Title
Using Geographic Location for Optimal Service Selection
Statement of Authorship

I certify that this bachelor's thesis has been composed by myself, and describes my own work, unless otherwise mentioned in the text. Information directly or indirectly taken from other sources has been noted as such. This scientific work will be submitted at two universities, the University of Applied Science in Karlsruhe, Germany and the Linnaeus University in Växjö, Sweden.

Eschbronn, January 10, 2016

[Signature]
Abstract

Nowadays, a multitude of functionally equal web services are available. By this broad offer, the need of a service recommendation based on non-functional characteristics (e.g. price, response time, availability) is increasing. The static Service Level Agreements (SLAs) of service providers cannot suffice this need. SLAs are not reliable enough, due to the fact that they do not cover the dynamic performance and quality changes of services during their lifetime. This bachelor’s thesis was written within a research project of the Linnaeus University in Sweden and the Karlsruhe University of Applied Science in Germany. The goal of this research project is to eliminate the issues as described above. For this reason, a framework for an optimized service selection was developed. Instead of using the static SLAs, measurements of each service call are taken. On the basis of the measurements and the requirements of the consumer, the framework then provides an automated best-fit service selection.

The purpose of this thesis is to involve the geographic location of each service consumer in the automated service selection. Therefore, a mobile app was developed to get a sufficient amount of real world test data. This app measures service calls and additionally records the geographic location of the user. Based on the geographic location, the collected measurement data then were grouped into regions. Thereby, it could be shown that the geographic location of the user can be used to improve the optimal service selection.

Keywords:
Optimal Service Selection, Service Recommendation, Service Level Achievements, Geographic Location
## Contents

1 Introduction 1  
   1.1 Background Motivation 1  
   1.2 Problem Definition 3  
   1.3 Purpose and Research Question 4  
   1.4 Scope / Limitation 4  
   1.5 Outline 4  

2 Theory 5  
   2.1 RESTful Web Services 5  
   2.2 Other Research in the Field 7  
   2.3 The Median 8  

3 Method 9  
   3.1 Scientific Approach 9  
   3.2 Software Development 9  
      3.2.1 Monitored Web Services 9  
      3.2.2 Backend 10  
      3.2.3 Android App 11  
   3.3 Analysis 12  
   3.4 Ensuring Reliability and Validity in the Study 13  
   3.5 Ethical Considerations 14  

4 Implementation 15  
   4.1 Backend 15  
      4.1.1 Google App Engine 15  
      4.1.2 Database 16  
      4.1.3 RESTful Web Interface 18  
   4.2 Android App 19  
      4.2.1 Background Service 19  
      4.2.2 Scheduling and User Interface 20  
      4.2.3 Publishing 21  
   4.3 Geographical Region Creation 21  

5 Results 22  
   5.1 Raw Data 22  
      5.1.1 Data Amount and Locations 22  
      5.1.2 Measurements per Phone 24  
      5.1.3 Measurements per Service 25  
   5.2 Region Data 26  
      5.2.1 Measurements per Region and Service 27  

6 Analysis 29  

7 Discussion 30  
   7.1 Results 30  
   7.2 Method Reflection 31
8 Conclusion

8.1 Further Research

A Appendix 1

A.A Appendix 1.1 - Median Response Times per Phone

A.B Appendix 1.2 - Average Response Times per Phone
List of Tables

2.1 CRUD - HTTP protocol [1] .................................................. 6
4.2 Service.java .................................................................. 16
4.3 ServiceInstance.java ...................................................... 16
4.4 Phone.java .................................................................. 16
4.5 FeedbackTicket.java ...................................................... 17
5.6 Measurements of each phone ........................................ 24
5.7 Measurements per Region ............................................. 26

List of Figures

1.1 Extended Service Binding [2] ........................................... 2
1.2 Adding a new service consumer to the framework .......... 3
3.3 Measurement feedback format ...................................... 10
3.4 Database scheme .......................................................... 11
3.5 Android fragmentation [3] .............................................. 13
4.6 Android App ............................................................... 20
5.7 Measurements world map ............................................ 22
5.8 Median response times per service ................................. 25
5.9 Average response times per service ............................... 25
5.10 Median response times per region and service ............ 27
5.11 Average response times per region and service .......... 28
1.12 Median response times per phone ............................... A
1.13 Average response times per phone .............................. B
List of Abbreviations

SOA  Service Oriented Architecture ............................................. 1
SLA  Service Level Agreement .................................................. 1
ESB  Enterprise Service Bus ..................................................... 2
UI   User Interface .................................................................... 4
REST Representational State Transfer ........................................... 5
HTTP Hypertext Transfer Protocol ............................................. 5
IP   Internet Protocol ............................................................... 7
CRUD Create, Retrieve, Update, and Delete ..................................... 6
URI  Uniform Resource Identifier .............................................. 6
JSON JavaScript Object Notation ................................................ 6
XML  Extensible Markup Language ............................................... 6
PCC  Pearson Correlation Coefficient .......................................... 7
UTC  Coordinated Universal Time ................................................ 10
API  Application Programming Interface ...................................... 13
GPS  Global Positioning System .................................................. 14
PaaS Platform as a Service ....................................................... 15
IDE  Integrated Development Environment ................................... 15
SQL  Structured Query Language ............................................... 15
SDK  Software Development Kit ................................................ 15
JPA  Java Persistence API ........................................................ 16
POJO Plain Old Java Object ....................................................... 16
URL  Uniform Resource Locator ................................................ 18
JAX-RS Java API for RESTful Web Services .................................. 18
1 Introduction

This first chapter will give an introduction to the area of this thesis.

Firstly, the background of this work is described and the motivation is pointed. Secondly, the problem of this work is discussed. Thirdly, the purpose and research question of this thesis is presented. Fourthly, the limitations are pointed out. Finally, a brief overview of the other chapters is given.

1.1 Background Motivation

This thesis is part of a research project between the Linnaeus University in Växjö, Sweden and the Karlsruhe University of Applied Science, Germany. The research takes place in the field of Service Oriented Architecture (SOA). The basic idea behind the research project is that there is an open market of web services, in which several services provide the same functionality. In such a market, it is getting more and more challenging to choose the best fitting service based on functional and quality requirements. Services, which provide the same functionality, only distinguish themselves in their non-functional characteristics (e.g. response time, availability and price). While service providers describe these characteristics in their Service Level Agreements (SLAs), Kirchner et al. state in their research that the selection of a service based only on the provided information, is inadequate and may lead to disadvantages against competitors [2]. A service may, for example, show different response times on weekdays than on weekends. Additionally a high consumed service may get less reliable because of its high workload. These quality characteristics change over time and are not reflected by static SLAs.

