universität
wien

MASTERARBEIT

Titel der Masterarbeit

Integrating Linked Data in Mobile Augmented Reality Interfaces - Development of a Mobile AR Mountain Guide

Verfasserin

Silvia Fürst, BSc

angestrebter akademischer Grad

Diplom-Ingenieurin (Dipl.-Ing.)

Wien, 2012

Studienkennzahl lt. Studienblatt:	A 066 935
Studienrichtung lt. Studienblatt:	Medieninformatik
Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas

Abstract

Today's mobile phones (so-called "smartphones") present powerful platforms and can be used as gateway from the real to a virtual world. Therefore, they are an attractive target in the domain of mobile Augmented Reality (AR). The basic idea of AR is to mix the real with the virtual world by overlaying the user's field of view with digital information (virtual graphics) to describe the immediate physical environment. Thus, AR on mobile phones allows users to directly access information that is relevant to a specific place or monument at anytime and anywhere.

Currently, many mobile AR applications present client-server-based environments that make use of proprietary data formats and access restricted data sources. Furthermore, they may be based on a vision-based tracking system which includes hardware resource consuming image pattern algorithms that require permanent connection to a server-side platform and do not allow real-time execution on mobile phones. Thus, this work focuses on sensor-based tracking which relies on the GPS radio, accelerometer, and digital compass of a mobile device to determine the current position, orientation, and heading of the device. Sensor-based AR tracking systems include a mathematical model that uses the information of the camera pose and GPS location to project real-world objects into the captured view of the device. The main contribution of this thesis is the conceptual and formal description of such a mathematical model.

Furthermore, AR systems need access to data sources to be able to identify real-world objects. Therefore, the Linked Open Data (LOD) project which provides a huge amount of interlinked data and is based on open standards and APIs, presents an appealing data source for AR systems. In consequence, this work also investigates the exploitation of Linked Data resources for integrating them into mobile AR systems in order to visualize LOD resources in AR interfaces and to identify and describe real-world objects. We demonstrate the feasibility of the presented approach in this work through the development of a proof-of-concept AR application that runs in real-time and allows the identification of real-world mountains. The system fetches Linked Data resources from two different LOD sets of the Web and consolidates (filters duplicate resources and merges them) and stores the resources directly on the mobile phone, to finally visualize them onto the screen of the device. The prototype is executable without permanent network connection and therefore does not depend on any server-side platforms.

Zusammenfassung

Heutige Handys, sogenannte “Smartphones”, stellen mächtige Umgebungen dar, die als Gateway von der realen zu einer virtuellen Welt eingesetzt werden können. Deshalb sind sie ein attraktives Ziel in der Domäne der mobilen Augmented Reality (AR). Die Grundidee von AR ist das Kombinieren der realen mit einer virtuellen Welt, indem das Livebild der Kamera mit digitaler Information überlagert wird, um auf diese Weise die unmittelbare physikalische Umgebung zu beschreiben. Folglich, erlauben mobile AR Systeme, dass User direkt auf standortspezifisch relevante Information jederzeit und überall zugreifen können.

Derzeit stellen viele mobile AR Applikationen Client-Sever-basierte Umgebungen dar, die proprietäre Datenformate verwenden und nur auf beschränkte Datenquellen zugreifen. Weiters sind einige AR Systeme auf visuelle Trackingsysteme aufgebaut. Visuelles Tracking beinhaltet komplexe Bildanalysealgorithmen, die die Hardware Ressourcen von Mobilgeräten leicht überfordern können und meistens eine permanente Verbindung zu einer serverseitigen Plattform erforden. Dies führt dazu, dass visuelle AR Trackingsysteme oft nicht in Echtzeit auf Handys ausgeführt werden können. Aus diesem Grund basiert der Ansatz dieser Arbeit auf ein Sensorbasierendes Trackingsystem, welches den GPS-Chip, Beschleunigungssensor und digitalen Kompass des Handys benutzt, um die Position, Orientierung sowie Ausrichtung des Gerätes zu messen. Solche Systeme beinhalten ein mathematisches Modell, welches die Information über die Kameraausrichtung und der GPS Positionierung des Handys verwendet, um Objekte aus der realen Welt auf das aufgenommene Bild des Gerätes zu projizieren. Der Kernpunkt dieser Arbeit stellt die konzeptionelle sowie formale Beschreibung eines solchen mathematischen Modells dar.

AR Systeme benötigen Zugriff auf Datenquellen, um Objekte aus der realen Welt identifizieren zu können. Das Linked Open Data Projekt, welches eine enorme Menge an verlinkten Daten umfasst und auf offenen Standards und APIs basiert, stellt dabei eine verlockende Datenquelle für AR Systeme dar. Daher wird in dieser Arbeit die Exploitation von Linked Data Ressourcen in mobilen AR Systemen untersucht, um diese in AR Interfaces integrieren zu können. Die vorliegende Arbeit beinhaltet die Entwicklung einer mobilen AR Applikation, die in Echtzeit ausführbar ist und die es erlaubt Berge der realen Welt zu identifizieren. Das System greift auf Linked Data Ressourcen von zwei verschiedenen LOD Plattformen des Webs zu und konsolidiert (das Herausfiltern von identischen Ressourcen und das Zusammenführen dieser) und speichert die Ressourcen direkt auf dem Handy. Der Prototyp ist ausführbar, ohne dass eine permanente Netzwerkverbindung zu einer Serverumgebung besteht. Daher ist die präsentierte AR Applikation unabhängig von jeglicher serverseitigen Plattform.

Acknowledgements

First of all, I would like to thank my supervisor Univ.-Prof. Dr. Wolfgang Klas and my secondary supervisors Dr. Stefan Zander and Dipl-Ing. Chris Chiu for all their thoughts, ideas, support, and help along the way and who gave me the possibility to complete this thesis. A special thanks goes out to my fellow student and friend Evelyn Hinterramskogler with whom I had many useful discussions. Finally, I would like to express my gratitude to my family and friends for all their support and encouragement.

Table of Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The continuous technical sophistication of today's mobile devices led to the creation of new application domains. One domain that currently receives enormous attention is that of mobile Augmented Reality (AR). AR enables the supplementation of the real world with digital information by using the mobile phone's camera as video-see-through interface [1].

On the other hand, semantic data on the Web has increased significantly in the past few years, mostly because of projects like the Linked Open Data (LOD) initiative[1]. The Linked Data paradigm allows the publishing of data on the Web based on a set of specific principles (cf. Section 2.2) and interlinks semantic data resources (e.g. real-world entities) through common global identifiers (HTTP URIs). Many data providers have followed these principles the last few years. Now, the so called "Web of Data" comprises descriptions about huge amounts of resources from various topical domains such as people, animals, music but also physical locations [2].

Many mobile devices (especially smartphones) are powerful enough to obtain the current position on earth (through the device's built-in GPS chip) in order to use location information as entry point to the Web of Linked Data. Thus, we could use Linked Data resources to identify and describe real-world objects of our physical surroundings and visualize Linked Data in AR interfaces of mobile AR systems. In consequence, the objective of this work is to merge AR technologies with the principles of Linked Data. Thereby, we aim to realize a new approach for accessing and discovering the Web of Linked Data within mobile AR interfaces.

1.1 Motivation

Today's powerful mobile phones (also called smartphones) already present an everyday companion of an uprising number of people because they are a helpful assistant for people's personal (e.g. surfing in the Internet, taking pictures, sending messages) or business (e.g. taking business calls, looking up emails) life. Pence [1] describes smartphones as "Swiss Army knifes of modern

[1]LOD community project: `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData/`

communication" because a smartphone not only presents a telephony device but also is equipped with additional features such as GPS radio, digital camera, digital compass etc. Access to online information is possible almost anywhere, as smartphones provide mobile networking. With their growing computational power smartphones present powerful platforms that are able to support mobile AR and thus the availability of AR systems on mobile devices already increased significantly in the last few years [3].

Fundamental to the creation of mobile AR systems are accurate *tracking* techniques and *registration* methods [4]. The term tracking in combination with AR systems stands for the method of gathering permanently the field of view of the user by determining the current location, orientation, and heading of the mobile device. Only an accurate tracking method allows the following process of registration to work accordingly [5]. Registration is the method of constantly aligning virtual objects with real-world objects. This is essential because the field of view is changing on occasion by moving the device. Despite the movements, virtual and real objects need to stay aligned, otherwise the AR system will not work correctly, as the real-world objects would be wrongly annotated and in consequence, the AR system would not be accepted by the user. As there exist different kinds of AR tracking methods, we further discuss them in the next section.

1.1.1 AR Tracking Systems

For a mobile AR system to work properly, we are able to apply a *vision-based*, a *sensor-based* or a *hybrid* AR tracking method:

The vision-based method is based on image matching algorithms. For instance, a vision-based method can include the capturing of pictures through the phone's integrated digital camera and the subsequent comparing of them with reference pictures on a server-side database. Comparing mechanisms are usually based on previously extracted features of the pictures. If the photos match with those in the database, it is possible to identify an object and further information about the recognized object can be gathered. But a vision-based tracking method requires the use of complex pattern algorithms to determine matching pictures. They may also be resource consuming and may overburden the device's hardware [6]. Moreover, a vision-based tracking system, only tracks objects which are in the view-point of the camera. To be able to compare images in first place, it is necessary to set up a database of recorded reference images. Thus, also a permanent connection to a server-side environment is necessary to compute image pattern recognition algorithms. In addition, the robustness and accuracy of vision-based tracking algorithms is effected by outdoor environmental conditions such as lighting conditions or bad viewing conditions [7]. For instance, Ravi et al. [8] outline that the lightening conditions for the recorded pictures change over time and may affect results of image comparisons. In consequence, Ravi et al. suggest to build an image database, which contains day-time images but also the night-time versions of them to overcome lightening problems. We discuss examples of vision-based approaches in Section 3.1.1.

A sensor-based tracking method, on the other hand, requires the development of a mathematical model that depends on several built-in sensors of a smartphone such as digital camera, GPS radio, digital compass, and accelerometer. A sensor-based tracking system comprises a complex algorithm which uses the GPS position of the device, sensor data (orientation and direction information recorded by the device), and the geographical information (GPS coordinates) of real-world objects (objects that have to be tracked) to determine the position of the real-world objects relative to the current field-of-view of the device's camera. Thus, the mathematical model is responsible for the calculation of screen coordinates in order to display AR content within an AR interface in perfect alignment to the real-world objects. To avoid drift the mathematical model has to update the screen coordinates of virtual objects periodically with the current orientation and heading measurements [9]. A sensor-based approach allows to display each real-world object of which geographical information is available. In consequence, also objects which are covered by other objects are visualized by the AR system. This is not possible if we utilize a vision-based tracking system.

Hybrid tracking systems combine techniques of sensor and vision-based methods. We describe examples of hybrid approaches also in Chapter 3.

In our work, we concentrate on a sensor-based tracking approach. Hybrid and vision-based tracking methods, may require huge amounts of reference images on the client or service-side, as well as complex image matching and classification algorithms. Thus, the integration of these tracking approaches in a mobile AR system may easily overburden the processing power of mobile phones. We further discuss this in Chapter 3.

1.2 Problem Description

In this section, we refer to two main problems which motivate our concentration on AR technologies on mobile phones and the additional integration of the principles of Linked Data to such AR systems. We start with a discussion about issues that a normal 2D map view exhibits in the context of spatial orientation. In the last part of this section, we discuss the problem that most currently available AR systems use proprietary data formats and operate on top of closed data repositories for delivering information about real-world objects.

1.2.1 2D Maps versus AR Interfaces

The implementation of an AR interface for visualizing resources might be advantageous in contrast to the visualization resources on a 2D map: a map view can lead to multiple user interactions to present the information the user desires [10]. AR allows the developing of more conceivable UIs to ease user navigation and provide a more intuitive presentation of information. Raubal and Panov [10] discuss several problems regarding user interaction with map-centric mobile applications. In the following, we outline the most prominent problems with map representations:

- **North-up alignment of map:** Most digital maps are aligned with the north-up direction (up on the map corresponds to north in the environment) just as analog maps. Based on a study conducted by Seager and Fraser [11], people tend to rotate a map displayed on a device by physical rotation towards a forward-up view (top-alignment of the map corresponds to the environment that is in front of the viewer). In consequence, every direction change by the user will lead to the additional user interaction of physical rotating the device until the map is aligned with the real world. Then, users are able to orient themselves more easily. Thus, Seager and Fraser [11] claim that people are able to understand maps easier when they are forward-up aligned. They consolidate their claim by referring to other accomplished studies [12–14].

- **Zooming, user position, and static map extent:** For maintaining orientation in relation to the surrounding environment, the user position is usually displayed by a marker in the center on a digital map of a mobile phone. To be able to get an overview of the landscape or a more detailed view to observe visualized resources in contrast to the real-word objects, users have to make use of the map scale or they have to navigate on the map in order to extend the map view.

The comprehension of maps requires cognitive and spatial abilities [15]. If the user understands the spatial relationship between the real 3D world and the flat 2D map, it is possible that the user comprehends the map and is able to reference real-world objects with objects displayed on the map [16]. Despite the fact that 2D maps are a common visualization technique of the real world, AR interfaces represent an efficient yet alternative way for visualizing geographical information and also for accessing location-based data [6]. An AR interface reduces the number of user interactions which a normal 2D map presentation requires (e.g. zooming, map rotation) because the user only has to point the device over the real-world object (about which the user requires more information) and has to wait until information is screened on the display. The basic idea of AR is to place information directly on top of real-world aspects rather than presenting information on isolated displays. By wrapping complex AR algorithms in easily conceivable and intuitive UIs, AR allows to blur the distinction between the real world and the UI [17]. Thus, users are not involved with complex cognitive processes to understand maps and to maintain orientation on them. Suomela and Lehikoinen [6] confirm that AR UIs are intuitive for the user because if virtual objects are integrated in live scenes, they become parts of the real world and the user naturally knows how to interact with the real world.

1.2.2 Closed Datasets

Wikitude World Browser[2] and Layar Reality Browser[3] are two mobile AR browsers developed for smartphones which enable to explore Points of Interest (POI) (cf. Definition 1.1 according

[2]Wikitude World Browser: `http://www.wikitude.org/`
[3]Layar Reality Browser: `http://www.layar.com/`

to the definition of [18]) through a mobile phone's camera. They present the two most known applications in the domain of mobile AR and are based on sensor-based tracking methods, as introduced in Section 1.1.1. Wikitude and Layar exhibit a similar architectural structure which comprises:

- an AR client presenting the running AR browser on the mobile phone,
- multiple provider servers which provide AR content[4],
- and an AR server acting as handler between the client and the provider servers.

> **Definition 1.1** (Point of Interest (POI))**.** *Is an individual data item typically associated with a geographic location (longitude, latitude, altitude) or a visual pattern (marker, book cover etc.) that can be rendered in some way by the AR application. The Point Of Interest data type must provide a description of the location or reference image used in tracking and the type of content to be rendered.* (Butchard, 2011)

The user is able to choose between several AR overlays to load a certain set of POIs from a specific content provider (e.g. Wikipedia[5] articles, YouTube[6] videos) into the AR interface [18]. Thus, POIs are hosted in isolated data repositories of the respective content providers.

However, these mobile AR systems exhibit severe problems and limitations regarding especially the integration of data sources. Although, they offer tools and APIs for publishing user-generated content, they make use of proprietary data formats. Thus, a content creator must contend with restricted control over the publishing of their own content [19] and they also have to use a specific data format for each different AR system. Kim et al. [20] claim that the providers of AR browsers *"lack a 'generally applicable and comprehensive' AR content model"*.

In addition, AR browser platforms, such as Wikitude and Layar, do not include the possibility of adding user-generated content to extend information about a specific AR content. Furthermore, data which are used in these systems can not be shared or reused in other systems. The user might experience domain-specific browsing because the displayed AR content is fetched from static isolated data silos. AR content is not interchangeable between different AR browser platforms. Thus, there is a need of data federation in order to fetch information about a single resource from multiple sources. Then, the browsing experience becomes more extended.

1.3 Attempt for Linked Data Integration

In this work, we investigate the integration of LOD resources in mobile AR systems. The principle of Linked Data relies on the publishing and the connection of structured data on the Web under the consideration of certain publishing principles (cf. Section 2.2). Bizer et al. [21]

[4]Data displayed by AR browsers to describe real-world objects.
[5]Wikipedia: `http://en.wikipedia.org/`
[6]YouTube: `http://www.youtube.com/`

explained the LOD community project as the most visible example of adoption and application of the Linked Data principles. The entities (resources) of the LOD sets are identified through Uniform Resource Identifiers (URIs), thus resources and resource descriptions are retrievable directly through HTTP or via a SPARQL endpoint[7] [2]. The *Resource Description Framework* (RDF)[8] is used as data format and presents a graph-based data model which enables information descriptions about things in the world by modeling data as statements in form of RDF triples (subject, predicate, object) (cf. Section 2.2.2) [21]. The RDF data model allows to set RDF links between related entities of different LOD sets and to generate in consequence a Web of interlinked data.

Everyone is able to contribute to the LOD project by publishing RDF data and interlinking the data with existing data of other Linked Data sets according to the Linked Data paradigm. This openness already led to a considerable growth of Linked Data sets, from one data set in the year 2007 to nearly 300 data sets (containing over 31 billion RDF triples with around 504 million RDF links between them) in the year 2011 [22]. As obvious, the number of data sets has more or less doubled the last few years.

The huge amount of available interlinked data can be used in AR systems to describe real-world aspects. Thereby, geographical information about the current whereabouts of the user's device is a possible entry point to the Web of Linked Data. The LOD paradigm enables it to federate data from several different LOD sets to provide a comprehensive information source for mobile AR systems. However, the federation of data from different LOD sets leads to the occurrence of duplicate resource descriptions. Linked Data allows to define rules in order to integrate and consolidate LOD resources properly in an AR system. In consequence, it is possible to filter duplicate resources and to merge the remaining resources to a single RDF graph. For instance, the identical resource detection can be based on a similarity metric such as the Jaro-Winkler[9] distance metric. We further discuss this in Section 4.3.2.

In the previous section, we mentioned that current mobile AR systems do not enable the adding of information to a specific identified resource in order to extend the resource description. In contrast, the principle of Linked Data (cf. Section 2.2) allows to link data to related data to extend resource descriptions.

