimride. Grab a Friend. Online: http://www.zimride.com/, [Last accessed on 23/10/12].

Appendix A

Questionnaires

A.1 Hitch-Hiking Questionnaire

See Figure A1 and A2 below.

A.2 Usability Testing

A.2.1 Background Questionnaire: User Background

Question	Yes	No
Do you travel by hitch-hiking?		
Do you pick-up hitch-hikers? (If you drive)		
Are you computer literate?		
Do you own a cell phone?		
Do you use SMS?		

QUESTIONNAIRE ON HITCH HIKING

Answer questions in the provided spaces and tick where specified.

1. What is your gender type? (Tick your selection)
   - ☐ Male
   - ☐ Female

2. In which age group do you belong? (Tick your selection)
   - ☐ Less than 18 years
   - ☐ 18 - 30 years
   - ☐ 31 – 60 years
   - ☐ Over 60 years

3. What is your occupation?

   _____

4. Do you travel by hitch hiking method? (Tick your selection)
   - ☐ Yes
   - ☐ No (skip to question 8)

5. If yes, in what role? (Tick your selection)
   - ☐ Commuter
   - ☐ Driver
   - ☐ Both

6. As a commuter, how do you find an appropriate hitch hiking spot for your intended destination?

   _____

   _____

7. How often do you use hitch hiking travel?

   _____

   _____

8. What do you think are the benefits of hitch hiking travel?

   _____

   _____

   _____

9. What do you think are the risks associated with hitch hiking travel?

   _____

   _____

10. What recommendations would you suggest to improve hitch hiking travel?

    _____

    _____

    _____

11. Do you own a mobile phone? (Tick your selection)
    - ☐ Yes
    - ☐ No

12. What brand is your mobile phone i.e. Nokia, LG, Siemens, Blackberry? (include model where possible)

    _____

    _____

13. Do you use internet on your phone? (Tick your selection)
    - ☐ Yes
    - ☐ No

14. If yes, how often?
    - ☐ At least once a day

Figure A.1: Questionnaire Part 1

- ☐ Once a week
- ☐ Not often
- ☐ Never

15. What application do you use for internet connection? (Built in browser, Opera Mini, Firefox, etc.)

_____

16. What information would you prefer to have, before participating in a hitch hiking travel?
    a) As a driver?

_____

_____

    b) As a commuter?

_____

_____

17. What estimated waiting-time would you allow in an organized hitch hiking travel? (From the time you make yourself available to the beginning of travel)

_____

Figure A.2: Questionnaire Part 2.

A.2.2 System Usability Scale Questionnaire

Does this application provide operations that represent hitch-hiking travel arrangements?				Yes	No
How easy was it to perform the operations?	Very difficult	Difficult	Moderate	Easy	Very easy
Is the application important for hitch-hiking travels?	Not at all	Not very	No opinion	Somewhat	Extremely
In your opinion, will most people (drivers and commuters) learn how to use this system quickly?	Very slow	Slow	No opinion	Quick	Very quick
How do you rate the entire system?	Needs improvement	Adequate	Good	Very good	Excellent

Appendix B

Code Snippets

B.1 Project Layout

Figure B.1: The DRS API Bundle Project

B.2 The *application-config.xml* file

Figure B.2 gives an extract of the *application-config.xml* file showing the bean definitions and their wiring for DI.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:osgi="http://www.springframework.org/schema/osgi"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
              http://www.springframework.org/schema/tx
              http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
              http://www.springframework.org/schema/context
              http://www.springframework.org/schema/context/spring-context-3.0.xsd
              http://www.springframework.org/schema/aop
              http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
              http://www.springframework.org/schema/osgi
              http://www.springframework.org/schema/osgi/spring-osgi.xsd">

  <!-- Configuration for the rewards application.  Beans here define the heart of the application logic. -->

    <context:annotation-config/>

    <!-- Instructs the container to look for beans with @Transactional and decorate them -->
    <tx:annotation-driven transaction-manager="transactionManager"/>


    <bean id="rideshareService" class="com.reedhousesystems.services.core.ridesharing.impl.RideshareServiceImpl">
        <constructor-arg ref="hibernateRideshareDao"/>
    </bean>

    <!-- Loads profile from the data source -->
    <bean id="hibernateRideshareDao" class="com.reedhousesystems.services.core.ridesharing.dao.HibernateRideshareDao">
        <constructor-arg ref="sessionFactory"/>
    </bean>

     <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location" value="file:~/../configuration/database.properties"/>
    </bean>

    <!-- A data source -->
    <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="${database.driverClassName}" />
        <property name="url" value="${database.url}" />
        <property name="username" value="${database.username}" />
        <property name="password" value="${database.password}" />
    </bean>
```

Figure B.2: Spring Bean Configurations.

Figure B.3 shows part of the *application-config.xml* for the datasource and Hibernate configurations.