Kirchner et al. describe an approach to solve these problems by monitoring services and use the gained knowledge for an optimized service selection [2]. Therefore the classic service binding process is extended to measure every service call and use the information to trace the quality attributes of a service over time. As a result, the dynamic behaviour of services is considered in the service selection. To solve this, the research group developed a framework which changes the local service binding and provides a central service broker for the service recommendation process.
As shown in figure 1.1 the concept is based on a logically centralized service broker which is aware of service consumers and providers. Firstly, the service provider has to publish itself to the service broker (step 1). Then a service consumer can ask for a provider by providing the service context and an utility function (step 2). The utility function includes weighting for the non-functional attributes (e.g. performance and price). Based on the context and utility function, the service broker is able to recommend the best-fitting service provider (step 3). After the service binding, the service consumer is able to send requests (step 4 and 5). Simultaneously, the performance of the service-calls is measured and reported to the service broker (step 6). Finally, the measurements are used to improve the automated service selection (step 7).

Since this fundamentally changed service binding would cause elementary changes in existing service consumers, a local integration component was developed. This component was developed as an Enterprise Service Bus (ESB) extension and is located in the local system environment of the consumer. It acts as an agent between the service consumers and providers. Service consumers in the local environment have to be registered at the local component, therefore a web interface is provided. After the registration, local services only communicate with the local integration component. Furthermore, the integration component sends the measurement result of every single service call to the central component. The central component is able to notify the local component if it identified a better-fitting service provider for a service consumer. Then the local component will change the service binding. This approach allows to benefit from a best-fit service selection without the need to change existing applications.
Indeed, the approach of Kirchner et al. provides a solution for the inadequate service selection based on SLAs. By monitoring the services, reliable real world data can be used to provide an independent service recommendation. While this approach works fine for service consumers with existing measurement results, adding a new consumer reveals a lack of knowledge. This problem is briefly described in the following chapter.

1.2 Problem Definition

Figure 1.2 illustrates the problem with the existing approach. The service broker recommendation for Service Consumer A is the Service Provider A, whereas Service Provider B is recommended for Service Consumer B. Both providers are functionally equal, but Service Provider A may have a shorter response time for Service Consumer A than for Consumer B. However, the service broker does not have measurements for the newly added Service Consumer C. Which of both providers shall now be recommended to the new consumer?

<table>
<thead>
<tr>
<th>ID</th>
<th>Service Consumer</th>
<th>Recommended Service Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Service Consumer A</td>
<td>Service Provider A</td>
</tr>
<tr>
<td>2</td>
<td>Service Consumer B</td>
<td>Service Provider B</td>
</tr>
<tr>
<td>3</td>
<td>Service Consumer C</td>
<td>?</td>
</tr>
</tbody>
</table>

Figure 1.2: Adding a new service consumer to the framework

Solving this issue is the aim of this thesis. This will be done by adding geographic location of the consumer to improve the optimal service selection and fill the gap in knowledge if a new consumer asks for the best-fit service.
1.3 Purpose and Research Question

The purpose of this thesis is to identify if the geographic location information of a user can be used to improve the automated best-fit service selection. Therefore, a mobile app shall be developed to collect real world test data of service providers including the geographic location of the user. This data shall then be used to fill the gap in knowledge and identify possibilities of using the users’ location. This leads to the following research question:

*How can geographic location information be used for the optimal service selection?*

1.4 Scope / Limitation

Several limitations within this research project need to be taken into consideration from the beginning. Firstly, since this research is within the limits of a bachelor’s thesis, the time frame for the elaboration was limited to three months. Secondly, due to the fact that the existing framework of the research project is a complex software, no production ready extension for involving the geographic location in the best-fit service selection will be developed. However, an app for mobile devices will be developed to collect geographical location enriched test data. This app will focus on the scope of the research project and will not provide possibilities to specify the web services used for generating test data through the User Interface (UI), or other comfort options.

Furthermore, the analysis will be done on a local system and (as mentioned above) will not lead into a production ready implementation.

1.5 Outline

This paper is subdivided into eight chapters. In the next chapter, the used concepts, theories and a similar research project are described. It will then go on to the method chapter, where the used scientific approach is stated. Additionally, the specifications for the software development and how reliability and validity is ensured in this study are pictured. The fourth chapter shows in detail how the software was implemented and the presented concepts and theories used. Afterwards, the collected data are presented in the results chapter. In the sixth chapter, the results are analysed. In chapter seven, the knowledge contributions reached through this research are discussed. Finally, a conclusion of this research and ideas for further research in addition to this bachelor’s thesis is given in the last chapter.
2 Theory

This chapter gives an overview of the used theories and techniques in this work.

To understand the theoretical background we begin by reviewing the Representational State Transfer (REST) architectural style, used for the developed backend in this thesis. It will then go on to present research done within the field of location based web service recommendation. Additionally, the definition of the median, used to compare the results, is given. Finally, the basic machine learning concepts are discussed.

2.1 RESTful Web Services

The concept of RESTful web services will be used to develop the web interface of the backend in this work. The REST architectural style was introduced by Roy T. Fielding in his PhD dissertation, published in 2000 [4]. Fielding defines the following constraints, which form a RESTful system:

Client-Server
Separates the client from the server and enables portability and scalability.

Stateless
All requests are independent from each other and no user sessions are kept on the server.

Cache
Responses to a request have to "be implicitly or explicitly labeled as cacheable or non-cacheable." [4].

Uniform Interface
Standardized information exchange, instead of application specific formats.

Layered System
The system is "composed of hierarchical layers", each layer encapsulates a component [4].

Code-On-Demand
Optional constraint, "allows client functionality to be extended by downloading and executing code" [4].

Indeed, this description only defines how data are exchanged. Consequently, a RESTful system can be created with every available technology. Nevertheless, only Java (as programming language) and Hypertext Transfer Protocol (HTTP) (as network protocol) will be used in this work.
The abstractions, which make a system RESTful, are "resources, representations, URIs, and the HTTP request types that make up the uniform interface used for client/server data transfers." [1]. The descriptions of these abstractions are given below.

**Resource**
Information that can be named and transferred. For example, an image, the current weather in Växjö or the geographic location of a user [1].

**Representation**
Describes the resource during its transmission over the network. An image, for example, could be transmitted as an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) stream [1].

**URI**
A Uniform Resource Identifier (URI) "in a RESTful web service is a hyperlink to a resource" [1]. However, a URI being a hyperlink, applies only in the context of web services and not to REST in general.