1.4 Contribution

The contribution of this work presents the creation of a formal and conceptual definition of a mobile AR system that integrates LOD into AR interfaces. We demonstrate the feasibility of our approach through a prototypical implementation of a mobile AR application for Android[10] phones to identify mountains. The prototype federates LOD resources from different

[7] HTTP-based query service that executes queries against Linked Data sets. cf. `http://www.w3.org/TR/rdf-sparql-query/`

[8] Resource Description Framework: `http://www.w3.org/RDF/`

[9] Jaro-Winkler: a measure of similarity between two strings.

[10] Andorid: `http://www.android.com/`

LOD sets and processes (filtering identical resources, merging and storing remaining resources) data directly on the mobile device. Furthermore, our approach focuses on the creation of an AR system that is built upon a sensor-based tracking method realized through a complex mathematical model. This model comprises a tracking algorithm that takes recorded sensor data of the device and the geographical information of the processed LOD resources for determining proper screen coordinates to place virtual content on top of real-world objects on the screen of the mobile device.

Our approach of integrating Linked Data resources into mobile AR systems exposes three significant advantages compared to other existing mobile AR application approaches:

1. Most of the existing mobile AR applications exhibit closed architectural environments and a closed object model because they use their own proprietary data formats for delivering relevant data and they require mostly a permanent network connection. In those AR systems, content providers have to provide data in a specific format. Consequently, they have to adapt their content to each AR browser system at which they publish data. In contrast, LOD makes use of HTTP which presents an uniform data access mechanism and RDF which presents a standardized graph-based data model [21]. Our approach exploits Linked Data as AR content and thus may present a possible approach to standardize AR content by processing RDF-based data directly on the client side of a mobile AR system.

2. Furthermore, our intent is to prevent from the additional buffering and processing of AR content on platform-specific Web servers. Our approach allows the use of the AR system without a permanent network connection. We try to utilize available Linked Data Web services and SPARQL endpoints to directly access the freely available data of the LOD community, to federate, process, and store the data on the mobile device, and to use the processed LOD resources to describe certain location points within an AR interface. The storing of already processed data on the mobile device enables a later execution of the system without maintaining network connection.

3. The possibility of browsing for further information about a specific resource is often not provided. AR content sources of existing mobile AR applications systems are locked down in isolated data repositories. Only content that is published by providers can be retrieved. The principle of Linked Data includes that RDF links can be followed at run-time. By using Linked Data sets as AR content sources, each resource is then identified by an URI (each RDF resource presents an URI). This URI can be accessed if additional information is requested. RDF descriptions of resources include links to related resources and these can be followed link by link which lead do a more extended browsing experience. Our approach is not restricted to a fixed set of data sources. The system retrieves resources from multiple Linked Data sets.

We believe that our work contributes to the LOD initiative by providing a new type of utilization for Linked Data in the domain of mobile AR computing.

Therefore, we specifically investigate the following research questions:

- What aspects have to be considered in developing a mathematical model that is capable of presenting relevant aspects of the real world on the corresponding positions in an AR interface?
- What elements should the AR tracking algorithm of the model comprise, to enable a constant calculation of proper screen coordinates for visualizing content on top of real-world aspects within an AR interface?
- How can Linked Data resources be exploited, to be able to integrate and visualize them in an AR interface?

The proposed prototype of this work represents an AR mountain guide for identifying mountains live in an AR interface. The application also offers the possibility to look up hiking tours for a certain restricted area near the physical location of the device. Thus, we define an additional research question:

- What kind of platform can serve as possible source for fetching hiking tours and how will it be possible to display GPS tours saved within a GPX (GPS eXchange Format - GPS data format) file on a map or an AR interface?

We already address the two main research questions in Chapter 3 and later in Chapter 4, where we present and discuss our approach. At last, we demonstrate the implementation of our approach in Chapter 5. The third research question is only briefly addressed within Chapter 3, and then further discussed in Chapter 5.

On the basis of the defined research questions, we also defined three main research objectives for this thesis:

- Evaluating the possibilities and requirements for developing an AR tracking method realized through a mathematical model which takes advantage of the different sensors available on current smartphones (e.g. digital compass, accelerometers etc.)
- Identifying of capabilities for exploiting certain Linked Data resources as AR content within the model's tracking algorithm so that it takes the current GPS information of the device, the geo-locations of the resources near the device, and orientation and direction sensor data in order to determine which mountains are in the current captured view of the device's digital camera view, and that it calculates in consequence screen coordinates for displaying the aspects of the real world on the corresponding positions in an AR interface.
- Elaborating on methods for the aggregation of Linked Data to be able to display federated data manifested in extended descriptions about a specific mountain.

Considering the fourth research question of this work, another research objective of this work is the elaboration of a method that displays downloaded GPS tracks on a 2D map or in an AR interface.

1.5 Thesis Organization

This thesis is structured as follows:

Chapter	Title	Description
Chapter 1	Introduction	Provides an introduction into the motivation, attempt, and contribution of this work and describes the specific objectives.
Chapter 2	Background	Presents a literature review and offers an overview on the technical background of this work by providing insights in AR, LOD, Geodesy, and the Android OS.
Chapter 3	Related Work	This chapter reports on publications and specific AR tracking systems. We compare similar or somehow related AR approaches and provide a current state-of-the-art analysis.
Chapter 4	Approach	Describes our approach and the concepts behind the development of a sensor-based tracking method for a mobile AR system that integrates and processes Linked Data resources fetched from different LOD sets as AR content. We outline how we exploit LOD resources, we present the conceptual constituents of our AR system, and we describe the mathematic model of the AR system on a formal basis. We conclude this chapter with the presentation of functional work flows our AR system will include.
Chapter 5	Proof of Concept	Contains implementation details of our AR prototype that is based on the conceptual architecture and the formal model presented in Chapter 4. In this chapter, we present the class structure and the data structures of the components of our AR system. The chapter also includes the presentation of several sequence processes and code snippets.
Chapter 6	Results and Discussion	Comprises the presentation (several screen shots), evaluation, and discussion of our developed mobile AR system.
Chapter 7	Final Remarks	Delivers final conclusions on this work as well as outlooks and other possible future works.

Table 1.1: Structure of this thesis

Chapter 2

Background

This chapter provides insights into the most relevant technologies and concepts used in this work and delivers an overview about the relevant literature research. First of all, we describe the basic idea and concept of AR and what application domains are existing for AR. After that, we explain the LOD project where we also describe specific LOD sets that we use within our proof-of-concept application. We explain RDF and the query language SPARQL as last topics within the LOD section. The chapter also gives a brief introduction into Geodesy[1] which is a scientific discipline that studies the shape and size of the earth. We also provide an introduction to the main concepts of GPS. We dedicade the last topic of this chapter to the Android operating system: we describe the architecture of Android, the Android SDK, and the main aspects of an Android application.

2.1 Augmented Reality

Augmented Reality can be seen as the principle of extending or complementing the real world with virtual objects in real time. The aim of AR lies in adding computer-generated information over real objects that are captured by the camera view. The augmented information let people's thoughts deepen on something they observe in the real world (e.g. a monument in the city) and thus makes their experience of the perceived object more meaningful [23]. Although many papers are dedicated to AR, there still exists no precise definition of AR until today [17]. Milgram et al. [24] explain the term AR by putting Reality and Virtual Reality (VR) as two opposites on both sides of a continuum called Reality-Virtuality (RV) continuum (cf. Figure 2.1). Unlike in Virtual Reality where the whole scene consists of artificial objects, Augmented Reality still allows the user to see the real world but combined with virtual objects.

Figure 2.1 illustrates the RV continuum of Milgram et al. [24], where the real world is placed on the left side of the continuum called Reality. In the real world, a person observes objects directly in her/his immediate surroundings. The VR is located on the right side of

[1] More information about Geodesy: `http://www.iag-aig.org/`

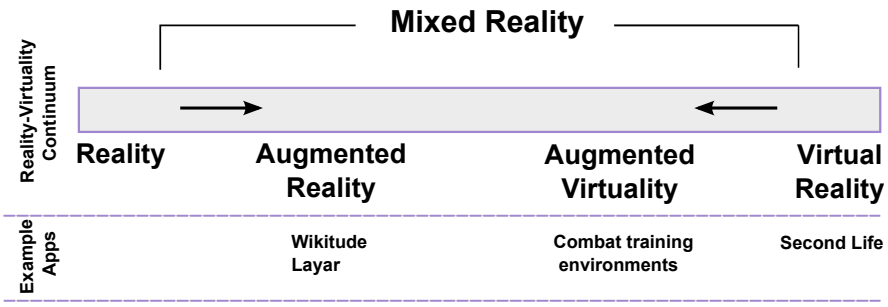


FIGURE 2.1: Milgram's Reality-Virtuality Continuum[2]

the continuum where the environment is completely artificial, so all the appearing objects are virtual objects. Examples for Virtual Realities are computer simulations, computer games etc. An example for a VR presents "Second Life". The term Mixed Reality (MR) stands for an environment where the real world and the virtual world are combined and the objects of both worlds are presented together, whereby the augmentation can happen in two different ways:

- augmenting the real world with virtual objects - *Augmented Reality* (AR)
- augmenting the virtual world with real objects - *Augmented Virtuality (AV)*

One example for AV, as mentioned in [25], is a soldier training simulation (or combat training environment) where the environment is completely artificial for instance a simulation presenting a forest where a training should be performed. The real soldier as person is operating within this artificially created training simulation whereby he only observes his comrades as real-world subjects. In Figure 2.1, we also depict examples for AR: the Wikitude World Browser (cf. Figure 2.2) and the Layar Reality Browser, two AR client applications runnable on mobile devices, which are able to superimpose the live camera view by displaying information on top of real-world aspects like monuments. They provide an interface to access and interpret AR data.

FIGURE 2.2: Wikitude overlays the camera's view with additional information about an upcoming event at a specific location at the Old Town of Salzburg [26]

[2]Based on figures of [24, 25])

According to the fact that in AR the real world is augmented with virtual objects, an AR system has to generate a composite view for the user. Azuma et al. [5] define an AR system to have the following properties (excerpt from [5]):

- *combines real and virtual objects in a real environment*
- *runs interactively, and in real time and*
- *registers (aligns) real and virtual objects with each other.*

As already mentioned in Chapter 1.1, AR systems need to cover methods for tracking and registration to be perceived as accurate AR systems [4]. Tracking aspects of the real world in the near physical vicinity and register virtual objects with physical aspects in order to keep them aligned. There already exist various kinds of tracking methods, which can be divided into *marker-based* (cf. Definition 2.1) and *marker-less* (cf. Definition 2.2) tracking.

> **Definition 2.1** (Marker-based tracking)**.** *Marker-based tracking relies on optical markers placed in the environment. The markers serve as reference points used by an AR system for identifying marker patterns to display AR content in correct alignment to the physical environment.*

> **Definition 2.2** (Marker-less tracking)**.** *This tracking method can be classified as vision-based, sensor-based, and hybrid tracking. Vision-based tracking methods rely on pattern recognition and image matching algorithms to identify current position and orientation of real-world objects in the physical environment. A sensor-based tracking method relies on the used device's built-in sensors like accelerometer, magnetometer or GPS radio and does not include any kind of image analysis processes. Hybrid tracking methods improve the performance of vision-based methods by combining them with sensor-based approaches* [27].

In Section 3.1, we present and discuss the most relevant AR tracking approaches for mobile AR systems.

2.1.1 Area of Application

First AR applications already appeared in the late 1960s. Over the years, researchers found more and more areas which could benefit from AR. Now, AR is utilized for instance in the field of medicine [28] (e.g. surgical training, AR overlay of medical scans), in the field of military (e.g. training tools, supplementing field of view for pilot during flights) or also in the field of entertainment (e.g. AR games, AR overlays during live sports broadcastings) [29].

Advances in AR tracking methods and the increasing computational power of mobile devices, led to the development of AR systems for mobile phones [5]. Mobile AR enables the

creation of AR applications supporting the user in navigation, situational awareness, and geo-located information retrieval. When combining location-aware computing with AR-based systems, the user has the possibility to discover pinpointed actual locations in the real world linked with additional electronic information through optical see-through displays. Thus, AR is a leading indicator of the next generation of Location-Based Services (LBS) [30]. To estimate orientation and direction, the phone offers a digital compass and an accelerometer. In addition, a built-in GPS chip determines the location information of the device. A mobile phone is also suitable for vision-based tracking approaches, as the digital camera of the device can be used for image recognition processes.

AR application platforms, such as Wikitude and Layar (cf. Section 1.2.2, Section 3.2.1.1), call themselves *AR browsers*. An AR browser presents a client application that displays information about things of the local surroundings on the top of the phone's camera view. They use the term "browser" because visualized data layers are like Web pages in traditional Web browsers. An AR browser retrieves content from different online sources and users are able to browse information directly in its corresponding real-world location. Today's mobile AR browsers not only allow to visualize textual overlays to pinpoint nearby physical objects, they may also provide video or audio overlays or even 3D animated objects.

2.2 Linked Open Data

Linked Open Data is a W3C community project to interlink freely available information content on the Web by means of semantic technologies [2]. Berners-Lee [31] outlined rules for the provisioning of data by linking them to each other to gain a global information space:

- URIs should be used to give things in the LOD project a name.
- In order to be able to look up those names the use of HTTP URIs is important.
- People should be provided with useful information when they lookup an URI through the usage of standards like RDF or SPARQL.
- The provided information for things has to include links to other URIs in order that more things can be discovered through the following of the links.

In Figure 2.3, we present the LOD project of the year 2011. The figure illustrates the LOD project in form of a cloud diagram. Within this diagram, each node presents a separate data set. RDF (cf. Section 2.2.2) is used to interlink data from different data sources, which means the interlinking happens in form of triples where a subject (resource in form of an URI) from one data set is linked with an object (another URI or literal) from another data set [21]. The presented LOD diagram reflects this interlinking of existing data sets (e.g. DBpedia, GeoNames, or LinkedGeoData).

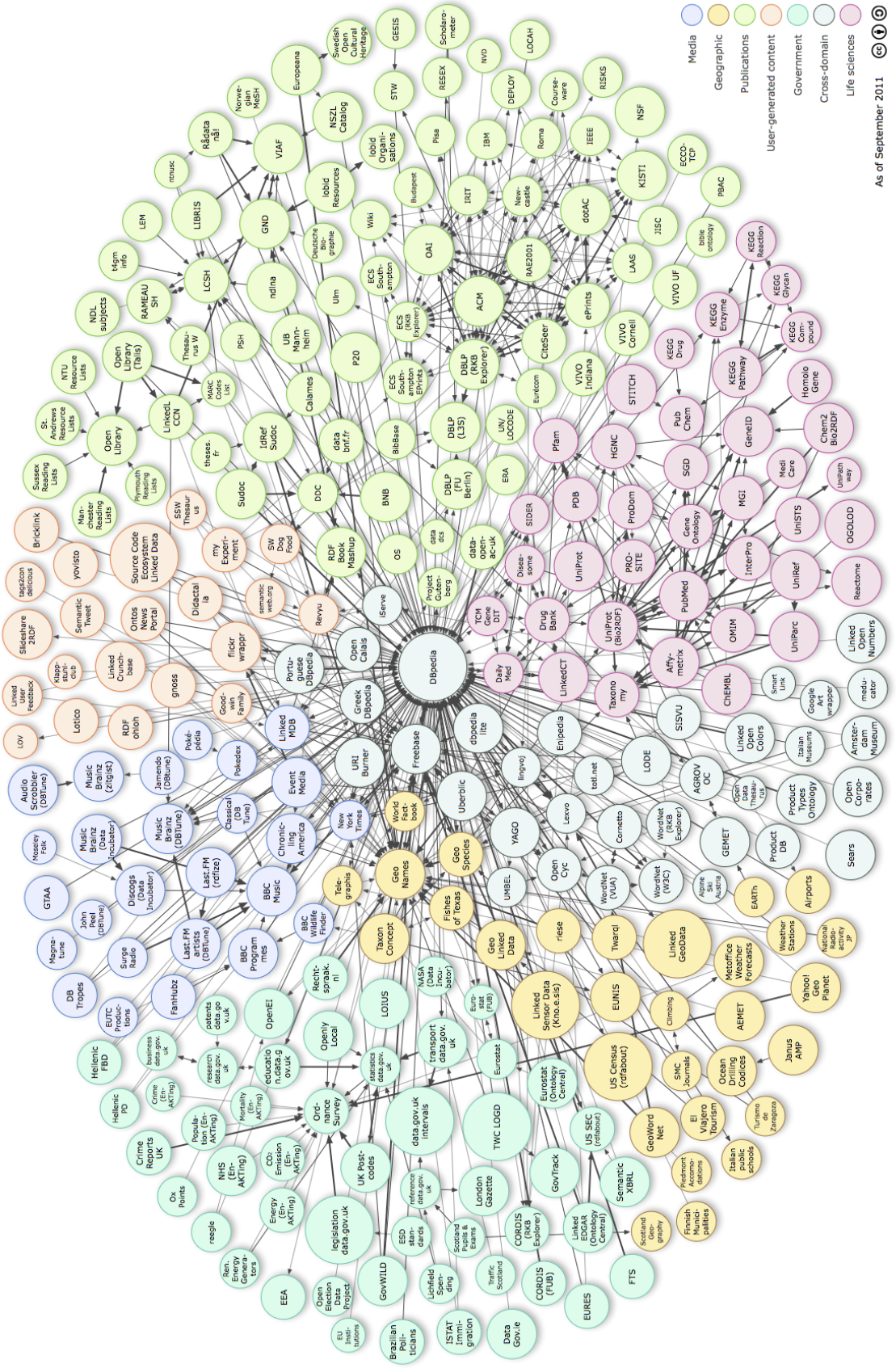


FIGURE 2.3: Linked Open Data Cloud 2011 [32]

The size of the nodes of the diagram in Figure 2.3 corresponds to the number of RDF triples provided by each data set. The arrows visible between the nodes reveal that there are at least 50 links between the data sets [33]. The thickness of the arcs indicates the number of links between them. Moreover, the content of the cloud exhibits different categories. Each category is marked through a different color. For instance, for this work, data sets which are marked yellow (data sets of category "Geographic") are important because they will be used to find out which mountains are located in a certain area. The blue marked nodes are representing linking hubs or cross-domains. A linking hub is a data set that contains a huge number of descriptions about things which are again referenced several times in other more specialized data sets. The data set is then referred to as hub because of the increasing number of links to other data sets [21]. The biggest linking hub, as seen in the diagram, is the DBpedia data set[3].