```
    <!-- A data source -->
    <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="${database.driverClassName}" />
        <property name="url" value="${database.url}" />
        <property name="username" value="${database.username}" />
        <property name="password" value="${database.password}" />
    </bean>

    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
        <property name="dataSource" ref="dataSource"></property>
        <property name="configurationClass">
            <value>org.hibernate.cfg.AnnotationConfiguration</value>datasource2
        </property>
        <property name="annotatedClasses">
            <list>
                <value>com.reedhousesystems.services.core.ridesharing.model.Hitchhiker</value>
                <value>com.reedhousesystems.services.core.ridesharing.model.HitchhikerPoint</value>
                <value>com.reedhousesystems.services.core.ridesharing.model.NextOfKin</value>
                <value>com.reedhousesystems.services.core.ridesharing.model.Person</value>
                <value>com.reedhousesystems.services.core.ridesharing.model.RoutePoint</value>
                <value>com.reedhousesystems.services.core.ridesharing.model.Trip</value>
                <value>com.reedhousesystems.services.core.ridesharing.model.TripRoutePoint</value>
                <value>com.reedhousesystems.services.core.ridesharing.model.Vehicle</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</prop>
                <prop key="hibernate.c3p0.min_size">5</prop>
                <prop key="hibernate.c3p0.max_size">20</prop>
                <prop key="hibernate.c3p0.timeout">1800</prop>
                <prop key="hibernate.c3p0.max_statements">50</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
                <prop key="hibernate.default_batch_fetch_size">5</prop>
                <prop key="hibernate.generate_statistics">true</prop>
                <prop key="hibernate.use_sql_comments">true</prop>
                <prop key="hibernate.jdbc.batch_size">10</prop>

                <prop key="javax.persistence.validation.mode">none</prop>
            </props>
        </property>
    </bean>
     <!--  A transaction manager for working with Hibernate SessionFactories -->
    <bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <osgi:service ref="rideshareService" interface="com.reedhousesystems.services.core.ridesharing.RideshareService" />

</beans>
```

Figure B.3: Spring Datasource and Hibernate Configurations

B.3 The RESTful Web Service Bundle

B.3.1 Setting the Configurations

```
public class RideshareBundleActivator implements BundleActivator {

    private ServiceRegistration registration;
    public void start(BundleContext bc) throws Exception {
        ServiceReference ref = bc.getServiceReference(RideshareService.class.getName());
        RideshareService service = (RideshareService)bc.getService(ref);

        Dictionary<String, String> props = new Hashtable<String, String>();
        props.put("service.exported.interfaces", "*");
        props.put("service.exported.configs", "org.apache.cxf.rs");
        props.put("service.exported.intents", "HTTP");
        //props.put("org.apache.cxf.rs.httpservice.context", "/rideshare");
        props.put("org.apache.cxf.rs.address", "http://localhost:9090/");
        RideshareRestService rs = new RideshareRestService();

        if(service != null){
            rs.setRideService(service);

        }else{
            System.out.println("There is a problem !!!");
        }
        registration = bc.registerService(RideshareRestService.class.getName(),rs, props);
```

Figure B.4: Apache CXF RESTful Web Service Configurations

B.3.2 JAX-RS annotations for a RESTful implementation

The code extract below show the implementation of the RESTful web service bundle with the JAX-RS annotations and the ridesharing service referrenced using a constructor method.

```
import java.text.ParseException;

@Path("rideshare")
public class RideshareRestService {

    public RideshareRestService() {

    }
    private RideshareService rideService;

    public void setRideService(RideshareService rideService) {
        this.rideService = rideService;
    }

    /*
     * * Register new user
     */

    @POST
    @Consumes("text/xml")
    @Produces("text/plain")
    @Path("/users")
    public String addUser(Message kannelXml) {
```

Figure B.5: The RESTful Service Implementation Class

B.4 The WAP Implementation Bundle

B.4.1 The *RideshareController* Class

The code extract in Figure B.5 shows the implementation of the web bundle using the Spring MVC. The @RequestMapping annotations complete the full URL address of each method. All methods return a *ModelAndView* object which contains the information and the name of the JSP file to render the view. The ridesharing service is referenced using the @Autowired annotation.