**HTTP requests**
In this work, the resource representations are transmitted through HTTP. Data modifications in modern web applications are generally described with Create, Retrieve, Update, and Delete (CRUD) actions [1]. The CRUD actions are represented within the HTTP protocol as follows:

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP protocol equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>POST</td>
</tr>
<tr>
<td>RETRIEVE</td>
<td>GET</td>
</tr>
<tr>
<td>UPDATE</td>
<td>PUT</td>
</tr>
<tr>
<td>DELETE</td>
<td>DELETE</td>
</tr>
</tbody>
</table>

Table 2.1: CRUD - HTTP protocol [1]

Understanding the described concepts, undoubtedly, is necessary to be able to design and implement a RESTful web service. The abstractions outlined above will be used to design and implement the RESTful web interface of the backend as described in chapter 4.
2.2 Other Research in the Field

In their article, "Personalized QoS-Aware Web Service Recommendation and Visualization", Chen et al. describe an approach to involve the geographic location of a user in the automated web service recommendation [5]. The researchers recognized that some non-functional characteristics, "like response time and availability highly relate to the users' physical locations." [5]. As a result, they designed a recommendation approach which consists of two phases. First, based on historical measurements and the geographic location, users are divided into regions. Second, unused services are predicted and new users are assigned to regions with similar non-functional characteristics and location.

The region creation process is subdivided into three steps. In the beginning, the users are grouped by their Internet Protocol (IP) address into small regions. Additionally, "region features" are extracted. After the different regions have been created, "the similarity between" them is calculated. Finally, "highly correlated regions" are grouped to large regions. Under the term "region features", the authors refer to a so called "region center as the main feature to reflect the average performance". This region center contains the median response time for each measured web service of the users within that specific region. Moreover, services with extremely high response times and the related region are extracted.

Before similar regions can be grouped, the similarities have to be calculated. This is done by comparing the region centers. Therefore, the median response times of each shared service and the average response time of all services within a region are used to calculate the Pearson Correlation Coefficient (PCC) [5].

In the next step, the calculated similarities are then used to group the regions to bigger ones. However, a region identified to be sensitive, due to extremely high response times, will not be grouped with others.

As a result, users are now clustered regarding their measurement results and geographic location. Consequently, the service recommendation for existing and new users is then based on these regions and considers the geographic location besides the non-functional quality attributes.

The presented approach will also be used in this thesis to make the web service recommendation location aware. Instead of extracting the location out of the IP address, the more accurate geographic coordinates provided by a mobile device are used in this thesis to determine the location of the user.
2.3 The Median

In statistics, the median is defined as the "point separating ordered sequence of data into two parts such that half of the data is less than the median and half of the data greater than the median." [6]. Within an odd number of data, it can be found by ordering all numbers in ascending order and selecting the middle one. For an even number of data, the median can be found by ordering all numbers in ascending order, sum the two middle values and divide by 2 [7]. The median is more robust than the mean (also known as average) and therefore will be used in this thesis to compare the results [7].
3 Method

This chapter defines and explains the methodological concepts used throughout this thesis project.

Firstly, the scientific approach used in this research is presented. Secondly, the methods used to obtain the results are stated. Thirdly, the concept to validate the results is described. Additionally, it is explained how reliability and validity is ensured throughout this work. Finally, ethical considerations occurring in this research are discussed.

3.1 Scientific Approach

While using the deductive approach means to have a theory from the beginning and validate it through the research process, inductive research starts with the observation and then leads to a theory [9]. Indeed, the inductive approach is more investigative, whereas the deductive approach is more constricted [9].

As stated in the introduction, this thesis is based on the hypothesis that using the geographic location improves the automated service selection process. Furthermore, the goal of this work is, to find ways how the geographic location can be used in the selection process. Therefore, action research will be done were data will be collected within a real world scenario and later be analysed. Consequently, deductive reasoning is used to answer the research question of this research.

3.2 Software Development

To answer the research question defined in this work, data including the geographic location of service consumers are needed. Therefore, the author will develop a mobile app to collect the required data. Due to the reason that the author has already gained experience with Android app development throughout his studies, the app will be developed for this platform. The app will measure the response times of web service calls and trace the geographic location of the phone. The measurement results are then transmitted to a backend were they are saved for the analysis. This backend will also be developed by the author.

3.2.1 Monitored Web Services

Significant data of real world web services are required for a quantitative analysis of the research question. Due to the fact that this research is focusing on the impact of the geographic location, at least two functionally equal web services with different locations are needed. As the previous research of Kirchner et al., this work also focuses on a stock exchange scenario and the same web services will be used for the measurements [2], namely, Yahoo Finance[1], Lang & Schwarz provided by Ad-hoc News[^2] and WebserviceX[^3]. While Ad-hoc News is hosted in Bavaria, Germany, WebserviceX is hosted in Arizona, United States of America. Yahoo Finance is also hosted in the United States [10].

[^1]: http://finance.yahoo.com
[^2]: http://www.ad-hoc-news.de
[^3]: http://www.webservicex.com
3.2.2 Backend

The backend will be responsible for receiving measurement data from the app and save it in a database. The author decided to develop it based on the Google App Engine\(^4\). This decision is based on the authors previous experience with this product. Using the Google App Engine enables the author to focus on the development without the need to administrate a server. The backend will be developed in Java. Firstly, because the author has several years of programming experience with this language. Secondly, because it is fully supported by Google. For receiving the measurement data, a REST Interface has to be designed \([1]\). As stated before, the app will send the device’s current geographic location and the measurement results for each of the three web services. This leads to the measurement feedback message showed in figure 3.3 below. This message will be generated by the app and has to be understood by the backend.