The LOD cloud is growing constantly because the LOD project is open, which means that anyone is able to contribute to the project by publishing data sets and interlinking them with already existing data sets depicted in the diagram [21]. In Figure 2.4, we illustrate how fast the size of the cloud has increased since the year 2007.

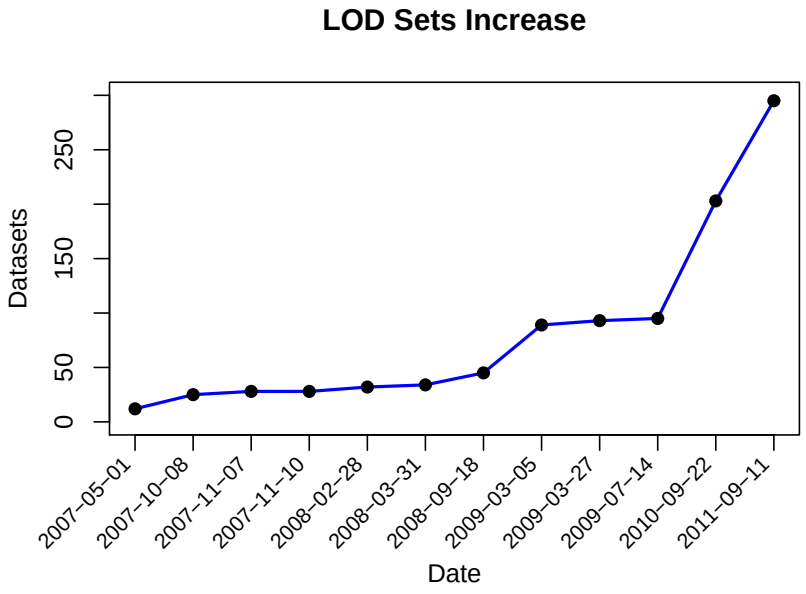


FIGURE 2.4: LOD sets increase from 2007 - 2011

In May 2007, there only existed 12 Linked Data sets. Since 2007 until 2010 the volume increased every year by more than a double. The last update happened in September 2011 which revealed a total of 259 data sets.

As the content of Linked Data on the Web increases year by year, the interest in creating tools which are making use of this content grows too. There already exist Linked Data browsers (navigate the information space), Linked Data search engines (crawl Linked Data from the Web by following RDF links) but also domain-specific Linked Data applications (exploit data

[3]DBpedia is further described in Section 2.2.1.

of specific data sets) [21]. Linked Data applications, designed for mobile phones, are yet a bit rare, but the interest in developing approaches in that kind of domain is growing. A noteworthy already published location-aware Linked Data browser is DBpedia Mobile[4] which visualizes DBpedia entities on a 2D map.

DBpedia Mobile

According to [34], DBpedia Mobile is a DBpedia client application for mobile phones that provides a map view on which the user can see her/his position and nearby DBpedia resources. It is realized as a client-server application whereby queries, data processing and formatting, and storage activities are performed completely on the server side by the *Marbles*[5] Linked Data engine. Marbles retrieves data from multiple sources and integrates it into a single graph. In addition, Marbles generates Fresnel[6]-based views to present the retrieved data. Thus, if a user selects a resource on the map, the server creates a Fresnel-based view that is displayed on the device and contains a description of the selected resource and RDF links into other Linked Data sets. In addition, DBpedia Mobile allows users to publish user-generated content and interlink it with nearby DBpedia resources.

2.2.1 Integrated LOD Sets

As seen within the LOD cloud digram (cf. Figure 2.3), there are many LOD sets available for accessing RDF data. In our proof-of-concept implementation, we use three different LOD sets (GeoNames, LinkedGeoData, and DBpedia) for fetching mountain specific data. We further describe them in the following sub-sections.

2.2.1.1 GeoNames

GeoNames represents a community project comprising a database which contains around 10 million names of geographical entities (e.g. countries, lakes) from all over the world. The official website of GeoNames [35] notes that data contents are organized through nine different feature classes to categorize geographical points of interest (mountains, lakes, cities etc.). There exist, as well, 645 feature codes for further sub-categorizing the data sets. The category "T" can be further sub-categorized in peak, rock, pass, mountain etc. In this work, the sub-categories could be used to display different groups of mountains in the AR interface. For instance, defining a filter to load mountains and rocks. Furthermore, resources of GeoNames can be freely accessed using one of the many available Web services (mainly REST-based) or by exporting the GeoNames database dump (text file) which is updated on a daily base.

An example RDF description file about a geographical entity, as obtained through one of the GeoNames Web services, can be accessed via the URI `http://sws.geonames.org/`

[4]DBpedia Mobile: `http://wiki.dbpedia.org/DBpediaMobile`
[5]Marbles Linked Data engine: `http://marbles.sourceforge.net/`
[6]Fresnel: browser-independent vocabulary for specifying how RDF-graphs are presented

`2774332/about.rdf`. The URI contains the code `2774332` which is a unique integer ID of that certain entity, and the ending `about.rdf` of the URI enables the loading of the content in RDF/XML format. We present the content of this file in Listing 2.1. It contains an RDF description about the Kitzsteinhorn[7]. This example file starts with the definition of prefixes. For instance, the prefix `gn` stands for the GeoNames ontology. The feature class and the feature code are defined in lines 12 and 13. They reveal that the entity of name `Kitzsteinhorn` is of class `T` and the feature code `MT` stands for the sub-category `mountain` in the GeoNames ontology

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <rdf:RDF xmlns:cc="http://creativecommons.org/ns#"
3           xmlns:foaf="http://xmlns.com/foaf/0.1/"
4           xmlns:gn="http://www.geonames.org/ontology#"
5           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
6           xmlns:wgs84_pos="http://www.w3.org/2003/01/geo/wgs84_pos#">
7   <gn:Feature rdf:about="http://sws.geonames.org/2774332/">
8    <rdfs:isDefinedBy>http://sws.geonames.org/2774332/about.rdf</rdfs:isDefinedBy>
9    <gn:name>Kitzsteinhorn</gn:name>
10   <gn:alternateName>Kitzstein</gn:alternateName>
11   <gn:alternateName>Kitzsteinhorn</gn:alternateName>
12   <gn:featureClass rdf:resource="http://www.geonames.org/ontology#T"/>
13   <gn:featureCode rdf:resource="http://www.geonames.org/ontology#T.MT"/>
14   <gn:countryCode>AT</gn:countryCode>
15   <wgs84_pos:lat>47.18805</wgs84_pos:lat>
16   <wgs84_pos:long>12.68751</wgs84_pos:long>
17   <wgs84_pos:alt>3209</wgs84_pos:alt>
18   <gn:wikipediaArticle rdf:resource="http://en.wikipedia.org/wiki/Kitzsteinhorn"/>
19   <rdfs:seeAlso rdf:resource="http://dbpedia.org/resource/Kitzsteinhorn"/>
20   ...
21  </gn:Feature>
22 </rdf:RDF>
```

Listing 2.1: GeoNames resource description in RDF/XML

2.2.1.2 LinkedGeoData

According to the official LinkedGeoData website [36], LinkedGeoData presents a large Linked Data spatial knowledge base that provides geographical data content collected from the OpenStreetMap[8] project as RDF triples and follows thereby the Linked Data principles. Currently, LinkedGeoData comprises about 350 million geographical nodes (longitude, latitude coordinates) and its RDF data content includes about 2 billion triples. Furthermore, LinkedGeoData provides multiple interlinks with DBpedia and GeoNames. The goal of this project is to add a spatial dimension to Linked Data. For instance, the URI for accessing the RDF description of the resource Kitzsteinhorn at LinkedGeoData can be accessed through `http:`

[7] Mountain located in Austria
[8] OpenStreetMap: http://www.openstreetmap.org/

//linkedgeodata.org/page/node36688518.rdf. Every entity notation within LinkedGeoData starts with the word node followed by an integer ID of that specific entity. We present the content of an example RDF file from LinkedGeoData in Listing 2.2. The LinkedGeoData ontology is presented by the prefix lgdo in this example. An entity which is a real-word mountain is defined as rdf:type of http://linkedgeodata.org/ontology/Peak.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3           xmlns:lgd="http://linkedgeodata.org/triplify/"
4           xmlns:lgdp="http://linkedgeodata.org/property/"
5           xmlns:owl="http://www.w3.org/2002/07/owl#"
6           xmlns:lgdo="http://linkedgeodata.org/ontology/"
7           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
8           xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
9           xmlns:georss="http://www.georss.org/georss/" >
10   <rdf:Description rdf:about="http://linkedgeodata.org/triplify/node36688518">
11     <owl:sameAs rdf:resource="http://sws.geonames.org/2774332/"/>
12     <lgdo:contributor rdf:resource="http://linkedgeodata.org/triplify/user22422"/>
13     <lgdp:created_by>Potlatch 0.7b</lgdp:created_by>
14     <lgdo:ele rdf:datatype="http://www.w3.org/2001/XMLSchema#float">3203</lgdo:ele>
15     <rdfs:label>Kitzsteinhorn</rdfs:label>
16     <rdf:type rdf:resource="http://linkedgeodata.org/ontology/Peak"/>
17     <georss:point>47.1872574 12.6859959</georss:point>
18     <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">12.6859959</
           geo:long>
19     <geo:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">47.1872574</geo:lat>
20   </rdf:Description>
21 </rdf:RDF>
```

Listing 2.2: LinkedGeoData resource description in RDF/XML

OpenStreetMap

OpenStreetMap (OSM) presents a collaborative project aiming to collect free geographical data to build an open modifiable map of the whole world [37]. Everyone has the chance to contribute to the project and also has the possibility to make use of the geographical data set under the terms of the Creative Commons Attribution-ShareAlike 2.0 (CC-BY-SA)[9] license. To be able to contribute to the project, one has to register at https://www.openstreetmap.org/user/new, so that it is possible to upload self-recorded GPS data or even whole GPS tracks. Moreover, registered users are allowed to modify incorrect data or enrich existing maps or vector data with further information or even render maps with one of OSM's provided editing tools [38]. As we intend to access GPS tracks within this work, we use OpenStreetMap we as the source for loading GPS tracks from the Web. The tracks are listed at http://www.openstreetmap.org/traces but unfortunately OpenStreetMap does not offer a Web service to be able to access tour data more easier. The listed tracks are described by different tags that a creator of a track determines during the uploading process. The tags allow to filter specific tracks.

[9]CC-BY-SA: http://creativecommons.org/licenses/by-sa/2.0/

However, all the tracks can only be listed within Web pages and you can only request tracks for one tag at a time.

2.2.1.3 DBpedia

DBpedia is a community project based on Linked Data principles that extracts data sets from Wikipedia and transforms them into RDF triples [39]. Auer et al. [39] outline that trough DBpedia it is possible to link Wikipedia contents with other data sets on the Web and the user has the possibility to perform sophisticated queries against these contents. Thus, DBpedia contents can easily be applied within Semantic Web applications. According to [40], the DBpedia version 3.7 contained descriptions of about 3.64 million things and about 1 billion pieces of information (RDF triples) in September 2011. The RDF description about the resource Kitzsteinhorn at DBpedia can be accessed via the URI `http://dbpedia.org/data/Kitzsteinhorn.rdf` (cf. Listing 2.3). The DBpedia ontology is presented by the prefix `dbpedia-owl`. An entity which presents a real-world mountain is defined as an entity of type `dbpedia-owl:Mountain`.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
4           xmlns:owl="http://www.w3.org/2002/07/owl#"
5           xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
6           xmlns:dbpedia-owl="http://dbpedia.org/ontology/">
7   <rdf:Description rdf:about="http://dbpedia.org/resource/Kitzsteinhorn">
8     <rdf:type rdf:resource="http://dbpedia.org/ontology/Place" />
9     <rdf:type rdf:resource="http://dbpedia.org/class/yago/MountainsOfTheAlps"/>
10    <rdf:type rdf:resource="http://dbpedia.org/ontology/Mountain" />
11    <owl:sameAs rdf:resource="http://dbpedia.org/resource/Kitzsteinhorn" />
12    <owl:sameAs rdf:resource="http://linkedgeodata.org/triplify/node/36688518#id"/>
13    <owl:sameAs rdf:resource="http://rdf.freebase.com/ns/m/05qwmq"/>
14    <owl:sameAs rdf:resource="http://sws.geonames.org/2774332/"/>
15    <rdfs:comment xml:lang="en">The Kitzsteinhorn is a mountain in the main chain of the
          Alps in the district of Kaprun, Salzburg, Austria. The Kitzsteinhorn is part of
          the Hohe Tauern range in the eastern Alps and reaches a height of 3203 m above
          sea level. It was first climbed in 1828 by Johann Entacher. The summit can be
          reached using the Kaprun Gletscherbahnen (Glacier Railways), from the valley
          station at 911 m.</rdfs:comment>
16    <geo:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#float">47.18805694580078</
          geo:lat>
17    <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#float">12.6875</geo:long>
18    <dbpedia-owl:mountainRange rdf:resource="http://dbpedia.org/resource/Hohe_Tauern"/>
19    <dbpedia-owl:elevation rdf:datatype="http://www.w3.org/2001/XMLSchema#double">3203</
          dbpedia-owl:elevation>
20  </rdf:Description>
21 </rdf:RDF>
```

Listing 2.3: DBpedia resource description in RDF/XML

2.2.2 Resource Description Framework

The Resource Description Framework (RDF) is a W3C standardized model for data interchange on the Web [41]. The data model is based on a directed graph and its data content are statements about resources. A statement represents a triple which consists of the following three parts [2]:

- **Subject:** an URI identifying the described resource about which the statement is made
- **Predicate:** exhibits the kind of relationship that exists between subject and object and is also identified by an URI
- **Object:** represents a certain kind of property value and presents a literal value (string, date etc.) or an URI

Figure 2.5 illustrates an example graph comprising two RDF statements. Subjects and objects are the nodes within an RDF graph. The subject within this example presents the URI `http://dbpedia.org/resource/Kitzsteinhorn` and is presented as an ellipse. The arcs are the predicates directed from the subject node to the object node. Object nodes which represent an URI are modeled as an ellipse, but objects which are literals are modeled as boxes.

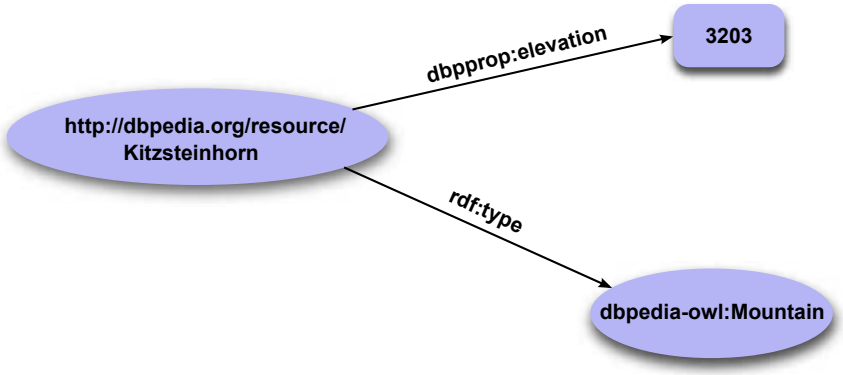


FIGURE 2.5: RDF graph example

Listing 2.4 depicts the RDF/XML syntax corresponding to the graph in Figure 2.5.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntaxns#" xmlns:dbpprop="http://
       dbpedia.org/property/">
3   <rdf:Description rdf:about="http://dbpedia.org/resource/Kitzsteinhorn">
4       <dbpprop:elevation xmlns:dbpprop="http://dbpedia.org/property/" rdf:datatype="
            http://www.w3.org/2001/XMLSchema#integer">3203</dbpprop:elevation>
5   </rdf:Description>
6   <rdf:Description rdf:about="http://dbpedia.org/resource/Kitzsteinhorn">
7       <rdf:type rdf:resource="http://dbpedia.org/ontology/Mountain"/>
8   </rdf:Description>
9  </rdf:RDF>
```

LISTING 2.4: Example of RDF statement in RDF/XML format

An RDF/XML-based file starts with a root node named `rdf:RDF`, the XML declaration and the necessary XML namespaces. The example in Listing 2.4 contains two statements. The subjects of the statements are `rdf:Description` nodes which include an `rdf:about` attribute that denotes the name of the subject through its URI. The `rdf:Description` tags enclose the predicates of the statements with the objects as their value. In line 4 the object presents a literal value and in line 7 we give an example of an object that presents an URI.

However, RDF does not provide the possibility to define classes of things in the world and properties to describe relations between them [42]. Instead, vocabularies like the RDF vocabulary description language - RDF Schema Specification (RDFS) [10] and the Web Ontology Language (OWL)[11] can be applied. They are more expressive languages for describing the syntax and semantics of vocabularies in a machine-understandable way [42].

Using RDF as data model in the LOD context, leads to following benefits [2]:

- RDF is designed to be used globally, so that anybody is able to refer to anything. This is possible through the utilization of globally unique identifiers namely HTTP URIs.
- Each RDF triple can present a starting point for exploring the huge data space on the Web. Any URI in RDF graphs can be accessed to retrieve information. As a result, additional information for each resource can be easily discovered by following the provided links.
- The RDF data model allows to mix and combine terms from different vocabularies for the description of resources. Vocabularies sometimes do not comprise all terms to sufficiently describe an entity. For instance, DBpedia entity descriptions mostly also contain terms of the FOAF (Friend-of-a-Friend) Vocabulary Specification[12], a vocabulary used to present data about people, or W3C'S Basic Geo Vocabulary[13], a vocabulary for representing latitude, longitude and altitude information in the WGS84 geodetic reference datum.
- The RDF data model and the schema languages RDFS and OWL together are expressive enough to present data as tightly or semi-structured as needed.

.