```
@Controller
public class RideshareController {

    @Autowired
    private RideshareService rideshareService;

    public RideshareService getRideshareService() {
        return rideshareService;
    }

    public void setRideshareService(RideshareService rideshareService) {
        this.rideshareService = rideshareService;
    }

    @RequestMapping("/index.htm")
    public ModelAndView login(HttpServletRequest request,HttpServletResponse response) {

        HttpSession session =  request.getSession();
        if (!session.isNew()){
            session.invalidate();
        }

        return new ModelAndView("login", "date", new Date());
    }
    /*
     * Call register new user form
     */
    @RequestMapping("/register.htm")
    public ModelAndView reg() {
        return new ModelAndView("register", "date", new Date());
    }
```

Figure B.6: The Controller Class of the Spring MVC Bundle

B.4.2 JSP page for WAP Clients

The code in Figure B.6 shows an example implementation of a dynamic WAP page of the DRS using JSP.

```
<%="<?xml version=\"1.0\" encoding=\"utf-8\"?>" %>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN" "http://www.wapforum.org/DTD/wml13.dtd">

<% response.setContentType("text/vnd.wap.wml"); %>
<%@ page import="com.reedhousesystems.services.core.ridesharing.PersonDTO" %>
<%@ page import="java.util.Date"%>
<wml>
 <card id="infopage" title="User Details">
    <p>
      Details of <b> ${person.username}</b> <br/><br/>
      <b>Firstname: </b> ${person.firstname}<br/>
      <b>Surname: </b> ${person.surname}<br/>
      <b>Gender: </b> ${person.gender}<br/>
      <b>Home: </b> ${person.homeLoc}<br/>
      <b>Joined on: </b> ${person.dayOfReg}<br/><br/>