<table>
<thead>
<tr>
<th>Measurement Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>androidId: String</td>
</tr>
<tr>
<td>gpoPosition: String</td>
</tr>
<tr>
<td>latitde: double</td>
</tr>
<tr>
<td>longitude: double</td>
</tr>
<tr>
<td>serviceIdentifier: String</td>
</tr>
<tr>
<td>date: long</td>
</tr>
<tr>
<td>timeZone: String</td>
</tr>
<tr>
<td>responseTime: long</td>
</tr>
<tr>
<td>availability: boolean</td>
</tr>
<tr>
<td>serviceIdentifier: String</td>
</tr>
<tr>
<td>date: long</td>
</tr>
<tr>
<td>timeZone: String</td>
</tr>
<tr>
<td>responseTime: long</td>
</tr>
<tr>
<td>availability: boolean</td>
</tr>
<tr>
<td>serviceIdentifier: String</td>
</tr>
<tr>
<td>date: long</td>
</tr>
<tr>
<td>timeZone: String</td>
</tr>
<tr>
<td>responseTime: long</td>
</tr>
<tr>
<td>availability: boolean</td>
</tr>
</tbody>
</table>

Figure 3.3: Measurement feedback format

Firstly, the androidId is stated. It "is randomly generated when the user first sets up the device and should remain constant for the lifetime of the user’s device" \([11]\). Being able to assign every measurement to a device is important for the analysis of the results. That is why this identifier is transmitted. Secondly, the geographic location object is stated with the values latitude and longitude. Lastly, the three measurement objects are illustrated. Each of them contains the serviceIdentifier, used to name which web service was used. Possible options are "Google", "Yahoo" and "LangSchwarz". To be able to compare all measurements from different geographic locations, date contains the time in the Coordinated Universal Time (UTC) format. The performance of a web service, however, can be affected by more or less network traffic regarding to the local time of day. This is why the timeZoneId of the user’s device is also transmitted. The attribute responseTime contains the response time of the request in milliseconds. The last attribute availability states if the request was successful or not (successful means: HTTP response code 200) \([12]\). Based on this message format the app will be able to send all the results, and the backend will be able to store the required data in a database.

\(^4\)https://cloud.google.com/appengine
The Google App Engine also offers a relational database service called "Cloud SQL" [13]. This service is based on the open source database MySQL\(^5\) and will be used to store the measurement results. The following database scheme will be used.

![Database scheme](image)

**Figure 3.4: Database scheme**

### 3.2.3 Android App

As already stated, an Android app will be developed. The app should call the three stated web services outline in chapter 3.2.1 in a 30 minutes interval. This action has to be ensured, even in case the user does not open the app. Thereby, it requests the SAP\(^6\) stock and transmit the measurement data to the backend as presented in figure 3.3. Since this research is about the geographic location, the current position of the phone has to be traced too. Furthermore, it should present what it is actually doing to the user, and presenting the time of the last interval to give the user feedback of the background work. Obviously, the app should be installable on multiple device types to find as many test devices as possible.

\(^5\)https://www.mysql.com

\(^6\)http://www.sap.com
3.3 Analysis

Initially, the existing optimal service selection algorithm should have been modified to consider the geographic location of the user. Thereby, it would have been possible to first train the existing algorithm with the collected data, and in the second step, also train the modified algorithm with the same data. The results could have been analysed based on the quality attributes availability and response time. Thus, the recommendations for the users could have been compared, and in case of a positive effect, some users should have gotten a web service recommendation with better quality attributes than before. Additionally it could have analysed if the modification had a negative performance influence to the optimal service selection.

Due to some problems with the existing software, this approach could not be applied. Consequently, the author changed the analysis and used Microsoft Excel to interpret and analyse the collected data. Therefore, the median response time was used as the main quality attribute. The median response time was calculated out of all collected data, once for each monitored web service. This represents the approach of the existing algorithm to not consider the geographic location of the user. Additionally, the collected data has been manually grouped into regions and the median response time was calculated for each web service within each region. This represents the approach described in this work. The median response times was then manually analysed and the characteristics was described.
3.4 Ensuring Reliability and Validity in the Study

The reliability of the developed software components in this research is ensured by manually testing the backend software and Android app before using it for this study. To ensure the reliability of the backend, different clients were used to transmit measurement feedback data. The author used Google Chrome\textsuperscript{7}, Mozilla Firefox\textsuperscript{8} and the parallel developed Android App as clients to transmit data to the backend. Besides sending proper data and making sure the backend software processes and saves it correctly, a variety of incorrect data was also sent to verify the refusal.

Verifying the reliability of an Android app is a quite complex task. On one hand, this is caused by the highly fragmentation of the Android operating system. According to the latest Google statistic from August 2015, only 2.6\% of all devices are running the latest Google Play Store App, are operating under the current Android 5.1 (Lollipop). 15.5\% of all devices are running the previous Android 5.0 (also Lollipop). The majority of devices are running Android 4.4 (39.3\%) and even version 2.3.x is still installed on 4.6\% of all devices [3]. Figure 3.5 illustrates the fragmentation of the Android system.

![Figure 3.5: Android fragmentation [3]](image)

On the other hand, more than one thousand manufacturers are producing ten thousands of phone models with different hardware. Corresponding to OpenSignal, a company specialized in collecting wireless network data, more than 24,000 distinct devices had installed their app in August 2015 [14]. As discussed before, knowing the geographic location is crucial for this study and was the main focus for the reliability of the app. Therefore, the app will only measure service calls if the latest version of the Google Play Services app is installed, as it provides the location Application Programming Interface (API). In the case, that the latest version is not installed or outdated, the user will get a hint to update it and no measurements will be performed until it is installed. If the user turned off the location services in the Android settings, a hint will show up to turn them on. Furthermore, before published the app was manually tested on a Google Nexus 4, a Google Nexus 5 (both with Android 5.1.1 installed) and on a Motorola Razr i (Android 4.4), all with different geographic locations.

\textsuperscript{7}https://www.google.com/chrome/browser
\textsuperscript{8}https://www.mozilla.org/en-US/firefox
The validity of this research mainly depends on the validity of the collected data. Due to the fact that this study is operating in a stock exchange scenario like the research project of Kirchner et al., it was limited to three public stock exchange services [2]. In addition, the author was not able to distribute the measuring app all over the world. Data from several countries in Europe, North and South America, and Southeast Asia could be collected. Furthermore, the author had a time constraint which limited the time to collect data to three weeks.

3.5 Ethical Considerations

Since this research is based on collecting measuring data with personal phones, and the geographic location of the user is traced, several considerations have to be preconceived. Firstly, the app is not collecting any personal data besides the Android identifier and the geographical location. Initially collected, to assign all measuring data to the corresponding phone, the Android identifier was anonymized after the end of the data collection. Thereby, it is ensured that all measurements are still assigned to a device without being able to identify a single person later on. Secondly, instead of using the quite accurate Global Positioning System (GPS) for tracing the location, "network location sources such as cell towers and Wi-Fi" were used [15]. These resources provide the "location with an accuracy approximately equivalent to a city block" [15]. This is precise enough for this research and has the lowest possible impact on the users’ privacy.
4 Implementation

This chapter describes how the backend and the Android app have been implemented in detail.

This chapter starts with the description of the backend including database and RESTful web interface. Furthermore, the app implementation for the Android operating system is discussed. Finally, the approach to use the geographic location for the optimal service selection is described.