2.2.3 SPARQL

SPARQL is a W3C Candidate Recommendation and represents a query language for RDF [43]. SPARQL not only presents a query language but also a protocol for data access so that all RDF data sources can be queried [44]. A SPARQL query specifies patterns within the data which have to match to receive results. These patterns present triple patterns just like RDF triples, whereby in the query each part of the triple can be a variable. For instance, the

[10]RDF Schema Specification: `http://www.w3.org/TR/2000/CR-rdf-schema-20000327/`
[11]Web Ontology Language: `http://www.w3.org/TR/owl-features/`
[12]FOAF Vocabulary Specification: `http://xmlns.com/foaf/spec/`
[13]W3C Basic Geo Vocabulary: `http://www.w3.org/2003/01/geo/`

example presented in Listing 2.5 contains the variable `?movie` which presents the subject of an RDF triple. The results of such SPARQL queries can be result sets (triples) that matched the determined pattern or RDF graphs.

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX movies: <http://www.example.com/movies#>
3  SELECT ?movie
4  WHERE {
5  ?movie rdf:type movies:Movie .
6  ?movie movies:Director "Stephen King" .
7  }
8  ORDER BY ?movie
```

Listing 2.5: SPARQL query example

In the case of Listing 2.5, we specify two triple patterns within the `WHERE` clause. Variables are indicated by an interrogation mark (e.g. `?movie`). Furthermore, in SPARQL it is possible to build prefixes. For instance, the prefix `rdf` stands for the URI `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. Generally, URI items are enclosed in angle brackets and literals are enclosed in apostrophes within SPARQL queries. In line 5 of Listing 2.5, we define that the type of the resources that we request, has to be of the class `Movie`. In line 6 we specify that only movies of which the director is named `Stephen King` should be returned. Finally, we define that results (which will be URIs of all the resources that match the two patterns in line 5 and 6) have to be in alphabetic order (line).

2.3 Global Positioning

Global positioning plays an important role in this work because in our approach location-based data is used to send spatial queries against LOD sets in order to fetch data about real-world objects that are located in the near vicinity of the device. Thus, within the following section, we give a brief overview of the scientific discipline Geodesy and we also explain details of the Global Positioning System.

2.3.1 Geodesy

Geodesy is a scientific discipline with the purpose of surveying the earth's shape and size and of studying its gravity field [45]. For instance, Geodesy approaches the positioning of points on the surface of the earth. Thereby, it uses different coordinate systems. The most common way of measuring a geographical location on the globe is with the use of the geographical coordinate system. The measures that are used in the geographical coordinate system are called longitude and latitude (measure in degrees).

To understand latitude and longitude, one can imagine a globe covered full of vertical and horizontal circles (cf. Figure 2.6). The longest circle horizontally is the equator whose

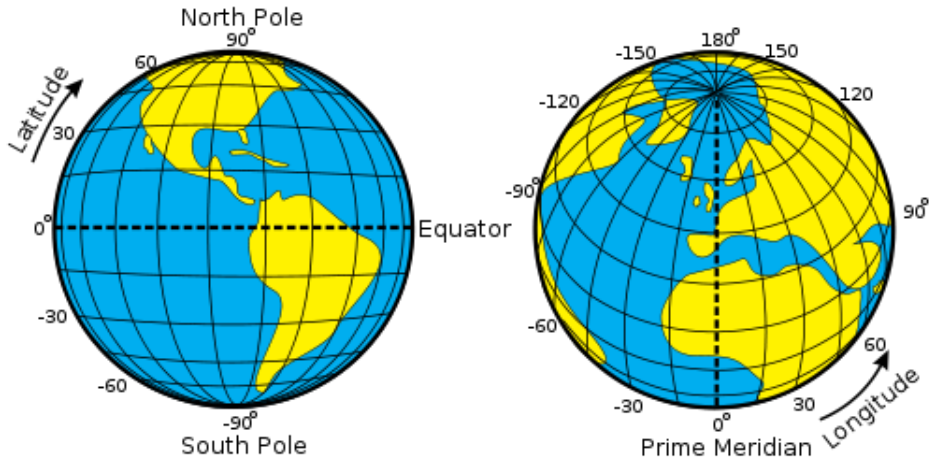


FIGURE 2.6: Illustration of geographic latitude and longitude of the earth [46]

latitude is 0°. The range of latitude values reaches from -90° to 90° (positive values in northern hemisphere, negative values in southern hemisphere). The lines of longitude cross the latitude lines and meet at the poles. Measurements of longitude start at the Prime Meridian (line through Greenwich England) with 0° and proceed with negative values till -180° towards west and positive values up to 180° towards east [47].

2.3.2 Global Positioning System

The Global Positioning System (GPS) is part of an existing satellite-based navigation system also known as Navigation System for Timing and Ranging (NAVSTAR) which was founded by The United States Department of Defense (USDOD) and can be used to gather signals with a GPS receiver for determining current positions on the earth's surface [48]. For instance, GPS finds usage in navigation on land, air, and sea, as well as, in surveying the land and generating precise geographical maps. The GPS system can be divided into three separate segments called space, control, and user [49]:

Space Segment:
This segment represents the GPS satellites and GPS orbits around the globe. There are 24 satellites (and some more representing a substitute in case others won't work anymore) which are circling around the globe on GPS orbits. Thereby, the orbits are tilted in an angle of 55° relative to the equator and they are shifted in multiples of 60° against each other. Each orbit contains four or more satellites which entails that at least four GPS satellites can be used around the clock from most points on the earth, to gather signals for determining the current position on earth. The satellites are equipped with a cesium atomic clock which is very accurate and used to extend the sending signals with timing information [48]. According to [50], the signals (sent by the GPS satellites) are two L-band carrier frequencies (L-band = 1,000 - 2,000 MHz) called L1 and L2. The two carrier frequencies are modulated by two pseudo-random noise (PRN) ranging codes, called coarse/acquisition (C/A) code and precision (P) code which are

illustrated in Figure 2.7. The C/A-code exhibits a 1,023 MHz chip rate (1 ms period) and enables, for instance, accurate range measurements and simultaneous range measurements from multiple satellites [48].

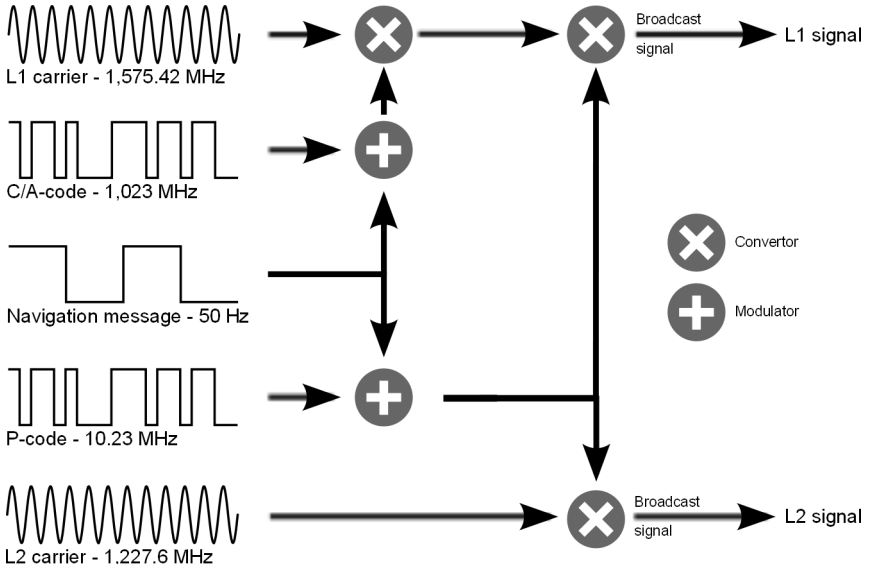


FIGURE 2.7: GPS signal modulation[14]

The C/A-code is carried only by the L1 signal which has a frequency of 1,575.42 MHz. In Figure 2.7, the C/A-code is modulated onto the L1 carrier. The figure also illustrated that the P-code can be modulated onto both L1 and L2 carriers, whereby it has a rate of 10.23 MHz (7 days period) and presents the navigation ranging code [50]. The L2 signal (frequency of 1,227.6 MHz) is only used by military applications and allows amongst other properties a more precise global positioning. In contrast, the L1 signal is freely available to the public.

In addition, each satellite sends a navigational message which comprises, for instance, the system time, clock behavior, the *almanac* data (enables the calculation of the location of every satellite within the GPS constellation - valid for several months) or the *ephemeris* data (enables more precise calculation of satellite position for user position estimation - only valid for a few hours) and presents a 50 Hz signal [48]. When transmitted, the navigational message is added either to the C/A-code or the P-code as seen in Figure 2.7.

Control Segment:

The U.S. military owns several ground stations that are located around the world for controlling, tracking, and observing the GPS satellites. Six of them present monitor stations without employed staff that continuously gather navigation signals of the GPS satellites [52]. These monitored data are then transmitted to a master control station (in Colorado, U.S.) where the data are analyzed. Thus, it is possible that the master station makes adjustments in case the signals are not accurate enough anymore. Therefore, the control segment also comprises four large ground-antenna stations which are responsible for constantly updating the GPS satellites (e.g. updating orbiting position) [49].

[14]Based on figure of [51]

User Segment:
The user segment presents a GPS receiver. Most smartphones are already equipped with a built-in GPS receiver. A receiver has the ability to receive, analyze, and elaborate on GPS satellite signals to determine the current position on earth. A GPS receiver needs two different information to be able to calculate a geographic position [53]:

- **Location of 3 or more satellites:** As earlier mentioned, the GPS satellites are continually sending a navigational message to the GPS receivers which contains among other information the almanac data or the ephemeris data. The almanac data include information about the orbits of all GPS satellites, so it is possible to determine the position of a satellite for a certain time. Ephemeris is the measured deviation from the orbit of a GPS satellite and is used for a more precise determination of the actual position of a satellite. As a result, almanac and ephemeris data can be used to calculate a precise position of a GPS satellite for a given time [53].

- **Distance to the satellites:** Transmitted signals of the satellites are traveling by speed of light to the ground of the earth. The distance is gained by the time the signal needed to arrive at the receiver multiplied by its speed (cf. Equation 2.1).

$$Distance = TransitTime * SpeedofLight \tag{2.1}$$

After the determination of the geographical location of 3 different GPS satellites and the distance of these satellites to the used GPS receiver, the process of trilateration is applied to calculate the actual position on earth. The known information (3 geo-points and the distances to the GPS receiver) is used to built 3 spheres (cf. Figure 2.8).

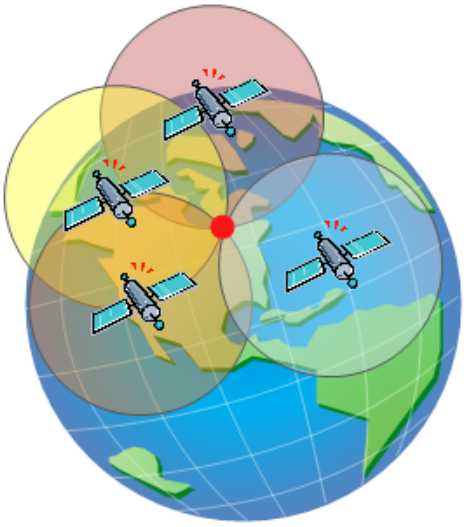


FIGURE 2.8: Geographic position determination with four satellites

The current possible GPS position lies on the intersections of these spheres and the possible locations on earth is narrowed down to two. According to [53], it is impossible to synchronize the

clock of a GPS receiver with the expensive and precise atomic clocks of the GPS satellites. Thus, the two location points, determined by using only three satellites, would be too inaccurate. In order to calculate a more precise position on earth, the GPS receiver uses instead of three four satellites (cf. Figure 2.8).

2.4 Android

Android presents a set of software modules including an operating system, middleware, and several key applications designed for mobile devices. It is led by Google[15] and was originally developed by the Open Handset Alliance[16]. Moreover, Android is an open source development platform, which means that a developer has complete access to the Android framework API. Thus, users have the possibility to create their own applications and reuse the core components of the Android architecture [54]. That is also one of the reasons why we chose to develop our proof-of-concept AR prototype based on the Android mobile operating system.

2.4.1 Android Architecture

The Architecture of Android consists of several layers which are visible in Figure 2.9 and further described in the following paragraphs.

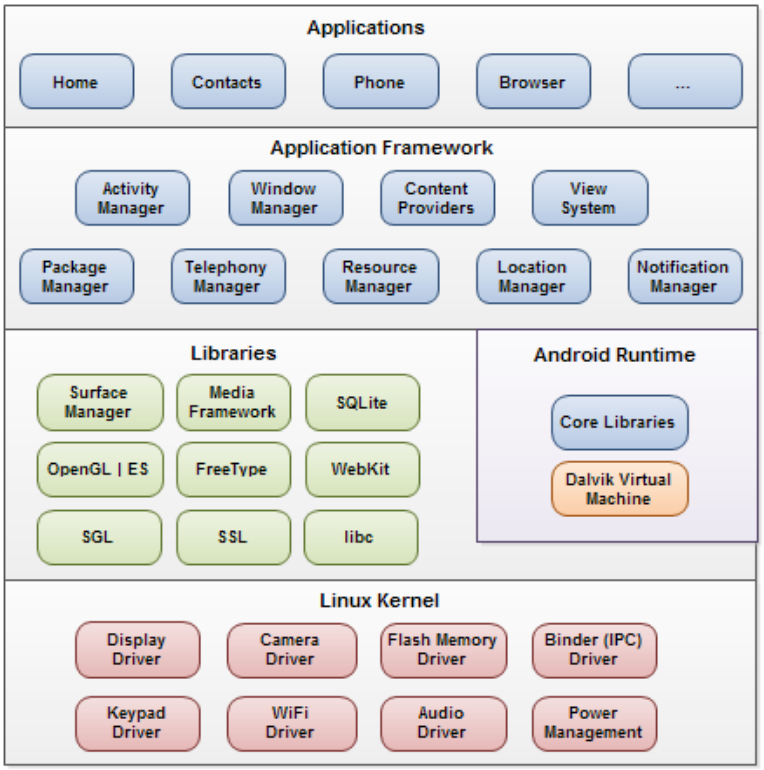


FIGURE 2.9: System architecture of Android[17]

[15]Google: `http://www.google.com`
[16]Open Handset Alliance: `http://www.openhandsetalliance.com/`
[17]Based on figure of [55]

Applications

The first layer contains applications. Android comes with various core applications like calendars, e-mail programs or Internet browsers. An average user mostly makes use of this layer [54].

Application Framework

As already mentioned above, the user has the possibility to access the Android framework API. Beneath all core applications of Android lies a group of systems and services which include for instance a set of Views (used to build a user friendly UI), Content Providers (enable to access data of other applications), a Resource Manager (enables access to objects such as graphics or layout files), a Notification Manager (for displaying notifications in the status bar) and an Activity Manager (life-cycle management, provides navigation back-stack). With the reuse of all these core components of the framework API the user can create a new fully executable Android application, which does not differ from the phone's core applications, because every application can be built to have the same access to a device's capabilities [54].

Libraries

Several components of Android are based on *C/C++* libraries. Some of the core libraries are listed in Figure 2.9 [54].

Android Runtime

Android programs are based on Java code. In consequence, Android contains several core libraries of the Java programming language. Furthermore, the architecture also contains a Dalvik Virtual Machine (VM) in this layer. If an Android application is executed, it runs with its own instance of Dalvik VM and in its own process [54].

Linux Kernel

The whole architecture of Android is built on top of a Linux Kernel which serves as some kind of abstraction layer between all the software components and the hardware components of an Android device [54].

2.4.2 Android SDK

The Android software development kit (Android SDK) presents a set of development tools for the creation of Android applications. The SDK is licensed under Creative Commons Attribution 2.5[18], freely available at the Android developer web site[19], and runs on most major OS platforms (eg. Windows[20], Mac OS X[21], or Linux[22]).

[18] Creative Commons Attribution 2.5: `http://creativecommons.org/licenses/by/2.5/`
[19] Android developer web site: `http://developer.android.com/sdk/index.html`
[20] Windows: `http://www.microsoft.com/`
[21] Mac OS X: `http://www.apple.com/osx/`
[22] Linux: `www.linux.org/`

The SDK contains the Android platform, many tools, libraries, emulators, sample code, and a complete documentation for creating Android apps. As Android applications are written in Java, the Android SDK comprises an add-on to the Java Development Kit[23] (JDK). There is also a plug-in for the software development environment Eclipse[24] freely available [56].

2.4.3 Android Application Components

As we present code snippets of the AR prototype within Chapter 5, we also describe the basic components of an Android application in the following paragraphs. This is necessary to follow the implementation explanations in Chapter 5.

Activities

An Android *activity* presents the presentation layer of an application [57]. An activity corresponds to one window in which the UI is drawn and presented on the screen, so that the user has the possibility to interact with the phone. Some examples of such interactions are for instance the sending of messages, the taking of photos, and the viewing of photos in a gallery. Android applications typically consist of several activities [58]. Each Android application comprises a single main activity. The main activity provides the graphical interface which is displayed when the application launches and thus presents the entry point to the application. Activities can start other activities and they are organized in some kind of a back-stack hierarchy. The current running activity is always on top of the stack. If a new one is started the old one is paused and moved back into the stack of the system. If the back-key on the device is pushed, the old activity in the back-stack is resumed and comes again on top of the stack [58].

Android applications are not destroyed and restarted very often, instead they are paused and resumed many times. An Android application is designed to run in its own process as long as possible, while its components are listening to state changes so that the application can react as appropriate [58]. In Figure 2.10, we present the life-cycle events of an Android application and the different kind of states an activity can transition into.

If a new activity is created, the system activates the `onCreate()` method. This method is mostly used to set up the UI of the application. The triggering of `onCreate()` also causes the starting of the `onStart()` and `onResume()` methods [59]. In case, the system puts an activity back into the back-stack (because the user moves to another activity), the `onPause()` method is activated and thus the new state of the activity changes to paused. If the activity is in the pause state, the system is able to use two methods to change the state of an activity: the `onResume()` method (if the activity is brought to the front again) or the `onStop()` method. If resources had been released in the `onPause()` method, they should be acquired again in the `onResume()` method [59]. The moment the activity becomes invisible for the user, the

[23] Java Development Kit: `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

[24] Eclipse: `http://www.eclipse.org/`

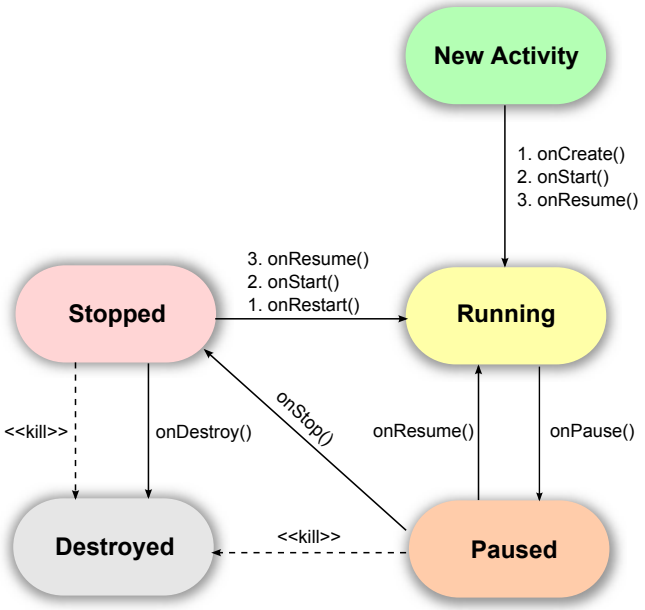


FIGURE 2.10: Application life-cycle

system calls the `onStop()` method. However, after the system has stopped an activity, it can re-activate this activity by calling the `onRestart()` method (followed by the `onStart()` and `onResume()` methods) [59]. Sometimes, if an activity is in the back-stack, the Android system decides to kill the activity. In this case, the `onDestroy()` method is triggered.

Not all of this state changing methods are used inside every Android application. Most of the time, developers are only defining `onCreate()`, `onPause()`, and `onResume()` methods within their Android projects [56].

Intents

Intents are used to send messages through the system. Therefore, they present asynchronous communication mechanisms between the components of the Android system. Intents can also be used to start Activities or to bind services and they present an object of passive data structure which holds the information about the operation which has to be performed or of the actions that took place and which were broad-casted across the system [58]. Moreover, intents can be *explicit* or *implicit* [56]:

Explicit: The intent specifies the class that has to be loaded.

Implicit: The intent does not specify a target class name but defines an action which should be performed.

Services

Services do not include an UI such as an activity. They present possible tasks which can be executed in the background of activities. The system activates or stops a service within an

activity through intents. As long as the service is running, the application components can bind to it [57]. As services perform processes in the background, they are able to ensure that the system continues to execute certain tasks in the background, even if the application's activities are not visible or active any more. Thus, it is possible that applications, which are not in an active state, are able to continue to respond to events through the use of an Android service [57].

2.4.4 Sensor Support

Our proof-of-concept AR prototype requires the access of several device sensors for making the AR tracking and registering of virtual objects with real-world objects possible. Most smartphones, available on the market, are equipped with sensors that are able to detect physical and environmental properties. For instance, the Android 2.3 version supports multiple kind of sensor types which we list within Table 2.1 [60].

Sensor Type	**Usage**
accelerometer	measures acceleration in m/s^2
gravity	indicates the direction and magnitude of gravity in m/s^2
gyroscope	measures orientation - the rate of rotation around the device's local axis ($< x, y, z >$)
magnetic field	measures magnetic field in micro-Tesla
light	measures ambient light in lux
proximity	measures distance of blocking object in centimeters
temperature	measures temperature in degrees Celsius
rotation vector	represents orientation of device as combination of angle and axis in which the device rotates through an angle θ around an axis $< x, y, z >$
pressure	measures air pressure in kilo pascals

Table 2.1: Sensors supported by Android

In the course of this thesis, we only make use of the *magnetic field* and *accelerometer* sensors to obtain the phones orientation and direction information constantly:

The *accelerometer* enables the measurement of acceleration along three directional axes lateral (left-right), longitudinal (forward-backward), and vertical (up-down). The acceleration is used to measure movement and is defined as the rate of change of velocity. Unfortunately, velocity is not measured thus speed is directly measured by determining acceleration changes over time [57].

The *magnetic field* sensor measures ambient magnetic field in micro-Tesla along the x-, y-, and z-axes and is also known as compass because this sensor is able to detect the earth's magnetic field and thus indicates where north is [56]. In Section 5.2.1.2, we further explain the implementation of those two sensor types within our AR prototype.

2.5 Summary

AR systems need to perform the process of tracking in order to permanently gather the current field of view of the device, to be able to identify nearby location points. Registration is also a necessary process of an AR system because the field of view of the device is changing on occasion and yet virtual and real objects have to stay aligned, otherwise the AR system will not be accepted by the user. Thus, our approach has to include proper methods for tracking and registration. We discuss different tracking approaches in Chapter 3.

The LOD initiative already comprises a huge amount of data sets that can be accessed easily with specific Web services and SPARQL endpoints. The amount of interlinked data is growing year by year. Especially, LinkedGeoData and GeoNames offer geo-spatial data sets that could be deployed within AR systems to visualize nearby location points in an AR interface. RDF presents a rich data model and SPARQL presents a suitable query language for fetching geo-spatial data from LOD sets. In addition, DBpedia presents a potential example for being deployed as data provider in our AR prototype implementation, for the purpose of retrieving additional information about already visualized resources. We discuss the most relevant aspects of integrating and exploiting LOD resources within our AR system in Chapter 4 and we present the realization of our approach design in Chapter 5.

At last, we gave an overview about the Android operating system because we will demonstrate the feasibility of our approach presented in Chapter 4 through a prototypical implementation of a mobile AR application for Android devices. We use Android as operating system due to the free and easy access to all the necessary features (such as digital camera, GPS radio, sensor data etc.) which are essential in order to build a proper mobile AR application.

Chapter 3

Related Work

In this chapter, we describe the most relevant related works and accomplish a state-of-the-art analysis of current sensor-based mobile AR systems. We already explained in Section 2.1, that every accurate AR system needs to implement methods for tracking and registration. In consequence, we discuss existing AR tracking approaches in the first part of this chapter. Then, we point out that we focus on a sensor-based tracking approach in this work. Therefore, we continue with a state-of-the-art analysis of currently existing AR applications that are based on sensor-based AR tracking systems. In the last part of this chapter, we present these introduced sensor-based AR applications within a table (Table 3.1) and compare specific features each of the applications exhibits.

3.1 AR Tracking Methods

An AR system has to comprise an accurate registration between real and computer-generated objects [29]. If users move their heads and change their viewpoint, visualized objects have to stay aligned with the three-dimensional real-world objects. To assure alignment, our AR system requires a proper tracking method. Then, it is able to ensure a precise estimation of the user's real-world *viewing pose* (we describe viewing pose in Definition 3.1 according to the definition provided by [61]) with respect to the coordinate system of the visualized objects [61].

> **Definition 3.1** (Viewing Pose, 6DOF)**.** *The viewing pose is a six-degree of freedom (6DOF) measurement: three degrees of freedom for position and three for orientation. The tracked viewing pose defines the projection of 3-D graphics into the real-world image so tracking accuracy determines the accuracy of alignment.* (Neumann and You, 1999)

We already explained in Chapter 2, that there exist marker-based and marker-less tracking methods (cf. Definition 2.1, Definition 2.2). In this thesis, we focus on marker-less tracking methods for mobile AR systems. In consideration, that there are many different distinctions of

marker-less AR tracking methods (cf. [27]), we separate them roughly into vision-based, sensor-based, and hybrid tracking methods and we present existing approaches for each different type of tracking method in the following sub-sections. Thereby, we chose the most relevant works of the last view years and we only mention those approaches which are deployable outdoors and follow the principle of *inside-out tracking* (cf. Definition 3.2 based on [62], [63]). Furthermore, all presented approaches are executable on hand-held devices (mobile phones, PDAs or tablet PCs). As we put focus on sensor-based tracking in our work, we further discuss sensor-based AR systems in Section 3.2.

> **Definition 3.2** (Inside-out/Outside-in Tracking)**.** *Inside-out tracking implies that the observer is moving and landmarks are fixed in the environment. The used camera device is equipped with sensors to estimate position and orientation constantly and a relationship between camera coordinates and world coordinates has to be determined. In contrast, outside-in tracking is based on stationary observing of a scene. Thus, the camera is stationary and the relationship between camera and world coordinates is fixed. Landmarks (e.g. LEDs) are placed on the moving object (e.g. user) in order to be tracked.*

3.1.1 Vision-based Tracking

Marker-less vision-based tracking systems rely on image pattern recognition and natural feature-detection processes to be able to estimate the camera position relative to the real-world objects. Natural features are, for instance, points, lines, edges or textures. A vision-based tracking system is able to provide higher accuracy than sensor-based systems due to error-prone sensor measurements [64].

However, mobile phones are less powerful computing platforms and exhibit limited processing power. Thus, it is difficult to apply natural feature-detection in mobile AR systems, as this process is complex and resource consuming [65]. This is the reason why fully vision-based tracking systems are not very common for outdoor mobile AR applications. Nevertheless, we present two approaches in the following paragraphs:

Wagner et al. [65] present a natural feature tracking system that runs in real-time on mobile phones and is capable of tracking full 6 degrees of freedom (6DOF) (cf. Definition 3.1) from beforehand known textured planar targets. Their approach is based on existing feature descriptors called SIFT (Scale-invariant feature transform) and Ferns. These are algorithms to detect and describe local features in images. Wagner et al. modified both algorithms (SIFT, Ferns) to make them suitable for mobile phones. They call the modified tracking techniques PhonySIFT and PhonyFerns. These modified algorithms are able to detect a target and estimate pose on a mobile device. An additional template-based tracker, called PatchTracker, increases the performances of PhonySIFT and PhonyFerns. The PatchTracker uses the estimated pose of the other trackers as starting pose to estimate a pose of a new frame and uses this pose for

continuous tracking in the following frames. Their approach runs at frame rate, relies on known planar targets, and is not suitable for being execute in unknown environments.

Klein and Murray [66] developed a keyframe-based SLAM (Simultaneous Localization and Mapping) system called PTAM (Parallel Mapping and Tracking) suitable for mobile phones which is able to parallel track and map in unknown environments. Their system allows to map features from the environment at runtime and it performs processes on two threads: one thread manages camera frame tracking and camera registration. In the second thread, their system optimizes the map by performing *bundle adjustment* (we describe bundle adjustment in Definition 3.3 according to the definition provided by [67]). Klein and Murray's approach runs on an iPhone and allows the creation and supplementation of small maps. However, Wagner et al. claims in [68] that their approach is limited to a small amount of key-points in a map because of the low processing power of mobile phones.