      <a href="main2.htm"><%=">>"%>Main menu</a> <br/>
```

Figure B.7: Serving Dynamic WAP Content

Appendix C

The RESTful Web Service API

Table B1 shows the RESTful Webservice API for the SMS application.

Resource Method	Method Designator	@Path	Example Request
UserManual	@GET	/instructions/general	http://127.0.0.1:9090/rideshare/instructions/general
DriverInstructions	@GET	/instructions/driver	http://127.0.0.1:9090/rideshare/instructions/driver
HikerInstructions	@GET	/instructions/hiker	http://127.0.0.1:9090/rideshare/instructions/hiker
AddUser	@POST	/users	http://127.0.0.1:9090/rideshare/users/
GetUser	@POST	/users/profile	http://127.0.0.1:9090/rideshare/users/profile
UpdateUser	@POST	/users/update	http://127.0.0.1:9090/rideshare/users/update
AddNextOfKin	@POST	/users/nextofkin	http://127.0.0.1:9090/rideshare/users/nextOfKin
AddVehicle	@POST	/users/vehicle	http://127.0.0.1:9090/rideshare/users/vehicle
ChangeLanguage	@POST	/users/lang	http://127.0.0.1:9090/rideshare/users/lang
NotifyNextOfKin	@POST	/users/sendmsg	http://127.0.0.1:9090/rideshare/users/sendmsg
CreateTrip	@POST	/trips	http://127.0.0.1:9090/rideshare/trips
TripDetailsByTripCode	@POST	/trips/code	http://127.0.0.1:9090/rideshare/trips/code
TripDetailsByTripId	@POST	/trips/id	http://127.0.0.1:9090/rideshare/trips/id
RideOffer	@POST	/trips/rideoffer	http://127.0.0.1:9090/rideshare/trips/rideoffer
AcceptHitchhiker	@POST	/trips/hitchhiker	http://127.0.0.1:9090/rideshare/trips/hitchhiker
AcceptHitchhikers	@POST	/trips/allhitchhikers	http://127.0.0.1:9090/rideshare/trips/allhitchhikers
CancelRideOffer	@POST	/trips/rideoffer/cancel	http://127.0.0.1:9090/rideshare/trips/rideoffer/cancel
ViewOffers	@POST	/trips/offers	http://127.0.0.1:9090/rideshare/trips/offers
Hitchhike	@POST	/hitchhikers	http://127.0.0.1:9090/rideshare/hitchhikers
ModifyHitchhiker	@POST	/hitchhikers/hiker	http://127.0.0.1:9090/rideshare/hitchhikers/hiker
CancelHitchhiker	@POST	/hitchhikers/cancel	http://127.0.0.1:9090/rideshare/hitchhikers/cancel
FindHikers	@POST	/hitchhikers/destination	http://127.0.0.1:9090/rideshare/hitchhikers/destination
AddRoutePoint	@POST	/routepoints	http://127.0.0.1:9090/rideshare/routepoints
GetCode	@POST	/routepoints/code	http://127.0.0.1:9090/rideshare/routepoints/code
GetRoutePointsByLoc	@POST	/routepoints/loc	http://127.0.0.1:9090/rideshare/routepoints/loc

Table C.1: The RESTful Web Service API

Appendix D

SMS Application User Instructions.

SMS phone number: 0785677238.

1. GENERAL OPERATIONS

    - Register user, MESSAGE FORMAT: *Reg firstname surname username password sex language homeLocation(max 3 words)*. NOTE: homeLocation allows maximum of 3 words. EXAMPLE:reg sacha miteche smiteche pass123 m e grahamstown
    - View profile of registered user, MESSAGE FORMAT: *Profile username* EXAMPLE: Profile smiteche
    - Add Next Of Kin, MESSAGE FORMAT: *kin title(Or firstname) surname phone homeLocation*. NOTE: homeLocation(maximum 3 words) EXAMPLE: kin Mr person 07565560000001 king williams town
    - View trip details using trip Id, MESSAGE FORMAT: *Infotrip TripId* NOTE: view a summary of trip details. EXAMPLE: Infotrip 106
    - View trip details using trip code, MESSAGE FORMAT: *rideinfo Tripcode*. NOTE: Trip information relevant to ride share partners i.e. driver and accepted hitchhikers. EXAMPLE: Rideinfo KWT575962