4.1 Backend

As written in chapter 3.2.2, the server is implemented as a Google App Engine\(^9\) software in Java. Because of several years of experience, the author used Eclipse\(^10\) as Integrated Development Environment (IDE). In addition, the build tool Apache Maven\(^11\) is used. One advantage of Maven is to enable the developer to specify all required libraries in a single XML file. Maven then downloads the required libraries and manages the build process [16]. Undoubtedly, this simplifies the build process a lot for software projects with third party libraries and dependencies.

4.1.1 Google App Engine

The Google App Engine is a so-called Platform as a Service (PaaS), which is defined as "the capability provided to the consumer [...] to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider" [17]. Thereby "the consumer does not manage or control the underlying cloud infrastructure [...], but has control over the deployed applications and possibly configuration settings for the application-hosting environment" [17].

In this work, the Google App Engine runtime environment is used to run the backend for receiving and storing the measurement feedbacks. Additionally, the provided Structured Query Language (SQL) database is used to store the received data. To make the Google App Engine development easier, a Maven artifact is provided by Google [16]. This artifact simplifies the creation of a Maven project including the Google App Engine Software Development Kit (SDK) dependencies. As a result, all required dependencies for the Java development are available on the development machine and the project can be run locally during development.

---

\(^9\)https://cloud.google.com/appengine/
\(^10\)http://www.eclipse.org/
\(^11\)https://maven.apache.org/
4.1.2 Database

For an easy access to the database the Java Persistence API (JPA) reference implementation EclipseLink\(^2\) is used. With the help of this framework, the database as described in chapter 3.2.2 is represented as Plain Old Java Objects (POJOs) in the source code. According to the database design, the following classes are representing the database scheme:

- Service
- ServiceInstance
- Phone
- FeedbackTicket

Every class represents a database table and every attribute of a class stands for a column. As seen in table 4.2, the Service object has an identifier and a functional service name. Due to the fact that this study focuses on a stock exchange scenario, the only functional service name is Stock.

<table>
<thead>
<tr>
<th>Service</th>
<th>Attribute</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>serviceId</td>
<td>Long</td>
</tr>
<tr>
<td></td>
<td>functionalServiceName</td>
<td>String</td>
</tr>
</tbody>
</table>

Table 4.2: Service.java

As seen in table 4.3 the ServiceInstance object represents the three web services used for the measurements. Therefore, it also has an identifier of the service instance, a reference to the Service object and the name of the web service.

<table>
<thead>
<tr>
<th>ServiceInstance</th>
<th>Attribute</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>serviceInstanceId</td>
<td>Long</td>
</tr>
<tr>
<td></td>
<td>service</td>
<td>Service</td>
</tr>
<tr>
<td></td>
<td>serviceInstanceName</td>
<td>String</td>
</tr>
</tbody>
</table>

Table 4.3: ServiceInstance.java

The Phone object simply stores the Android identifier as a String. The Android identifier is the primary key and must be distinct (see Table 4.4).

<table>
<thead>
<tr>
<th>Phone</th>
<th>Attribute</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>androidId</td>
<td>String</td>
</tr>
</tbody>
</table>

Table 4.4: Phone.java

\(^2\)http://www.eclipse.org/eclipselink
Table 4.5 illustrates the *FeedbackTicket* class with all its attributes. Every feedback ticket has a reference to the corresponding phone and service instance. Additionally to the date attribute, the time zone, weekday and daytime are saved. The geographic location is represented by the latitude and longitude attributes as double values. Finally, the response time in milliseconds and the availability of the service are saved.

<table>
<thead>
<tr>
<th>FeedbackTicket Attribute</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>feedbackTicketId</td>
<td>long</td>
</tr>
<tr>
<td>phone</td>
<td>Phone</td>
</tr>
<tr>
<td>serviceInstance</td>
<td>serviceInstance</td>
</tr>
<tr>
<td>date</td>
<td>Date</td>
</tr>
<tr>
<td>timeZoneId</td>
<td>String</td>
</tr>
<tr>
<td>weekday</td>
<td>String</td>
</tr>
<tr>
<td>daytime</td>
<td>int</td>
</tr>
<tr>
<td>latitude</td>
<td>double</td>
</tr>
<tr>
<td>longitude</td>
<td>double</td>
</tr>
<tr>
<td>responseTime</td>
<td>long</td>
</tr>
<tr>
<td>availability</td>
<td>boolean</td>
</tr>
</tbody>
</table>

Table 4.5: FeedbackTicket.java

With the help of these POJOs and special annotations for the attributes, the JPA is able to generate the corresponding database tables. Annotations "provide data about a program that is not part of the program itself" [18]. In this project the annotations @Id and @GeneratedValue(strategy = GenerationType.IDENTITY) are used to specify the primary key for the database table. @GeneratedValue indicates, that the primary key will be automatically generated based on "a database identity column" [18].

After defining the classes, they have to be tagged as persistence units. This is done in the *persistence.xml* file, located in the *META-INF* directory of the Maven project. How the tagging of the persistence units looks like can be seen in Listing 4.1.

```
< persistence − unit name= "feedbackdb" >
  < class > de . manuelhauch . soc . backend . model . Service </ class >
  < class > de . manuelhauch . soc . backend . model . ServiceInstance </ class >
  < class > de . manuelhauch . soc . backend . model . GpsPosition </ class >
  < class > de . manuelhauch . soc . backend . model . Phone </ class >
  < class > de . manuelhauch . soc . backend . model . FeedbackTicket </ class >
  < properties >
    < property name = "javax.persistence.jdbc.user" value = "root" />
    < property name = "eclipselink.ddl−generation" value = "create−tables" />
  </ properties >
</ persistence−unit >
```

Listing 4.1: Persistence.xml

Firstly, the name *feedbackdb* is stated, it is used to load the settings later on. Secondly, the classes to persist are listed. Finally, the user is given as *root* and the value *create-tables* defines that JPA automatically creates the database tables based on the Java objects.
Since the Google App Engine uses a specific MySQL driver, it must be made sure whether the code runs on the local development system or on the Google system. Therefore the Google App Engine provides an easy way to check if the code is running on the production or on a development system, and then be able to choose the correct driver and Uniform Resource Locater (URL). This pattern is displayed in Listing 4.2.

```java
Map<String, String> properties = new HashMap();
if (SystemProperty.environment.value() == SystemProperty.Environment.Value.Production) {
    properties.put("javax.persistence.jdbc.driver", "com.mysql.jdbc.GoogleDriver");
    properties.put("javax.persistence.jdbc.url", "jdbc:mysql://socbackend:sql/feedbackdb");
} else {
    properties.put("javax.persistence.jdbc.driver", "com.mysql.jdbc.Driver");
    properties.put("javax.persistence.jdbc.url", "jdbc:mysql://127.0.0.1:3306/feedbackdb");
    properties.put("javax.persistence.jdbc.password", "root");
}
emfInstance = Persistence.createEntityManagerFactory("feedbackdb", properties);
```

Listing 4.2: Choose MySQL driver

After implementing the database functionality, the next step in the development process is the web interface.