> **Definition 3.3** (Bundle Adjustment)**.** *Bundle adjustment is the problem of refining a visual reconstruction to produce jointly optimal 3D structure and viewing parameter (camera pose and/or calibration) estimates. Optimal means that the parameter estimates are found by minimizing some cost function that quantifies the model fitting error, and jointly that the solution is simultaneously optimal with respect to both structure and camera variations. The name refers to the bundles of light rays leaving each 3D feature and converging on each camera centre, which are adjusted optimally with respect to both feature and camera positions.* (Triggs et al., 2000)

3.1.2 Sensor-based Tracking

AR systems that comprise a sensor-based tracking system rely on a device's built-in GPS radio, digital compass (magnetometer), and accelerometer but also on the geographic coordinates of the real-world objects to be tracked. Through the determination of the current position on earth (with the help of the GPS radio), it is possible to load information about resources for a specific area near the device. The AR system uses sensor data measured by the digital compass and accelerometer of the hand-held device to estimate the current camera pose (cf. Definition 3.1). Additionally, such sensor-based AR system comprise a mathematical model which is able to project virtual objects into the AR interface (that captures the current scene of the real-world) of the AR system. The model computes projections based on the estimated camera pose and the geographical information of local real-world objects. This means that the mathematical model is able to calculate screen coordinates for a virtual object to be placed upon the real-world objects and to augment in this way the current captured real-world scene with digital information.

Kähäri and Murphy [69] developed an AR system called MARA[1] (Mobile Augmented Reality Applications) that presents one of the first sensor-based AR browsers running on mobile

[1]More information about the MARA project: `http://research.nokia.com/page/219/`

phones. The system estimates current position and orientation of the device and supplements the current camera view with virtual annotations. They fetch data for the annotations from external Internet services to be able to determine if real-world objects are captured by the device's camera. Once real-world objects are visible, the system displays digital information and provides hyperlinks (if available) to navigate to further respective information about the selected visible object [69]. Their project also allows users to annotate locations with hyperlinks to add text which can later be detected by other users.

The Wikitude World Browser and the Layar Reality Browser also rely on a fully sensor-based tracking system. We further describe and compare Wikitude, Layar, and additional sensor-based approaches in Section 3.2.

3.1.3 Hybrid Tracking

Hybrid tracking technologies mix methods of both vision and sensor-based tracking. Therefore, measurements of vision-based and sensor-based tracking methods are fused to provide a more robust tracking solution.

Takacs et al. [70] present an outdoor mobile augmented reality system which relies on image matching through landmark recognition. Their system matches camera-phone images against a database of geo-tagged images (images associated with GPS locations and names of specific landmarks) by the use of an image retrieval algorithm. Their approach is client-server based. On the server side, the geo-tagged images are grouped by location into cells. Then, they extract features of images within a cell. The feature extraction process is additionally followed by a clustering algorithm that builds groups of geometrically consistent meta-features (meta-features which represent the most repeatable features in a cell). The generated feature descriptions are compressed and stored in files on the server side. In consequence, the server always keeps feature sets of the closest (to the mobile device location) location cells in memory. The mobile phone uses these feature files for image matching processes and caches some of these files on the local storage system. Thus, the mobile phone has to update the server with location information to obtain new feature descriptions (which are not already in the local cache), in the case the user moves to a new location. The client captures new images with the camera, extracts features, matches them against the cached features and performs also a geometric consistency check. The highest matching results lead to the displaying of information on the device's display. This approach presents a hybrid tracking method because GPS sensor data is combined with image matching processes. They tested their approach on a mobile phone where it runs close to real time and their system also achieved a high image matching rate [70].

Wagner et al. [68] describe a method for both outdoor and indoor localization that creates panoramic maps in real time on a mobile phone and allows 3DOF (two values x and y for position and one angle theta for orientation) tracking. Their approach includes, first of all, the projection of camera frames onto a cylindrical map. During the mapping keypoint features are

extracted. These extracted keypoint features are used during the tracking process. They track keypoints from one frame to the following, so keypoints in the camera image are matched against their counterpart in the map. After the matching process, the system updates the orientation and camera frames are again projected into the map space. The system always extends the map with new features from viewing directions that have not been observed before. Their method generates color and grayscale panoramas of the surrounding environment. It is possible to later load these maps from the device's storage, in order to start future initializations from existing maps. If no map exists at the beginning, the tracking method relies on the orientation and direction of the first frame that is fetched from the phone's sensors (accelerometer and digital compass).

Langlotz et al. [64] present an enhanced AR system for tracking and registering local POIs which is suitable for mobile phones and therefore operates in real time. The technique of generating panoramic maps proposed in [68] and a later extended work which allows annotating these maps presented in [71], present the foundation of the tracking system of Langlotz et al. which also includes the creation of panoramic maps on which local POIs are annotated. This work focuses on improving the low re-detection rate of annotation on the map under different environmental conditions (e.g. lightening). Therefore, they made three improvements [64]:

- Enhancement of the quality of panoramic maps through the help of an extended dynamic range representation of the map.
- Improved exploitation of sensor features from currently available smartphones by applying sensor-based tracking to restrict the search area for the vision-based tracking.
- Enhancement of the global consistency of their tracking systems is obtained by statistical estimations. In that way, their system guarantees the reduction of the overall position deviation of visualized points during the projections from the points from the original panoramic map to the current captured panorama by the device.

They tested their system by implementing an AR campus guide application for mobile phones. Their enhanced approach increases the re-detection rate of accurate matches to 90% despite of significant lightening conditions [64].

The approach of Marimon et al. [72] presents a mechanism to fuse data of sensor measurements (digital compass and accelerometers) from a mobile phone with data of visual (image pattern) recognition methods. Thereby, they use an image recognition algorithm to identify geo-referenced images of a database that match with the current captured view of a device's camera. The sensor data is then used to estimate the most probable location. Their system is client-server based and performs all computations on the server side.

Reitmayr and Drummond [7] introduce a textured 3D model-based hybrid tracking system which enables accurate, real-time overlay visualizations for hand-held devices. Model-based tracking usually relies on specific features, such as, lines or edges. In their approach, they

use an edge-based tracker for estimating the camera pose in an urban outdoor environment by performing edge detection on a rendering of a 3D textured model from the current pose prediction. Sensor measurements (gyroscope measurements for fast motion, measurements of gravity and magnetic field to avoid drift) are then fused with the pose estimates with an *extended Kalman filter (EKF)* (we describe EKF in Definition 3.4 according to the definition provided by [73]). The 3D models which are used for tracking are created from photographs. Creating such 3D textured models for larger environments can be very laborious. Their approach was tested on a hand-held AR platform consisting of a tablet PC, an USB camera, and Inertial Measurement Unit (IMU).

> **Definition 3.4** (Kalman filter)**.** *The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown. [...] A Kalman filter that linearizes about the current mean and covariance is referred to as an extended Kalman filter or EKF.* (Welch and Bishop, 1995)

3.1.4 Discussion

In this work, we put the focus on sensor-based tracking methods because by applying visual tracking methods, only objects which are in the view-point of the device's camera can be captured, analyzed, and compared with images in a database. Objects which are covered by other objects can not be recorded by the camera and also not compared with reference images in a database.

In addition, an image database has to contain similar images to the ones which are captured by the device to even be able to recognize the captured real-world objects or to make the determination of the current position on earth possible in first place. Thus, the use of vision-based tracking enforces the availability of an image database (with images of the environment in which the tracking is performed). Estimating the camera pose in larger unconstrained environments is undue because efficiently finding a matching reference image in a database with a huge amount of images is very difficult [74].

As our approach is executable in outdoor environments, we would have to build a image database in advance that comprises image data about the environment in which the tracking will be executed. The extent of such a database would be enormous as the outdoor environment is not restricted. Furthermore, the access to an external database, to be able to execute image pattern algorithms, requires a permanent active connection between client and server side of the AR system. Thus, a vision-based tracking system is more suitable for indoor AR systems which require a image database of a smaller and constrained environment. A sensor-based tracking

system does not require a reference image database. It makes uses of the mobile device's built-in sensors to determine current position and orientation. As such a system uses the GPS radio of the device, sensor-based systems are suitable for outdoor execution.

Our approach includes the integration of Linked Data resources to AR systems. Specific Linked Data sets in the Web contain geographical data resource descriptions (e.g. latitude/longitude information of resources) which we use to identify local real-world objects (e.g. mountains) that appear in the captured field of view of a mobile device's camera (cf. Section 4.4.1). Additionally, the GPS location information of the mobile device narrows the amount of loaded geographical data from the LOD sets down to data for a specific defined range. In our approach, we do not require image matching processes, our approach is solely based on sensor-based methods. We further discuss our approach in Chapter 4.

3.2 Sensor-based AR Applications

As we focus on a sensor-based AR tracking approach in this work, we now discuss existing mobile AR applications that are based on fully sensor-based tracking systems.

3.2.1 Introduction of Works

In the next paragraphs, we categorize existing sensor-based AR applications according to the following categories: commercial AR applications, open-source AR applications, AR applications for identifying mountains, and other applications that integrate Linked Data resources.

3.2.1.1 Commercial AR Applications

The AR applications presented in this sub-section exhibit similar system architectures [75]: they comprise a main server which acts as handler between provider servers on which POI (cf. Definition 1.1) information is stored and an AR browser client running on the a mobile phone. Only the respective client application is able to communicate with the main server of the AR system.

Wikitude World Browser

The Wikitude World Browser is an AR browser for smartphones which enables the discovery of POIs on a live camera view. The Wikitude World Browser uses ARML[2] (Augmented Reality Markup Language) for mapping geo-referenced POIs and their metadata onto a mobile phone's display. In reference to [76], ARML is built on a subset of KML[3] (Keyhole Markup Language). Wikitude is the first browser supporting ARML. An ARML file contains entries which present POIs. Users are able to generate their own ARML file which can be displayed within the

[2]ARML: `http://www.openarml.org/`

[3]KML is a XML-based standard for geographical annotations on 2D maps or 3D Earth browsers.

Wikitude World Browser. The POIs of Wikitude are grouped into different Wikitude worlds. An example of a Wikitude world is Wikipedia, where the loaded POIs are based on data from Wikipedia. Each world presents a different data provider. Figure 3.1 illustrates a rough overview of the architecture of Wikitude. The client application is running on the device and makes a request (contains current location information) to the Wikitude server. The response to the client's request is an ARML file, which contains the search results (POIs) that have to be displayed for that certain location and orientation of the phone.

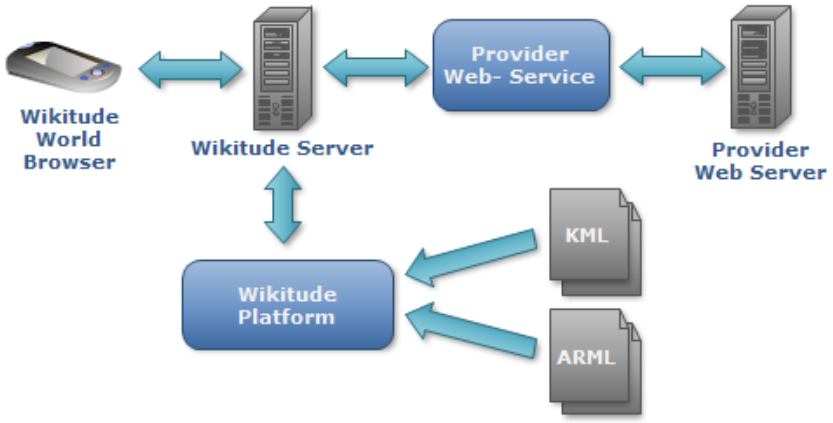


FIGURE 3.1: Overview of Wikitude architecture[4]

Figure 3.1 also illustrates that providers have the possibility to add their content by creating a Web service to communicate with their own Web server. The Wikitude server requests the provider's data by using HTTP GET requests [77]. Wikitude is able to process ARML files only for displaying POIs, so the data returned from the provider's Web service need to be in ARML format. The Wikitude platform offers a map-interface through which users are able to add POIs. If they want to add several POIs at once, they are also allowed to upload KML or ARML files at the Wikitude platform.

Layar Reality Browser

The dutch Layar company also offers an AR browser which augments a smartphone's camera view with digital information. POIs are grouped into layers, similar to a Wikitude world. There are several layers available to display different kinds of POIs (e.g. YouTube to display posted YouTube videos near the device) and everyone can subscribe to create their own layer.

Figure 3.2 provides an overview about the architecture of the Layar platform. The exact model structure is hidden from the user so a content developer is only able to create a layer through the developer API provided by Layar. At the Layar Web platform, the user is able to modify and publish layers. According to the developer documentation [78], a Layar service provider has to provide an API URL to its Web service and also a database where the POIs are stored. If the client API requests for POIs, then the Layar server forwards the request to

[4]Based on figure of [77]

the service provider. The `getPOIs` response, which presents the returned AR content, is later sent back to the client API. The client API finally displays the POIs on the display. The Layar browser allows JSON as response format. Layar does not use a standard format, such as ARML, for describing AR content.

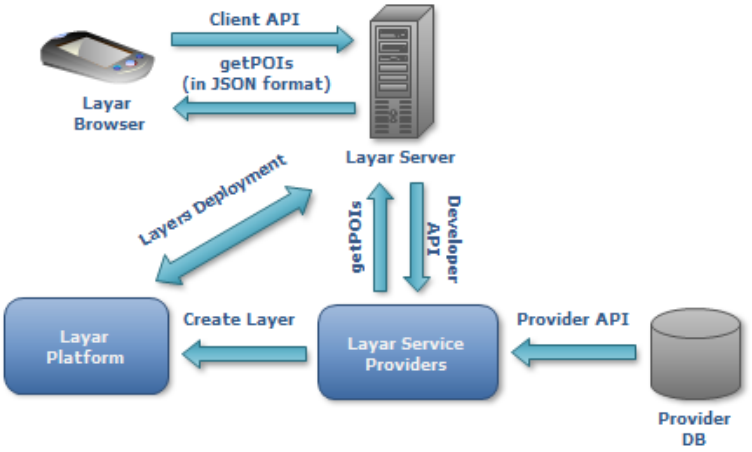


FIGURE 3.2: Overview of Layar architecture[5]

Sekai Camera

Sekai Camera[6] is a Japanese social AR application. This application includes social-sharing features and therefore users are able to post comments, images, videos, and audios in the physical environment which can later be found by other users who come across and make use of Sekai Camera [18]. The architectural infrastructure is closed and similar to that of Wikitude and Layar. The Sekai Camera system includes a client application running on a mobile device, a main server that communicates with the client application and extern POI providers, and the POI servers of the providers. Sekai Camera also uses JSON as response data format [79] but it does not use a standard format, such as ARML, for describing POIs.

3.2.1.2 Open-Source AR Applications

The AR applications which are available as open-source application provide their source code to the public and thus allow developers to generate various kinds of AR applications based on the provided AR framework.

[5]Based on figure of [78]
[6]Sekai Camera: `http://sekaicamera.com/`

Mixare

In contrast to Wikitude and Layar, Mixare[7] represents an open source AR platform. Mixare is downloadable as standalone launcher application (data is fetched from GeoNames, response format of POIs in JSON) or developers are able to load the source code of Mixare to customize an AR application that serves their needs.

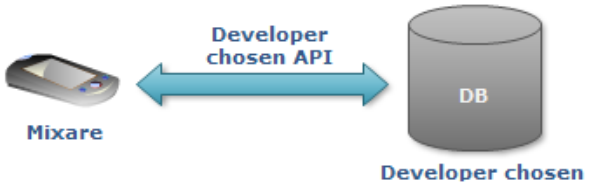


FIGURE 3.3: Architecture of Mixare for developers[8]

Figure 3.3 clarifies that the architecture of Mixare does not comprise a main server as within the architecture of Layar and Wikitude. The developer has the possibility to connect directly his/her own Web server with an implemented Web service and a database for the POIs to the client application. As developers have access to the source code, they can define which data format they prefer for the POI responses. Mixare does not support any AR formats.

LibreGeoSocial

LibreGeoSocial[9] presents a community project to build an open source framework for creating mobile AR social network applications [18]. There exists a standalone LibreGeoSocial mobile client for Android phones on the Android marked with which it is possible to obtain information about captured nearby real-world objects. The information can be discovered through an AR interface but also in form of lists or displayed on a 2D map.

Just like Wikitude and Layar, LibreGeoSocial allows to display different layers of geo-located content (e.g. nearby posted YouTube videos) provided by third parties. However, Wikitude and Layar require to push POI descriptions from the provider databases to their own platform, the LibreGeoSocial framework fetches data directly in real time from the provider databases [80]. LibreGeoSocial does not provide any kind of AR relevant data format.

Additionally, LibreGeoSocial offers social networking features: users can share content with other users by tagging objects, posting comments, images, audios or videos directly in the physical environment. Thus, LibreGeoSocial allows to link user-generated content with geo-location coordinates. In consequence, other users are able to detect content that was posted by other users. Sekai Camera, also provides the functionality to link media to different geo-locations but it is not possible to associate information or media with specific objects [80].

[7] Mixare: `http://www.mixare.org/`
[8] Based on figure of `http://www.mixare.org/usage/`
[9] LibreGeoSocial: `http://www.libregeosocial.org/`

As the LibreGeoSocial framework is freely available and open source, developers are able to load the source code and extend it with new functionalities or to build a completely new AR scenario, e.g., AR games, AR tourism guide, AR shopping guide etc. [81].

3.2.1.3 Mountain Identification Applications

As the proof-of-concept AR application of this work identifies mountains, we outline existing examples of such applications in the following paragraphs.

SwissPeaks

Karpischek et al. [82] presents an AR browser which runs on iPhones and which also enables to detect mountains through a mobiles phone's camera. SwissPeaks allows to display additional information about a certain mountain in form of Wikipedia articles. Furthermore, it identifies mountains by loading resources from GeoNames. Only mountains, which are in the user's viewpoint are displayed in an AR interface. In addition, the user is able to correct inaccurate sensor data with manual user input. There is no information about the architecture of SwissPeaks available to the public. It is also not known, if this AR system makes use of any kind of AR data format for describing AR content.

Peak.AR

Peak.AR[10] was developed by Salzburg Research[11] and presents a fully commercial AR browser for the identification of mountains. According to the official website of Peak.AR [83], the AR content is extracted from OpenStreetMap and loaded to the local storage system of the device during the first activation of the application. Thus, this application requires a single active network connection. If the user taps on a visualized mountain icon, the distance to the mountain, the height of the mountain, and the geographical coordinates are displayed on the screen. Requesting further information about specific mountains is not possible. Unfortunately, there is no further information about the architecture of this application available, so we do not know how data is fetched from OpenStreetMap or if they use a specific AR format for the AR content.

3.2.1.4 Linked Data as AR Content

One application which uses LOD resources within AR systems to identify real-world objects but also for the provision of additional information about certain POIs presents the previous mentioned SwissPeaks application presented in [82]. But Karpischek et al. [82] do not describe how

[10]Peak.AR: `http://peakar.salzburgresearch.at/`
[11]Salzburg Research: `http://www.salzburgresearch.at/`

they process the LOD resources which they use as AR content within their work. Another example that combines Linked Data principles with mobile AR systems presents the SmartReality[12] project, which we further describe below.

SmartReality

SmartReality presents an Austrian research project started in October 2010 which defines a SmartReality AR framework. This framework makes use of semantic and Web service technologies and combines them with AR technologies to provide information about local POIs in the user's physical environment [84]. A real-world object or simply a "thing" is called a *Thing of Interest (TOI)* which has a name and an unique identifier (URI). TOIs are described semantically and linked to the Web of Linked Data. Further meta-data about a TOI can be retrieved from other LOD sources.

The SmartReality project aims to support different types of application scenarios in order to encourage the development of various smart AR environments. Therefore, they propose to process data on a single platform so that different clients are able to access preprocessed data from the SmartReality platform. The platform will federate data from various LOD sets through the help of Web services and will manage semantic-based reasoning and aggregation of the different types of LOD resources [84].

However, the AR applications that will be developed based on the SmartReality framework will not present fully sensor-based AR system. Their attempt is to improve existing object tracking algorithms for mobile devices. They already presented an example AR application that is able to identify annotations of event posters and overlay them with virtual content on top of the posters such as a map of the event location or audio and video samples of artists mentioned on a poster.

We do not include SmartReality to the comparison table in Section 3.2.2 as it presents a project that currently is under development and the created client applications of this project also include object tracking algorithms. Thus, these applications are not comparable with the sensor-based AR applications that we list in Table 3.1.

3.2.2 Table of Discussed Works

In Table 3.1, we compare all the presented sensor-based AR browsers of Section 3.2.1.1 and Section 3.2.1.3. The last column of the table contains the AR system of our proof-of-concept implementation. We compare the applications on the base of specific selected features that we explain and discuss in Section 3.2.3.

[12]SmartReality: `http://www.smartreality.at/`

Projects / Attributes	**Wikitude World Browser**	**Layar Reality Browser**	**Sekai Camera**	**Mixare**	**LibreGeoSocial**	**Peak.AR**	**SwissPeaks**	**AR Mountain Guide**
License	freeware, proprietary	freeware, proprietary	freeware, proprietary	freeware, open source	freeware, open source	freeware, proprietary	freeware, proprietary	not available to the public
Federation Infrastructure	no	no	no	no	no	no	no	yes
LOD Processing	no	no	no	no	no	no	no	yes
Data Representation Standard	ARML, KML	-	-	-	-	-	-	-
Filters	yes	no	yes	yes	no	no	no	no
Social Networking Features	yes	no	yes	no	yes	no	no	no
AR Content Features	Infobox, Browsing, Multimedia, Map, Sharing, Route	Infobox, Browsing, Multimedia, Map, Sharing, Route	Infobox, Multimedia, Map, Sharing	Infobox, Browsing	Infobox, Multimedia, Map, Sharing, Route	Infobox	Infobox, Browsing	Infobox, Browsing
Platform	Android, BlackBerry OS[13], iOS[14], Symbian[15], bada[16], Windows Phone[17]	Android, BlackBerry OS, iOS, Symbian, bada	Android, iOS	Android, iOS	Android	Android, iOS	iOS	Android
Off-line Mode	no	no	no	no	no	yes	no	yes
Tour Guide Features	no	no	no	no	no	no	no	yes

TABLE 3.1: Comparison of sensor-based AR browsers

[13]BlackBerry OS: `http://us.blackberry.com/`
[14]iOS: `http://www.apple.com/ios/`
[15]Symbian: `http://www.developer.nokia.com/Devices/Symbian/`
[16]bada: `http://www.bada.com/`
[17]Windows Phone: `http://www.windowsphone.com/`

3.2.3 Feature Description and Discussion

First of all, we introduce every criteria within Table 3.1 by describing its meaning followed by a brief discussion to see how the criteria is realized in each of the selected projects.

License

We distinguish applications that are freeware or not available (e.g. only scientific approach) to the public. Some of the listed commercial AR browsers are proprietary software products (software is restricted in use). Thus, no modification and further distribution of the software is allowed. Some applications are open source. In consequence, the source code of these applications is freely available and users are able to load, modify, and extend the source code.

The AR browsers provided by Wikitude and Layar, as well as Sekai Camera, SwissPeaks, and Peak.AR are commercial applications. It is possible to download these applications for free but the user has no access to the source code. However, Wikitude and Layar provide developer APIs which allow to create overlays. Only Mixare and LibreGeoSocial are open source applications, so they can be used as an engine on which other apps can be built upon. We realized the AR Mountain Guide as a proof-of-concept prototype for this thesis and therefore this application is not available to the public at the moment.

Federation Infrastructure

This criteria exhibits if the infrastructure of the AR application includes the integration of data from multiple data sources, so that it possible to retrieve information about a single resource (POI) from several data sources.