    - Find hitch-hiking points in location, MESSAGE FORMAT: *hikepoints location*. NOTE: location (maximum 3 words OR use a Short code). EXAMPLE: hikepoints grahamstown
    - Find short code of a location, MESSAGE FORMAT: *Code location*. NOTE: Use short code OR location maximum 3 words! EXAMPLE: Code port elizabeth

2. DRIVER OPERATIONS

- Add vehicle details to your profile, MESSAGE FORMAT: *veh regNo make colour bodytype seats*. NOTE: Add vehicle details to your profile, bodytype saloon (s) or bakkie (b). EXAMPLE: veh test123 toyota white s 5
- Create a trip, MESSAGE FORMAT: *Trip VehRegNo destination origin street landmark*. NOTE: destination and origin use 1 word or code of the location. A secret trip code is generated that would be shared with potential ride partners for their identification at the meeting point. EXAMPLE: trip test123 pe gt beaufort shoprite
- Trip offer, MESSAGE FORMAT: *Offer seats cost minutes extra-conditions*. NOTE: minutes should be an integer value e.g. 20 = approx. 20 minutes to pickup time, extra conditions is optional if needed use maximum 5 words. The matcher returns a list of hitch-hikers limited by seats offered. First priority is to hitch-hikers with exact street and nearest landmark followed by those at other landmarks along the driver specified street!) EXAMPLE: Offer 4 80 20 OR Offer 4 80 20 no big luggage
- Add a single hitch-hiker in a trip, MESSAGE FORMAT: *Add hitchhikerId*. NOTE: Add a hitch-hiker from the match list using his/her Id. EXAMPLE: Add 91
- Add all matched hitch-hikers in a trip, MESSAGE FORMAT: *Addall hikers*. EXAMPLE: Addall hikers
- Cancel ride offer, MESSAGE FORMAT: *Canceloffer username password*. NOTE: Cancel your current ride offer. EXAMPLE: Cancel smiteche pass123
- Find Hitch-hikers available in a location looking for ride offers to a destination, MESSAGE FORMAT: *Hitchhikers origin destination*. NOTE: Use short code OR location name with 1 word as long as it is unique! EXAMPLE: Hitchhikers gt pe

3. HITCH-HIKER OPERATIONS

- Post hitch-hiker request, MESSAGE FORMAT: *hike destination origin street landmark seats*. NOTE: seats is optional, must be an integer representing a request for extra seats. EXAMPLE: hike pe gt beaufort kfc
- Cancel hitch-hiker ride request, MESSAGE FORMAT: *Cancelhike username password* EXAMPLE: Cancelhike smiteche pass123

- Find ride offers available from a location to another, MESSAGE FORMAT: *getoffers origin destination*. NOTE: Use short code OR location name with 1 word as long as it is unique! EXAMPLE: getoffers gt kwt

Appendix E

Additional System Test Results

E.1 Functional Tests

Figure E.1 shows the main menus for the driver and hitch-hiker roles using the web application.

Figure E.1: Selecting Roles in the WAP Application

Rideshare trip information can be viewed by registered users with details as shown below in Figure E.2.

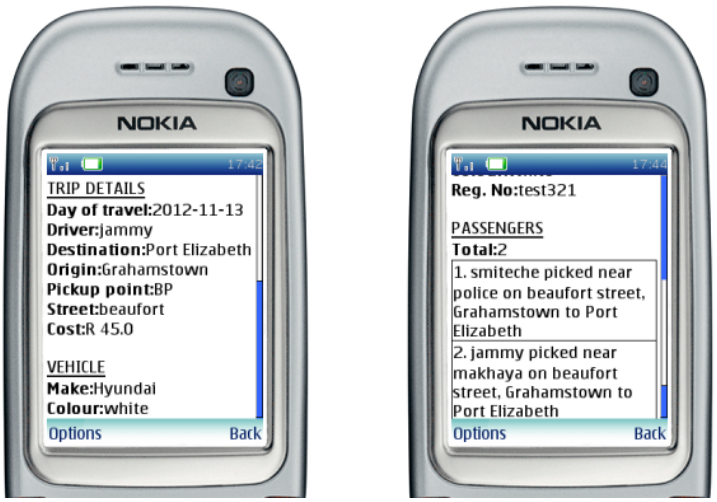

Figure E.2: View Trip Details

Information about known hitch-hiking spots in a particular location can be searched and is provided as shown in Figure E.3.

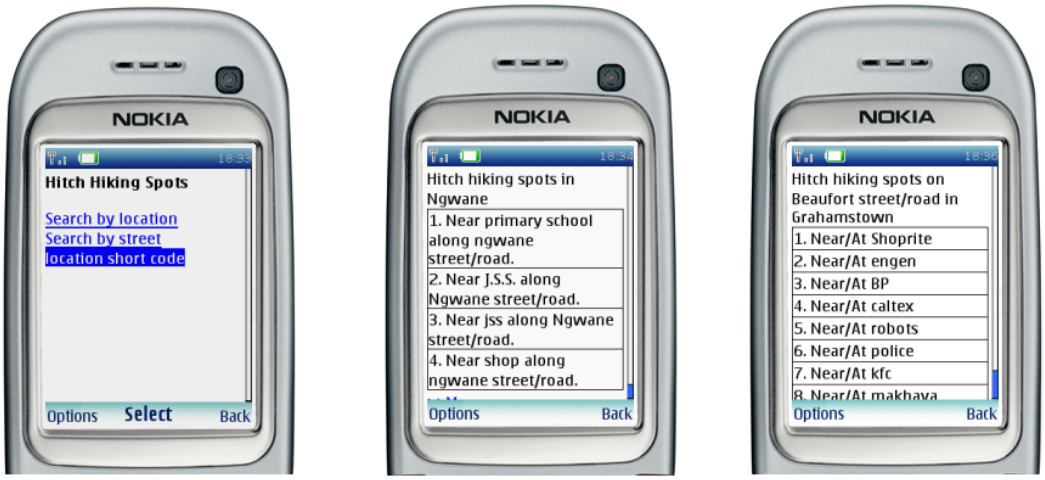

Figure E.3: View Hitch-Hiking Spot Details

Appendix F

The Siyakhula Living Lab