4.1.3 RESTful Web Interface

The RESTful web interface is based on the Java API for RESTful Web Services (JAX-RS) reference implementation called Jersey\(^{13}\). With its help, the necessary Java method for receiving the measurement results is shown in Listing 4.3.

```java
@POST
@Path("/post")
@Consumes(MediaType.APPLICATION_JSON)
public Response createFeedbackTicket(Feedback feedback) {
    // Save feedback in database
    // ...
}
```

Listing 4.3: POST method

The @POST annotation specifies, that the method responds to HTTP POST requests. Through @Path, the method is made available under http://server/app/post/. Because of the design decision to use JSON messages, this format is specified to be used through @Consumes.

Referring to the abstractions presented in chapter 2.1, the resource in this implementation is the Feedback object holding the measurement results. The JSON stream is the representation of the Feedback object. An URI is given through defining the URL http://server/app/post/. And finally, the HTTP method POST is used.

\(^{13}\text{https://jersey.java.net}\)
4.2 Android App

As defined in chapter 3.2.3, the app was developed for the Android operating system. The development was first started using Eclipse, but the author switched to Android Studio\(^{14}\), the official IDE for Android development. This was caused by several dependency problems that occurred in Eclipse. Migrating the Android project to Android Studio immediately had a positive effect on the development process. Additionally, Google announced the end of support for the Eclipse SDK by the end of the year 2015 [19].

The following section explains how the defined requirements are realized in detail.

4.2.1 Background Service

In chapter 3.2.3 it was defined that the app shall start a measurement process every 30 minutes, without any action of the user required. Since no user interaction is necessary, this task has to run in the background. This use case is covered by the Android system with the concept of services, "an application component that can perform long-running operations in the background and does not provide a user interface" [20]. For this app, when started, the service first checks if the phone has an active network connection to be able to perform the measurements. If so, the measurements are taken and transmitted to the backend server on the Google Cloud. Afterwards, the execution date is set in the UI to comply the requirement to give the user feedback of the performed action. Finally, when all actions are performed, the service stops itself.

An Android service does not run in a separate process and does not create an own thread, it "runs in the main thread of its hosting process" [20]. That is why network operations have to be performed in a separate thread to not block the hosting process, which could lead to killing the app by the Android system. Therefore, Android provides the so-called AsyncTask [21]. The measurement service starts its own AsyncTask to perform the network operations, which are then executed in an own process and do not affect the UI. These network operations cover the measurement of the SAP stock request at Lang & Schwarz, WebserviceX and Yahoo. The measurement of every request is started at the point were the HTTP connection is established until the response with the stock data from the web service is retrieved. The measured response time for each of the three stock services, including the current geographic location of the user, is then sent to the backend server.

4.2.2 Scheduling and User Interface

To start the background service every 30 minutes, the Android AlarmManager is used [22]. The AlarmManager makes it possible to run code at a specific time, no matter if the app is running or not. Listing 4.4 shows how the AlarmManager is used. After getting a reference, a new Intent to the StartServiceReceiver class is created. "An intent is an abstract description of an operation to be performed" and "provides a facility for performing late runtime binding between the code in different applications" [23]. In order to avoid delay because of too many simultaneously requests, the interval was scheduled with ELAPSED_REALTIME_WAKEUP instead of the wall clock time [22]. Hereby, the different phones collecting test data, request the web services all at different times and not simultaneously. Moreover, the time until the first trigger and the interval is defined (INTERVAL_HALF_HOUR).

```
alarmMgr = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(context, StartServiceReceiver.class);
alarmIntent = PendingIntent.getBroadcast(context, 0, intent, 0);
alarmMgr.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                              AlarmManager.INTERVAL_HALF_HOUR,
                              AlarmManager.INTERVAL_HALF_HOUR, alarmIntent);
```

Listing 4.4: Service Interval

The StartServiceReceiver class then requests the geographic location of the device and the androidId for identifying the device. The location is determined with network sources (cell towers and WiFi) which is faster, but still adequately precise. After receiving the location information, the background service is started to perform the measurements.

As written above, the time of the last measurement is presented in the UI. Additionally a button to contact the author via E-Mail (in case of problems) was implemented. After pressing the button, the user is asked which app should be used for sending and the authors E-Mail address and subject is automatically filled out. Figure 4.6 shows the user interface.

![Figure 4.6: Android App](image)

Figure 4.6: Android App
4.2.3 Publishing

Getting as many measurement results as possible was crucial for this study. Consequently, it was important to find people from different countries willing to install the Android app. Therefore, the author published the app in the Google Play Store\textsuperscript{15}, making it as easy as possible to distribute it. After the registration as a developer in the Google Play Store, the app could be uploaded and got checked before publishing for inappropriate content and behaviour by Google. Once published, the app could be installed on every device with the Google Play Store and Android 4.0.3 or newer installed.

4.3 Geographical Region Creation

After collecting the measurement data, the approach described in chapter 2 is used to classify the data. The measurement results are first grouped into regions. In the research of Chen et al., this is done by looking up the IP address of the user and group users with similar addresses [5]. Thereby, every user is a member of exactly one region. However, in this work the geographic locations are captured as latitude and longitude positions. Consequently, the users are grouped by similar geographic locations and not by IP addresses. Due to the fact that phones are used to collect the data, a user can have plenty of different locations and therefore be in multiple regions. That is why, in contrast to the research of Chen et al., in this work regions are grouped by the geographic location of each measurement instead of being grouped by users. As a result, users can collect data for different regions and the measurements will always be assigned to the proper region. The regions in this work are created according to the countries, because the data set is not large enough to create smaller regions.

As in the research of Chen et al., the next step after the region creation is the extraction of the region features. This research focuses on the response time as non-functional feature to describe the quality of a service. As outlined in chapter 2.2 the region center is used to describe the performance of web services within a region. It is defined as the median of all response times collected by the user within a region. Or more precisely, the "element $i$ of the center is the median RTT value of service $i$ observed by users from the region." [5]. Where RTT means round-trip time, also known as response time.

The second step of the approach from Chen et al. would be the calculation of the region similarities and then group highly similar regions. As explained before, the regions in this work are equal to countries and therefore already widespread. For this reason, the regions are not grouped to larger ones.