Only our proof-of-concept prototype includes data federation. The AR Mountain Guide retrieves geographical data about mountains from two different data sources (GeoNames and LinkedGeoData). We describe our approach of federating and processing LOD resources in the next chapter in Section 4.3.

Wikitude, Layar, Mixare, LibreGeoSocial, and SekaiCamera allow it to display different groups of POIs (data from YouTube, Wikipedia etc.). The information about a single POI is retrieved from a single data source. For instance, if the user loads the YouTube channel at Wikitude, only information provided by the YouTube channel can be accessed for one resource. There is no federation of data included. POI descriptions are locked down in isolated repositories. The infrastructure of Peak.AR and SwissPeaks also do not integrate data federation.

LOD Processing

Indicates if an application includes the processing of LOD resources on the client side. Processing LOD resources means that identical resources are searched, merged and stored on the device.

Our prototype and SwissPeaks are the only listed applications that use LOD resources as AR content. SwissPeaks fetches data only from GeoNames. Thus, the loaded resources do

not have to be checked for duplicate resources. The AR Mountain Guide fetches data from LinkedGeoData and GeoNames and therefore includes the filtering of identical resources.

Data Representation Standard

Data format supported by an AR application to process and display AR content.

The Wikitude World Browser uses ARML for describing POIs. Layar hides data structures from the user. Users are able to build AR overlays by creating a MySQL database and define and store POI descriptions in that database. But there does not exist any form of AR data format for describing the AR content. Also Mixare and LibreGeoSocial do not use an AR data format. Data is processed directly on the client side. SekaiCamera, Peak.AR, and SwissPeaks do not provide any information regarding the use of any data format for describing POIs. Our approach includes the fetching of RDF-based data directly through Web services from LOD sets like DBpedia or LinkedGeoData. Our system processes data on the device.

Filters

Indicates if the application provides filter options in order to load specific sets of POIs. For example, whether it is possible to only load Wikipedia articles or nearby published YouTube videos.

Wikitude World Browser, Sekai Camera, Mixare, and LibreGeoSocial provide filter options which allow end-users to choose between different overlays and POI categories. The Layar Reality Browser allows the loading of a single layer. But it is not possible to mix the layers for loading different groups of POIs at once. Peak.AR, SwissPeaks, and our AR prototype do not offer filters.

Social Networking Features

Projects which provide social networking features allow users to perform actions such as creating profiles, inviting friends, posting digital comments or posting media (image/audio/video). The user is able to share content and leave posts directly in physical space. Thus, friends of the social network are able to detect these posts later. As a result, these applications allow the integration of user-generated content directly through the AR interface of the application.

LibreGeoSocial and Sekai Camera offer social networking features and allow users to label geo-locations in physical space and leave text messages or even voice messages for friends. Wikitude offers a feature called myWorld[18] that enables users to create their own AR world and to geo-tag (associate with geo-coordinates) locations and notes in real-time directly on the mobile device. This Wikitude feature allows users to share the new world with their friends.

All the other presented applications, including our AR prototype, do not offer such kind of interactivity.

[18]Wikitude myWorld: `http://www.wikitude.com/myworld`

AR Content Features

Typically, the displayed AR content can be selected (e.g. touch icon) to trigger an action, e.g., view more information, send SMS or e-mail, take picture etc. So, this feature indicates if there are POI actions provided by the browser. POI actions could be:

Infobox: *information about the POI is displayed in a separate window*

Browsing: *the browsing for further information is possible*

Multimedia: *it is possible to listen to audio samples or to watch videos that are associated with a POI*

Map: *POI visualization on a 2D map*

Sharing: *share specific POI infos per e-mail, sms, or social networking platforms*

Route: *displaying route to POI on a 2D map*

Wikitude, Layar, Sekai Camera, and LibreGeoSocial offer the most POI actions. The Mixare standalone application only offers info boxes and browsing for further information. But the developer my extend the browser with further POI actions by modifying the source code. Peak.AR allows to display an info box if a POI is selected. SwissPeaks and our AR prototype are able to display info boxes and it is possible to browse for further information.

Platform

This criteria indicates on which mobile platform an application is deployable.

The table shows that Wikitude and Layar applications are available for many different mobile platforms. As these two platforms provide developer APIs for creating new AR overlays, they aroused the interest of many users and developers, and therefore became available on several mobile platforms. Mixare and Peak.AR are available for Android smartphones and iPhones. LibreGeoSocial is only available for Android smartphones, and SwissPeaks only for iPhones. Our prototype the AR Mountain Guide is deployable on the Android platform.

Off-line Mode

Indicates if the application also allows off-line support. Providing an off-line mode means that the application does not require a permanent network connection to work properly. It is possible to load AR content from the local storage system of the device.

Peak.AR provides an off-line mode. This means it is possible to load AR content in advance to save it on the local storage system of the device and to use the stored data as POI source. Our AR prototype also includes an off-line mode because the fetched resources from LinkeGeoData and GeoNames are stored on the local storage system after they have been processed (cf. Section 5.2.2). Additionally, GPS tour tracks may be saved on the the local storage system of the device and therefore users are able to display the tours on the map without an active network connection.

All the other appearing projects require permanent network connection for loading AR content.

Tour Guide Feature

This feature indicates if an application offers tour guide options. Tour guide option could be:

- ***loading** of **GPS tracks** for a specific area near the device*
- ***displaying GPS tracks** on a map*
- *offering of a **guide mode**, so that the user is able to pass a displayed tour*
- ***recording** of **GPS tracks***
- ***loading self-recorded GPS tracks** of the local storage system of the device*

Only our AR prototype includes tour guide features. The users are able to load tour tracks on a map and to activate a tour guide mode to be guided through the displayed hiking tour. We are interested in the possibilities of saving and exporting geo-coordinates in some kind of output-file. Furthermore, we elaborate on the visualization of tours on 2D maps, instead of integrating the tours in an AR interface. The visualization of tours in AR interfaces presents a possible future work, but we do not elaborate on this topic within this work. We present the realization of tour guide features in Chapter 5.

3.3 Summary

Several approaches of tracking methods for AR systems have already been developed. Currently available AR browsers for mobile phones mostly utilize sensor-based methods. Vision-based tracking systems include image matching and feature detection algorithms which may overburden the CPU of mobile phones if they are processed on the client side. Performing image matching processes on the server side causes the need for a permanent network connection as captured images by the device are transmitted to the server. Therefore, we define the conceptual model as a sensor-based approach to reliably determine the user's position and the camera's orientation based on this kind of tracking method.

Most AR browser platforms provide users the possibility to create AR content but the data have to be in a certain data format that the respective AR browser uses. AR content providers have to adapt their content to several AR systems to be able to publish content and visualize this content with the browser clients. The need for elaborating on a general open standard in the domain of AR and to standardize the way POIs are described and stored has to be addressed. Each AR browser architecture implements the storing and delivering of AR content in a different way. Wikitude's data format ARML for describing POIs, presents the first AR specification to standardize AR content descriptions [85]. Our approach of using Linked Data sets and thus requesting POIs based on the RDF format presents one possible step towards AR standardization.

The integration of Linked Data enables the creation of an AR system that follows an architectural infrastructure that includes data federation. The Web of Linked Data comprises geo-tagged resources which are linked with further descriptions. Data federation is possible because for a single resource descriptions from multiple Linked Data sets are retrievable. The examination of the architectural styles of most current AR browser systems exhibits that available AR content is locked down in separate data repositories. Thus, current AR browser systems are limited regarding to the selection and integration of data sources. They do not support the federation of POI description from different sources. Our approach exploits Linked Data resources which can be accessed through standard Web technologies. No data are stored in a separate database, no proprietary data format is necessary because Linked Data relies on the two fundamental Web technologies HTTP and URIs [21], and the federation of data to fetch related information of a single resource from multiple linked data sets is possible.

As a result of the previously accomplished state-of-the art analysis, we are able to affirm that the approach of this thesis presents a new way in the domain of mobile AR to identify aspects of the real world with the help of Linked Data sources and to deliver further information about these real-world objects based on Linked Data resources.

Chapter 4

Approach

In the last chapter, we described several AR tracking methods but also current projects that allow to visualize content within AR interfaces on mobile devices. Therefore, we now present our sensor-based tracking approach that allows to integrate Linked Data resources in a mobile AR system and to visualize these resources as virtual content within the AR interface of the system.

In the first section of this chapter, we describe the requirements and design considerations of our approach. We continue with an illustration of the abstract model of the AR system that serves as our proof-of-concept implementation and we also present the conceptual constituents of this system. In the next part of this chapter, we discuss aspects and issues of the integration of LOD sets into our AR system. Then, we follow with a formal description of the system's mathematical model. Finally, we present two activity models which present a functional illustration of work flows of our proposed AR system.

4.1 Requirements and Design Considerations

Our attempt of designing a mobile AR system that is capable of integrating LOD resources follows certain requirements and design considerations that we present in this section. Our AR system is specifically designed for direct deployment on mobile devices. The system does not present a client-server-based architecture where data is stored and processed on the server side. We retrieve data through existing LOD Web services and aggregate, consolidate (e.g. filter identical resources and merge them), and store LOD resources directly on the client side. Thus, our approach does not require a permanent network connection and is not constrained to any existing server-based infrastructure.

We separate our AR system in different modes: AR mode, Map mode, and Tour mode. The AR mode comprises the main concepts of our approach: the realization of the mathematical model, and the integration of LOD resources into AR interfaces. The other two modes present additional features to the system, as our proof-of-concept prototype presents an AR mountain

guide. The requirements for the map and tour mode will be mentioned in this section, but we will not further discuss and approach them in the next sections of this chapter. However, we present functional processes of the map and tour mode in Section 4.5 and we discuss implementation specific aspects of the map and tour mode in Chapter 5.

AR mode:

- Informing the AR system about the user's current location by using the GPS feature of the used mobile phone.
- Permanently accessing the device's sensors to determine the horizontal and vertical orientation of the phone.
- Fetching of mountain resources from LOD sets, so that the AR system is able to use RDF data as AR content.
- Identifying identical LOD resources in order to remove duplicate resources and to merge the resource descriptions of the identical resources.
- Integration of recorded sensor data, GPS data, and Linked Data resources into the mathematical model to be able to calculate screen coordinates for virtual objects.
- Visualizing virtual objects (representing the real-world resources) on the corresponding positions on the screen by using the earlier calculated screen coordinates.
- Consisting alignment of virtual objects with the corresponding real-world resources on the screen, if the user moves the device in various directions.
- Retrieving additional information about resources.

Map/Tour mode:

- Displaying of a map.
- Loading hiking tours for a certain restricted area.
- Extracting tour data of GPX files. Followed by the processing and visualizing of GPX-based tour data on a map.
- Displaying user position on the map by accessing the GPS radio of the device.
- Loading of mountain resources in the near vicinity and the screening of markers (presenting the resources) on a map.
- Retrieval of additional information about displayed resources.
- Displaying digital compass for active tour guide mode.

Our approach will not include methods for loading tours into the AR interface of the AR system.

The scope of our approach is not to provide a mobile AR LOD browsing platform that allows the visualization of all kind of LOD resources. We limit our scope to the use of mountain-specific Linked Data resources and we fetch these kind of resources from three specific LOD sets LinkedGeoData, GeoNames, and DBpedia to enable not only a theoretical elaboration of the research questions in Section 1.4 but also to provide a demonstration of our approach in practice.

Concerning the environment in which the AR system will be used, the system should be designed to be accurate, robust, and low cost so that the system does not overburden the mobile device's hardware. This means we have to consider the issue of power consumption and the draining of the battery life of the device. Our approach requires that the used mobile phone is equipped with a digital camera, a GPS radio, a digital compass, an accelerometer, and it has to offer networking support in order to access online LOD resources.

4.2 Abstract and Conceptual Architecture

In the primary part of this section, we present the abstract model of our AR system approach. In the second part, we present the conceptual constituents of the proposed AR system that integrates LOD resources and includes a sensor-based mathematical model.

4.2.1 Abstract Model

The model in Figure 4.1 presents an abstract illustration of our AR system. The AR system is running on a mobile device and comprises the mathematical model which is in need of the device's recorded sensor data (location and orientation data), and the Linked Data resources fetched from different LOD sets.

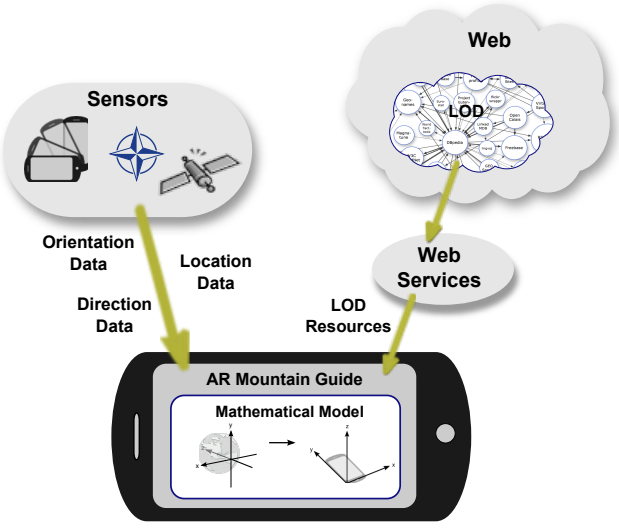


FIGURE 4.1: Abstract model of AR system[1]

A sensor-based mobile AR system has to be informed about the current position on earth and the orientation and direction of the device. This is important because with the knowledge about heading and orientation of the device, it is possible to figure out which resources are located near the device and whether some of them appear in the captured view of the digital camera of the device. In our system, sensor data presents data received from the GPS radio to obtain location information, and data recorded from the digital compass and accelerometer for

[1]The figure includes pictures from [32, 86]

determining the current orientation and heading of the device. The mathematical model is in need of sensor data because it is responsible for calculating the screen coordinates to visualize LOD resources on the corresponding positions onto the device's screen. As mentioned in the previous section, we make use of Web services provided by LOD sets of the Web, to load LOD resources.

4.2.2 Conceptual Constituents

In the following section, we explain our AR system on a conceptual basis. Therefore, we illustrate in Figure 4.2 a component diagram of the proposed AR system that relies on a sensor-based mathematical model (cf. Section 4.4). The component diagram meets the requirements specified in Section 4.1 and is based on the abstract illustration (cf. Figure 4.1) presented in the previous section.

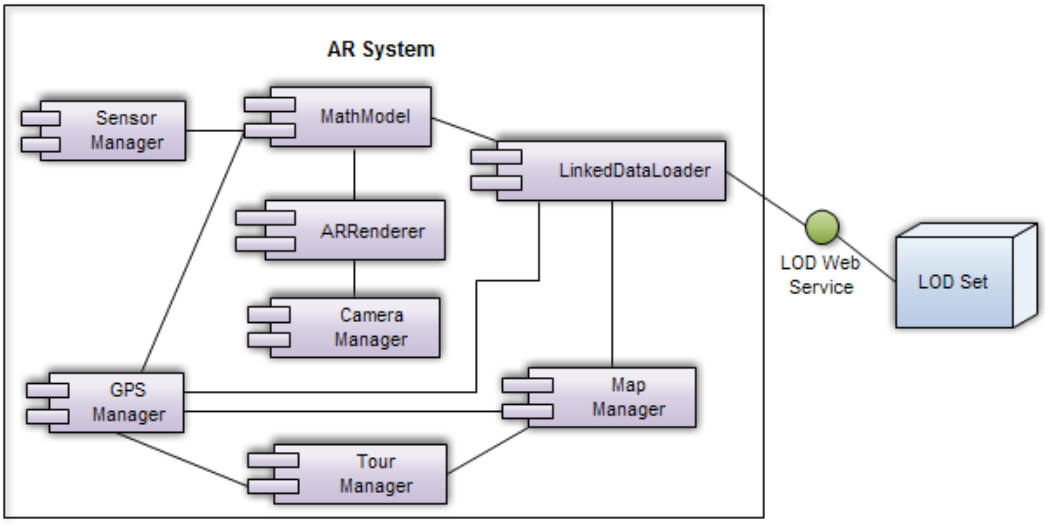


FIGURE 4.2: Conceptual system architecture of proof-of-concept AR system

Figure 4.2 contains the package component *AR System* which comprises all the components of the system:

- The system includes a *SensorManager* which is responsible for managing the access to the sensors of the device, and for updating the system with new sensor values. With the help of the *SensorManager* we are able to inform the AR system about the current heading and orientation of the mobile device.
- The *GPSManager* is responsible for activating the GPS radio of the device in order to determine the current position on earth and to update the other components of the system about location changes.
- The *LinkedDataLoader* component is responsible for loading Linked Data resources in form of RDF-based data from LOD sets. For accessing data, all the queries against the

external LOD Web services are in need of spatial parameters (geographical coordinates) to enable the fetching of resources that describe nearby real-world objects. Therefore, the *LinkedDataLoader* is informed by the *GPSManager* about the current location of the device and this information is used as the spatial parameters for the queries against the LOD sets. After resources are locally available on the device, the component is also responsible for the filtering of identical resources and for the merging and storing of the remaining resources.

- The *MathModel* is responsible for recognizing real-world resources in the captured scene of the device's digital camera and for mapping the geographical coordinates of these resources into screen coordinates. Thereby, the *SensorManager* and the *GPSManager* constantly deliver the updated sensor values to the *MathModel* and the *LinkedDataLoader* provides the LOD resources.

- The *ARRenderer* generates the AR Interface of the system. The interface consists of several virtual layers and a camera layer (cf. Section 5.1.2). This component also manages the drawing and redrawing of virtual objects on top of the live camera view. Thereby, the *ARRenderer* uses the calculated screen coordinates from the *MathModel* component for visualizing the virtual content.

- The *CameraManager* is responsible for accessing the digital camera of the device and for providing the system with live camera view that serves as basis layer for the *ARRenderer* to place virtual content on top of it.

As our proof-of-concept AR system also offers functions for displaying a map and for loading tour tracks from the Web for a certain restricted area, two more components are visible in Figure 4.2:

- The *MapManager* is responsible for displaying a map, if the user activates the map mode of the system. This component is also associated with the *GPSManager*, as location updates are used to visualize the current position on the map. Additionally, the system is able to visualize LOD resources on the map. Thus, the *MapManager* is connected with the *LinkedDataLoader* to be able to load Linked Data resources and to visualize them on the map.

- The *TourManager* component implements methods for loading GPS tracks from the Web. This component is associated with the *GPSManager* to be able to load nearby tours but also with the *MapManager*, so that we are able to visualize loaded GPS tours on a 2D map.

4.3 LOD Integration

In this section, we discuss several aspects that we have to consider if we integrate LOD resources into a mobile AR system. First of all, we explain the data structures of LOD sets to be able to access specific resources and the respective descriptions. We also have to consider that the data may comprise duplicate resources. Thus, in the last part of this section (Section 4.3.2), we discuss the issue of identifying identical resources and of merging multiple resource descriptions.

4.3.1 Ontology Structures

To be able to receive data from LOD sets, we have to be aware of the data structures of the Linked Data sets. Therefore, Linked Data ontologies enable it to categorize data and search content of different types. An ontology defines classes of entities, instances, and their properties [87]. Each Linked Data set of the LOD initiative defines an ontology which comprises a different vocabulary and structure. Thus, we have to determine how the geo-information instances of the entities are defined in each ontology. Then, we are able to define proper search queries and to use location information as entry point to the LOD sets. In addition, we have to determine how entity types are defined within the different ontologies, to be able to fetch sets of data which exhibit a certain type such as places, mountains, or cities. As we already outlined in Section 4.1, we use in the scope of our thesis only resources of LinkedGeoData, GeoNames, and DBpedia. Thus, we discuss the structures of these ontologies in the following paragraphs. Table 4.1 comprises all the prefixes that we use within this section to describe the different ontologies.

Prefix	URL
gn:	`http://www.geonames.org/ontology#`
lgdo:	`http://linkedgeodata.org/ontology/`
wgs84:	`http://www.w3.org/2003/01/geo/wgs84_pos#`
dbpedia:	`http://dbpedia.org/resource/`
dbpedia-owl:	`http://dbpedia.org/ontology/`
dbpedia-prop:	`http://dbpedia.org/property/`
rdf:	`http://www.w3.org/1999/02/22-rdf-syntaxns#`
rdfs:	`http://www.w3.org/2000/01/rdf-schema#`
owl:	`http://www.w3.org/2002/07/owl#`
dcterms:	`http://dublincore.org/documents/2012/06/14/dcmi-terms/`
georss:	`http://www.georss.org/georss/`

TABLE 4.1: Table of prefixes

GeoNames Ontology

The GeoNames ontology defines each instance to be of type `gn:Feature`. In addition, every type belongs to a specific `gn:featureClass` and exhibits also a specific `gn:featureCode`. For instance, the feature class `T` stands for mountains, hills, and rocks, and a feature code `P`

further sub-categorizes `T` into peaks or a feature code `PASS` into mountain passes. The name of an entity is defined via the property `gn:name`.

The GeoNames ontology describes geo-spatial information with the properties `wgs84:lat`, `wgs84:long`, and `wgs84:alt` of the W3C's Basic Geo Vocabulary.

LinkedGeoData Ontology

The LinkedGeoData ontology defines classes as instances of `owl:Class`. Entities which present mountains are expressed as a `rdf:type` of the class `lgdo:Peak`. This ontology does not comprise sub-categorizes to further categorize mountains. To define an entity's name, this ontology uses the property `rdfs:label`.

LinkedGeoData applies W3C's Basic Geo Vocabulary and describes geo-spatial information as a pair of `wgs84:lat` and `wgs84:long` properties and also as a single property with `wgs84:geometry` (e.g. `POINT(12.7106 47.1003)`). Additionally, LinkedGeoData uses GeoRSS Simple encoding[2] of the W3C Geospatial Vocabulary[3] to also describe geographical information with the property `georss:point`. This ontology does not use the property `wgs84:alt` for defining the elevation value of a geographical point. It describes the elevation value of resources via the property `lgdo:elevation`.

DBpedia Ontology

We apply GeoNames and LinkedGeoData resources as the AR content that we visualize on the screen, and we use DBpedia for fetching further descriptions about the already visualized resources within the AR interface of the system. Thus, we describe more properties of the DBpedia ontology than we did when presenting the LinkedGeoData and GeoNames ontologies.

The DBpedia ontology defines mountain resources as type `dbpedia-owl:Mountain` which is also an entity of type `owl:Class`. DBpedia divides mountains into different categories (e.g. glaciers of the Alps, glaciers of Austria, Glockner Group) using the property `dcterms:subject`. The DBpedia ontology also provides a property (`dbpedia-owl:mountainRange`) to assign mountains to the respective mountain range in which the mountain is located. This ontology uses the two properties `rdfs:label` and `dbpedia-prop:name` for defining the label of the corresponding DBpedia resource.

Geo-spatial information is available via the properties `wgs84:lat`, `wgs84:long`, `wgs84:geometry`, and `georss:point`. Instead of using `wgs84:alt`, DBpedia defines the elevation value of a resources with the property `dbpedia-owl:elevation`.

DBpedia entities are described by short and long abstracts. The ontology uses `rdfs:comment` for defining short abstracts and `dbpedia-owl:abstract` for defining long abstracts (about 500 words). As we access data on a mobile device, long descriptions are not suitable for the small screens of mobile devices. Thus, we concentrate on the property `rdfs:comment`.

[2]GeoRSS: `http://www.w3.org/2005/Incubator/geo/XGR-geo-20071023/#model`
[3]W3C Geospatial Vocabulary: `http://www.w3.org/2005/Incubator/geo/XGR-geo-20071023/`

4.3.2 Discovering Identical Resources

If we use several Linked Data sets to load data, we have to find a way to consolidate the fetched resources. A single resource my occur multiple times in different Linked Data sets but is identified with a different URI. Thus, we also have to resolve the issue of redundancy among the various resources. This means that we have to filter identical resources and merge the remaining resource descriptions. Comparing resources by their geographical coordinates to find redundant resources is not efficient enough, as for the same resources often different geo-information is available. Discrepancies of geographical locations from equivalent resources from different LOD sets often exceed a distance of 20 m, sometimes hundreds of meters [88]. By contrast, some ontologies include predicates which help in identifying identical resources:

owl:sameAs

The predicate `owl:sameAs` of the OWL ontology can be used to define that two different URIs describe a single resource. Content creators are able to use this OWL predicate to interlink differed LOD sets. Therefore, we can use it to merge data from different LOD repositories [89].

By example, the URI of the real-world entity Kitzsteinhorn at LinkedGeoData is `http://linkedgeodata.org/page/node36688518`, at GeoNames at is `http://sws.geonames.org/2774332`, and in DBpedia `http://dbpedia.org/resource/Kitzsteinhorn`. Thus, a description of the Kitzsteinhorn at DBpedia includes statements (cf. Listing 4.1) which define that this resource is also described at other LOD sets.