The Siyakhula Living Lab (SLL) is organised along the lines of the emerging Research Development and Innovation (RDI) process living lab methodology [51, 52]. Its field test site is in the Mbashe municipality in the vicinity of the Dwesa-Cwebe Nature Reserve in the rural Eastern Cape Province. Rhodes and Fort Hare Universities have been active in ICT4D in this project, through the two Telkom Centres of Excellence in Telecommunication hosted in their Computer Science departments [52].

The original objective of the SLL was to develop and field-test the prototype of a simple, cost-effective and robust e-business/telecommunication platform, to deploy in marginalized and semi-marginalized communities where a large number (over 40%) of the South African population live [51, 52]. The project has evolved to include a generic service integration platform to support services for rural and peri-urban areas in South Africa.

Current study areas [51] in the SLL are:

1. Broadband telecommunications network models for rural and peri-urban communities.
2. eService provisioning for rural and peri-urban communities.
3. Financial, technical and cultural models for rural and peri-urban ICT initiatives.
4. Monitoring and evaluation of rural and peri-urban ICT initiatives.
5. Rural and peri-urban user requirement elicitation.
6. ICT in education.

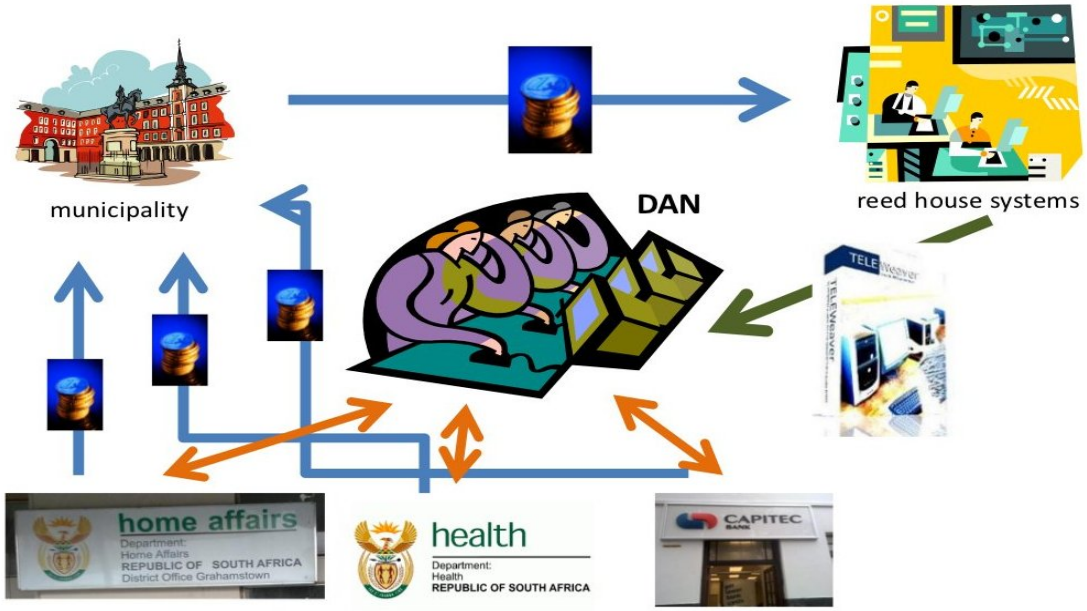

Figure F.1: TeleWeaver Business Model. Taken from [51]

F.1 ReedHouse Systems

The ReedHouse Systems (RHS) [98] is a software company that transforms the experimentation in the SLL into robust industrial products. It started its operations in 2010 and is used by researchers from Rhodes and Fort Hare Universities for their innovations that focus on community-oriented, integrated eServices solutions [98]. The two universities each have the Telekom Centre of Excellence (CoE) [51] whose main focus is the introduction of meaningful ICTs in marginalized areas.

The combination of the two CoEs, the SLL and RHS shows in a practical way how marginalized areas that are difficult to reach, may in future be joined with other communities to the economic, social and cultural benefit of all [51, 52].

ReedHouse System has a service integration platform called TeleWeaver [98]. TeleWeaver is an open source technology platform, based on OSGi (Open Services Gateway Initiative) [5], that comes with service adapters for the South African eGovernment, rural eCommerce, important workflows; and also blueprints for external services through open web services interfaces [52, 98].

The TeleWeaver business model has the capacity to provide the appropriate way of introducing ICT infrastructure in marginalized areas. Figure F.1 shows the TeleWeaver business model with example entities.

Appendix G

Accompanying CD-ROM

The accompanying CD-ROM contains the following:

SachaMiteche.pdf- This thesis in pdf format.

Instructions.txt- How to start the DRS in TeleWeaver.

/References/Documents- Electronic copies of most of the reference material cited in this thesis.

/References/ref.bib– Bibtex database for references.

/SourceCode/Projects- The source code of the DRS: API, Implementation, Restiful web service and WAP bundle projects.

/SourceCode/TeleWeaver- The directory folder for TeleWeaver container with DRS bundles.

/SourceCode/rideshare.sql- The MySQL database dump file of the DRS.

/SourceCode/Kannel.conf- The configurations file for Kannel (bearerbox, wapbox, smsbox and sms application settings).

/SourceCode/RideshareWebservice- Settings of the RESTful Webservice for Kannel message requests.