The region creation and the extraction of the features was not done within the developed software as originally intended. This is because of the limited time frame and some organizational problems. However, the measurements for each region were extracted from the database and evaluated in Microsoft Excel\textsuperscript{16}.

\textsuperscript{15}https://play.google.com/store
\textsuperscript{16}https://products.office.com/en-us/excel
5 Results

In this chapter, the collected data are presented. Firstly, the raw data collected by the phones are illustrated. Secondly, the data grouped to regions as described in the previous chapter, are presented.

5.1 Raw Data

The app was published by Google on the 5th July. Measurement results from the 8th July on (when a sufficient amount of people had installed it), until the 28th July have been used. The app stopped collecting data at this date to avoid battery and data volume consumption.

At the beginning of this chapter, the amount of collected data and the measurements geographic locations are illustrated. Furthermore, the amount of collected data per phone and the countries are shown. Finally, the quality attributes per web service are presented.

5.1.1 Data Amount and Locations

In the 21 days of collecting data, a total amount of 43,686 measurements were taken. From the total amount of measurements, 14,563 measurements were collected for Lang & Schwarz and 14,563 each for WebserviceX and Yahoo.

The data were collected from 30 phones in total. Figure 5.7 illustrates the different geographic locations of the measurements on a map.

As can be seen, most collected data are from Europe and North America. Additionally, data from South America and Southeast Asia could be collected.
The 30 phones thereby collected data from 12 different countries, which are:

- Canada
- Colombia
- Denmark
- Germany
- Hungary
- Mexico
- Netherlands
- Spain
- Sweden
- Ukraine
- Thailand
- Turkey
### 5.1.2 Measurements per Phone

Since it could not be ensured that every user installed the app directly, not every phone transmitted data from the beginning. Additionally, phones are personal devices and might not always have an internet connection and are not running 24 hours a day. For these reasons, the amount of transmitted data from each device is very diverse.

Table 5.6 shows how many measurements each phone has done and from which country the measurements were made:

<table>
<thead>
<tr>
<th>Phone</th>
<th>Measurements</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2304</td>
<td>Sweden</td>
</tr>
<tr>
<td>4</td>
<td>2487</td>
<td>Colombia</td>
</tr>
<tr>
<td>5</td>
<td>2688</td>
<td>Germany</td>
</tr>
<tr>
<td>6</td>
<td>1245</td>
<td>Colombia</td>
</tr>
<tr>
<td>7</td>
<td>384</td>
<td>Sweden</td>
</tr>
<tr>
<td>8</td>
<td>1731</td>
<td>Sweden, Spain, Ukraine</td>
</tr>
<tr>
<td>10</td>
<td>129</td>
<td>Canada</td>
</tr>
<tr>
<td>11</td>
<td>1782</td>
<td>Mexico</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td>Hungary</td>
</tr>
<tr>
<td>13</td>
<td>21</td>
<td>Mexico</td>
</tr>
<tr>
<td>14</td>
<td>1293</td>
<td>Germany, Netherlands</td>
</tr>
<tr>
<td>15</td>
<td>2118</td>
<td>Germany</td>
</tr>
<tr>
<td>16</td>
<td>2838</td>
<td>Germany</td>
</tr>
<tr>
<td>17</td>
<td>2892</td>
<td>Germany</td>
</tr>
<tr>
<td>18</td>
<td>2055</td>
<td>Sweden</td>
</tr>
<tr>
<td>19</td>
<td>1446</td>
<td>Sweden</td>
</tr>
<tr>
<td>20</td>
<td>2400</td>
<td>Germany</td>
</tr>
<tr>
<td>21</td>
<td>2715</td>
<td>Germany</td>
</tr>
<tr>
<td>22</td>
<td>891</td>
<td>Germany</td>
</tr>
<tr>
<td>23</td>
<td>1266</td>
<td>Germany</td>
</tr>
<tr>
<td>24</td>
<td>402</td>
<td>Turkey</td>
</tr>
<tr>
<td>25</td>
<td>2277</td>
<td>Mexico</td>
</tr>
<tr>
<td>26</td>
<td>2745</td>
<td>Germany, Netherlands</td>
</tr>
<tr>
<td>27</td>
<td>72</td>
<td>Sweden</td>
</tr>
<tr>
<td>28</td>
<td>240</td>
<td>Mexico</td>
</tr>
<tr>
<td>29</td>
<td>973</td>
<td>Mexico</td>
</tr>
<tr>
<td>30</td>
<td>285</td>
<td>Germany</td>
</tr>
<tr>
<td>31</td>
<td>1071</td>
<td>Mexico</td>
</tr>
<tr>
<td>32</td>
<td>69</td>
<td>Thailand</td>
</tr>
<tr>
<td>33</td>
<td>2837</td>
<td>Denmark, Sweden</td>
</tr>
</tbody>
</table>

Table 5.6: Measurements of each phone

The non-consecutive phone numbering is caused by the fact that not all phones transmitted data and consequently are not listed. As can be seen, the most measurements are from Germany, Sweden and Mexico. Namely, eleven phones collected data from Germany, seven from Sweden, six from Mexico, two each from Colombia and Netherlands, one each from Canada, Denmark, Hungary, Spain, Thailand, Turkey and Ukraine.
5.1.3 Measurements per Service

Figure 5.8 shows the performance (based on the response time) of the three web services as the median response time of each service. The calculation is based on all collected measurement results with a response time greater than 0.

![Median Response Time](image1)

Figure 5.8: Median response times per service

Additionally, figure 5.9 shows the average response time for all three measured web services. Similar to the median response time data, all collected measurements with a response time greater than 0 were used for the calculation.

![Average Response Time](image2)

Figure 5.9: Average response times per service
5.2 Region Data

In this chapter, the resulting data from the region creation process, as described in chapter 4.3, are illustrated. Table 5.7 shows the resulting regions, how many phones collected data per region, and how many measurements are related to each region.

<table>
<thead>
<tr>
<th>Region</th>
<th>Amount of Phones</th>
<th>Measurements Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germany</td>
<td>11</td>
<td>21586</td>
</tr>
<tr>
<td>Sweden</td>
<td>7</td>
<td>6630</td>
</tr>
<tr>
<td>Mexico</td>
<td>6</td>
<td>6325</td>
</tr>
<tr>
<td>Colombia</td>
<td>2</td>
<td>3714</td>
</tr>
<tr>
<td>Netherlands</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>Canada</td>
<td>1</td>
<td>2118</td>
</tr>
<tr>
<td>Denmark</td>
<td>1</td>
<td>2731</td>
</tr>
<tr>
<td>Hungary</td>
<td>1</td>
<td>2118</td>
</tr>
<tr>
<td>Spain</td>
<td>1</td>
<td>505</td>
</tr>
<tr>
<td>Ukraine</td>
<td>1</td>
<td>706</td>
</tr>
<tr>
<td>Thailand</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>Turkey</td>
<td>1</td>
<td>401</td>
</tr>
</tbody>
</table>

Table 5.7: Measurements per Region
5.2.1 Measurements per Region and Service

Figure 5.10 and 5.11 shows the performance (based on the response time) for each region and web service. Figure 5.10 shows the median response time of each web service within each region. The calculation is based on all collected response times greater than 0 within a region.