```
1 dbpedia:Kitzsteinhorn owl:sameAs gn:2774332
2 dbpedia:Kitzsteinhorn owl:sameAs lgdo:node36688518
```

Listing 4.1: `owl:sameAs` example

rdfs:seeAlso

The predicate `rdfs:seeAlso` of the RDF Schema Specification provides an alternative way to refer to an further equivalent resource of different data sets. For instance, the GeoNames ontology uses this predicate to interlink resources with related resources of external data sets (cf. Listing 4.2).

```
1 gn:Kitzsteinhorn rdfs:seeAlso dbpedia:Kitzsteinhorn
```

Listing 4.2: `rdfs:seeAlso` example

4.3.2.1 The Silk Framework

If we fetch LOD resources and save them locally within a triple store, we are able to use an existing framework to identify identical resources and to further merge identical resource

descriptions into a single graph. The Silk [90] link discovery framework is a tool for discovering identical resources within different data sets. The framework includes Silk - Link Specification Language (Silk-LSL), which is a declarative language that allows to define certain linkage rules to be able to compare properties of the entities. The user has the possibility to define conditions that RDF links must comply in order to be interlinked. These link conditions can be based on various similarity metrics (e.g. Jaro-Winkler Distance). In addition, it is possible to weight and combine the receiving results through various similarity aggregation functions (e.g. weighted average) [90]. The data sources can be accessed using the SPARQL protocol. Hence, it is possible to identify identical links between local and remote data sources.

In Listing 4.3, we present a Silk-LSL example that enables to identify identical entities of LinkedGeoData and GeoNames. A LSL file is XML based and starts with the root element `<Silk>`. Within the `<Interlinks>` section in the LSL file we are able to define the interlinks that have to be generated between the two LOD sets. In this example, we state that a link of a given type (we use `owl:sameAs`, cf. line 5) has to be established between two entities if a specified condition (defined under `<LinkageRule>` lines 17-36) is met. We define the sets of data items, which have to be compared, through the `<SourceDataset>` (lines 6-11) and `<TargetDataset>` (lines 12-16) directives. We also set a restriction for LinkedGeoData and GeoNames resources to compare certain restricted groups of resources. The `<RestrictTo>` (e.g. lines 7-10) statements may contain SPARQL expressions that are usually defined in the WHERE clause of a SPARQL query [90]. In our example, we restrict the data sets to GeoNames entities which exhibit a feature class `gn:T` and to LinkedGeoData entities which are of type `lgdo:Peak`. Additionally, we restrict the data sets to the geographical information of these entities expressed via the WGS84 Geo Positioning Ontology.

We are able to specify linkage rules using the `<LikageRule>` directive. A linkage rule defines how the selected data items are to be compared for similarity [91]. In Silk, the input comparison is based on distance measures and a threshold defined by the user. We define a threshold of 1 in order to obtain a positive similarity score if distance values are between 0 and 1 (distance value 0 = perfect match, higher values imperfect) [91]. Similarity scores are also between between 0 or 1 where higher values indicate a greater similarity.

First of all, we compare the names of the mountains using the Jaro-Winkler distance (lines 20-22). We use the aggregation function `max` to combine the similarities of the Jaro-Winkler distance metric. This function evaluates to the highest confidence in the group [91]. We also define a weight parameter of 2. The weight indicates which comparison metric results are more relevant than others during the generation of a link between two resources [91], and it is considered by the weighted average aggregation function that we use to combine the similarities between the names of the entities and the similarities based on the geographical distance between these entities (lines 24-34). To calculate the geographical distance we use a numerical distance measure called `wgs84` that allows us to define a maximum distance parameter in kilometers. Prior the comparison, we transform the geographical resource information into

equal formats to normalize these values because the different data sources use different data formats (lines 25-32). Silk offers again various forms of transformation functions (e.g. lower case, upper case transformations). In this example, we use the function `concat` which enables the concatenation of two input strings.