Figure 5.10: Median response times per region and service
Similarly, figure 5.11 shows the average response time for each region and web service. Just as in figure 5.10, all collected response times greater than 0 were used for the calculation.

Figure 5.11: Average response times per region and service
6 Analysis

In this chapter, the results of this work will be analysed with help of the introduced approach in chapter 4.3, which is based on the research described in chapter 2.

With the help of the developed app, it was possible to collect real world measurement data from different geographic locations. Measurement data from 30 phones within 12 countries could be collected. Due to fact that the author lived in Germany and Sweden, the majority of the users who installed the app were from these two countries. The measurements were taken for three functionally equal stock exchange web services. Namely, Lang & Schwarz, WebserviceX and Yahoo.

The results in chapter 5.1.3 show the median response time of each service for all measurement results. Lang & Schwarz is clearly the fastest with 608 ms, then WebserviceX with 835 ms and finally Yahoo with 927 ms. These results stand for the approach to not consider the geographic location in the optimal service selection.

On the contrary, the results in chapter 5.2 are standing for the approach to divide users into regions, as described in the theory chapter. Figure 5.9 shows the median response times per region. As can be seen, WebserviceX, which is located in the United States, has the best performance for the regions Canada, Colombia and Mexico. In fact, the regions closest to the location of the web service. Furthermore, WebserviceX showed the best performance in the Netherlands. One reason could be the fact that the Netherlands have optimal access to transatlantic cables. However, since only 22 measurements were taken from the Netherlands, this result has to be taken with caution. While for the most other regions, the web service from Lang & Schwarz usually showed the best performance, but the results still showed significant differences from region to region. For example, the median response time for Lang & Schwarz in Hungary was 158 ms, whereas WebserviceX as the second fastest had a median response time of 962 ms. In other regions, the three web services do not significantly differ from each other. For instance, in Spain the median response time was 798 ms for Lang & Schwarz, 1097 ms for WebserviceX and 1073 ms for Yahoo.
7 Discussion

In this chapter, the results of this research are discussed and reflected.

7.1 Results

As described in chapter 1.2, the purpose of this thesis is to identify how the geographic location can be used for the optimal service selection, and if it has a positive impact to the optimal service selection.

To be able to answer this and for defining further research, a mobile app including a backend was developed to collect measurement results with the geographic location of the user. This first goal of this work is entirely achieved. The app reliably collects measurements and transmits them to the backend. During the three weeks of collecting measurement results, no error related to the app or backend occurred. Consequently, the app can be used for further research in the field of optimal service selection. While in this thesis project the time frame was limited, especially the long term data collection with the app is of interest. In the opinion of the author, this could lead to more accurate results and to new knowledge about the interplay between the geographic location and service performance. Additionally, finding more users from even more places would bring the research forward. Making the app more flexible could help to achieve this goal. One suggestion for improvement is to provide the possibility to configure the web services and backend within the UI. Undoubtedly, this would simplify the reuse of the app for similar research projects.

The second question of this thesis, namely how the geographic location can be used for the optimal service selection, could partly be answered. The analysed results show clearly that the performance of web services is highly correlated to the geographic location of the user, and that grouping users into regions can lead to a significantly improved optimal service selection. The approach described in chapter 2.2 is partly used in this research. The idea of the region creation and extracting region features is adopted in this research. However, due to a low number of measurement results, the regions were created on basis of countries and no region aggregation is applied. Furthermore, due to the limited time frame and some organizational problems, the machine learning algorithm of the existing framework could not be extended to consider the geographic location in the optimal service selection. Nevertheless, with the presented approach, the existing algorithm can be used nearly unmodified. Only the region creation, feature extraction and the assigning of new users to a region has to be implemented. With further data collection, the process of region creation can be made more precise and then lead to a more accurate optimal service selection.
7.2 Method Reflection

The described approach in chapter 3, developing an app for the data collection turned out to be an easy way to collect data from a variety of users and different locations. Using the Google App Engine as backend platform allowed to concentrate on the software features instead of the required infrastructure. Indeed, on one hand the decision to only develop an Android app limited the complexity of this work. On the other hand, more users would have collected measurement data if also an iOS\textsuperscript{17} version would have been available. Especially in North America, plenty of people asked by the author had an Apple phone. One solution for this problem could be to develop a cross-platform app with PhoneGap\textsuperscript{18} or similar solutions.

\textsuperscript{17}http://www.apple.com/ios/
\textsuperscript{18}http://phonegap.com/
8 Conclusion

This chapter gives a conclusion of this thesis by reflecting the developed software, its results and answering the purpose of this research.

Based on the hypothesis that using the geographic location has a positive impact on the optimal service selection, a mobile app and backend to collect measurement results have been developed in this work. During three weeks about 40,000 measurements results from 12 different countries could be collected. The results were then grouped into regions according to the geographical location. Therefore, the approach described in chapter 2 was adapted to the circumstances of this research project. Comparing the raw data with the region data clearly showed that the geographic location of a user has impact on the web services response times and consequently also on the optimal service selection. Extending the existing machine learning algorithm in the service selection framework of the research project to consider the geographic location in the optimal service selection process could not be solved in this thesis project.

8.1 Further Research

As mentioned in the discussion chapter, an important factor for further research is to collect more real world test data including the users’ location. Therefore, the app should be made available for other mobile platforms like iOS and Windows Phone. With more collected data, the region creation process could be improved to consider more quality attributes regarding the region features. Furthermore, the existing framework of the research project can be adapted to reflect the knowledge gained in this research.

To compare the existing approach of an optimal service selection with the approach described in this thesis, only the median response time as main quality attribute was used in this thesis. In further research other quality attributes should be considered as well. Also the performance differences of web services during different day times in conjunction with the geographic location is an interesting field to investigate in further research.
References


A Appendix 1

In the following, the measurements per phone are illustrated.

A.A Appendix 1.1 - Median Response Times per Phone

Figure 1.12: Median response times per phone
Figure 1.13: Average response times per phone