```
1  <Silk>
2   ...
3    <Interlinks>
4      <Interlink id="mountain">
5        <LinkType>owl:sameAs</LinkType>
6        <SourceDataset dataSource="geonames" var="a">
7          <RestrictTo>
8            ?a gn:featureClass gn:T.
9            ?a wgs84:lat ?lat.  ?a wgs84:long ?long.
10         </RestrictTo>
11       </SourceDataset>
12       <TargetDataset dataSource="linkedgeodata" var="b">
13         <RestrictTo>
14           ?b a lgdo:Peak.  ?b wgs84:lat ?lat.  ?b wgs84:long ?long.
15         </RestrictTo>
16       </TargetDataset>
17       <LinkageRule>
18         <Aggregate type="average" required="true">
19           <Aggregate type="max" required="true" weight="2">
20             <Compare metric="jaro" threshold="1">
21               <Input path="?a/gn:name"/> <Input path="?b/rdfs:label"/>
22             </Compare>
23           </Aggregate>
24           <Compare metric="wgs84" required="true">
25             <TransformInput function="concat">
26               <Input path="?a/wgs84:lat"/> <Input path="?a/wgs84:long"/>
27               <Param name="glue" value=" " />
28             </TransformInput>
29             <TransformInput function="concat">
30               <Input path="?b/wgs84:lat"/><Input path="?b/wgs84:long"/>
31               <Param name="glue" value=" " />
32             </TransformInput>
33             <Param name="unit" value="km"/> <Param name="threshold" value="1"/>
34           </Compare>
35         </Aggregate>
36       </LinkageRule>
37       <Filter threshold="0.8"/>
38       ...
39 </Silk>
```

Listing 4.3: Silk-LSL example: Interlinking mountain resources between LinkedGeoData and GeoNames

Within a Silk file it is possible to set a filter for additionally filtering the generated links (line 37). The threshold parameter allows the removal of all the links that fall below the defined value. After the execution of the Silk file, the generated links present the foundation of merging resources from LinkedGeoData and GeoNames to a single data graph. Using the Silk framework is just one way to solve the issue of duplicate LOD resources. We will not make use of Silk

in our proof-of-concept implementation because it was not possible to apply this framework on a mobile device. Therefore, we realized another method which also relies on the Jaro-Winkler distance measure and the calculation of the geometric distance of the resources to identify identical resources in our data dump. We describe our realization of this filter method and the further explanations in the next chapter in Section 5.2.2.3.

4.4 Formal Model

As the mathematical model of our proposed AR system is a central aspect of our approach, we now describe the main elements of this model. Thereby, we describe it on a formal basis using set theory and algebraic specifications. At last, we present the AR tracking algorithm of our mathematical model based on the introduced symbols and equations.

4.4.1 Symbols and Formal Definitions

To be able to operate on the incoming sensor values and the loaded LOD resources, the mathematical model of our AR system comprises several calculation steps. We summarized these calculations to a tracking algorithm that we present later in Section 4.4.2. In the following sections, we define and explain all the calculations our mathematical model contains. Therefore, we introduce a number of symbols that we use throughout this section to describe and define the calculations, and the algorithm of the mathematical model within Table 4.2. Along the presentation of the calculations, we also give a connected example scenario to present the presented formulas in practice. In the end, this example scenario delivers screen coordinates for a LOD resource which enable a proper visualization of a graphical icon representing this LOD resource within the AR interface of our AR system.

First of all, we start with a formal definition of a LOD resource within our AR system and the definition of the geographical location of the used device on which the AR systems runs (cf. Definition 4.1). Then, we give further definitions for all the other symbols listed in Table 4.2 in the following sub-sections.

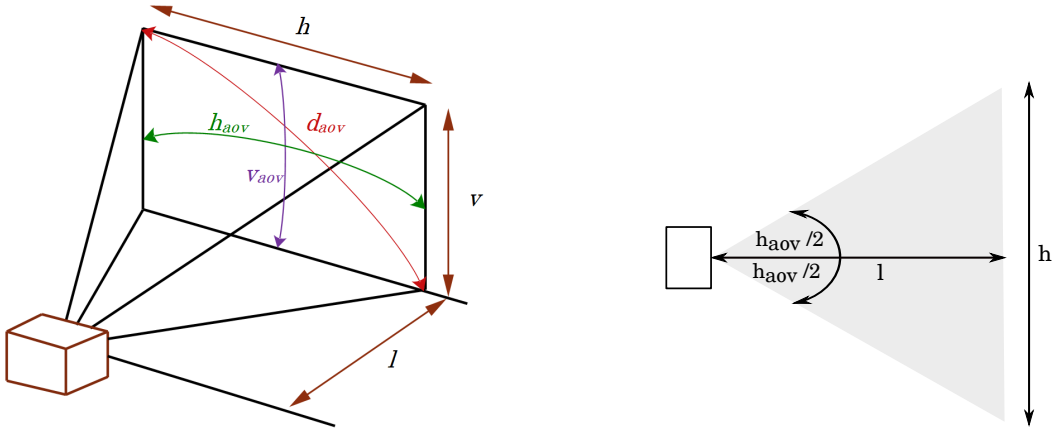
Definition 4.1. Let RES denote the set of all loaded LOD resources $res \in RES$ of our AR system. A resource $res \in RES$ is defined as the 3-tuple $(\lambda_{res}, \varphi_{res}, \psi_{res})$. Let DL be the set of all device location parameters $dl \in DL$, where a device location $dl \in DL$ is defined as the 3-tuple $(\lambda_{dl}, \varphi_{dl}, \psi_{dl})$.

Symbols	**Description/Unit**	**Reference**
res	Loaded LOD resources of the AR system	Definition 4.1
dl	Device location	Definition 4.1
ds	Device sensor parameters	Definition 4.7
fov	Field of view	Definition 4.8
rp	Resource position on the horizontal or vertical angle of view	Definition 4.8
sp	Screen point	Definition 4.9
RES	Set of all loaded LOD resources res	Definition 4.1
DL	Set of all device location parameters dl	Definition 4.1
DS	Set of all device sensor parameters ds	Definition 4.7
FOV	Set of all border parameters of the field of view fov	Definition 4.8
RP	Set of all resource positions rp on the horizontal and vertical angle of view	Definition 4.8
SP	Set of all screen points sp	Definition 4.9
α	Recorded azimuth angle of device / in degrees	Section 4.4.1.4
ρ	Recorded pitch angle of device / in degrees	Section 4.4.1.4
λ_{dl},λ_{res}	Longitude of $dl \in DL$ and $res \in RES$	Definition 4.1
φ_{dl},φ_{res}	Latitude of $dl \in DL$ and $res \in RES$	Definition 4.1
ψ_{dl},ψ_{res}	Altitude of $dl \in DL$ and $res \in RES$	Definition 4.1
d	Geodesic distance between device $dl \in DL$ and resource $res \in RES$/ in meters	Definition 4.3, Definition 4.4
r	Range radius within which LOD resources are queried / in meters	
β	Bearing angle between device $dl \in DL$ and resource $res \in RES$ / in degrees	Definition 4.5
θ	Inclination angle from device $dl \in DL$ to resource $res \in RES$/ in degrees	Definition 4.6
h_{aov}, v_{aov}	Horizontal and vertical angle of view / in degrees	Definition 4.2
l_{border}, r_{border}	Left and right border of field of view fov / in degrees	Definition 4.7
u_{border}, lo_{border}	Upper and lower border of field of view fov / in degrees	Definition 4.7
$screen_w$, $screen_h$	Screen width and height of device / in pixels	Definition 4.7
x_{haov}, y_{vaov}	Position of resource $res \in RES$ on horizontal and vertical angle of view of camera / in degrees	Definition 4.8
x_{cor}, y_{cor}	x and y coordinate for a resource $res \in RES$ on screen / in pixels	Definition 4.9
l	Focal length of camera lens / in millimeters	Definition 4.2
h, v	Horizontal and vertical dimensions of camera image format / in millimeters	Definition 4.2

Table 4.2: Mathematical symbols for formal model

4.4.1.1 Angle of View of Camera

The angle of view of a camera defines the scene of the reality which the camera lens captures. Figure 4.3 (a) illustrates that there are three types of angles: the horizontal h_{aov}, vertical v_{aov}, and diagonal d_{aov} angle of view. We also depict the focal length l of a lens and the horizontal h and vertical v dimension of the used image format from the captured scene. The focal length represents the distance from the center of the camera lens to a focal point (object of a scene) [92].



(a) Three angles of view

(b) h_{aov} - camera top view

FIGURE 4.3: Angles of view of a camera lens

In Figure 4.3 (b), we illustrate the different angles of view from the camera's top view. We split the triangle into two right triangles, so that we are able to apply a basic trigonometry formula (cf. Definition 4.2).

Definition 4.2. We define two inverse trig functions

$$f_{h_{aov}}(l, h) = 2 \arctan\left(\frac{h/2}{l}\right) \tag{4.1}$$

$$f_{v_{aov}}(l, v) = 2 \arctan\left(\frac{v/2}{l}\right) \tag{4.2}$$

that calculate the viewing angles h_{aov} and v_{aov} of a fixed focal length l and with the given image format dimensions h and v.

As an example, let the fixed focal length l of the device's camera lens be 3.53 mm and let the horizontal image format h of the camera be 3.66 mm and the vertical image format 2.746 mm. In

consequence, we obtain the horizontal and vertical angle of view with the following calculations:

$$\begin{aligned} h_{aov} &= 2 * \arctan\left(\frac{3.66\,mm/2}{3.53\,mm}\right) \\ &= 2 * \arctan(0.5184136) * (180/\pi) \\ &= 2 * 27.40284^{\circ} \\ &\approx 54.8^{\circ} \end{aligned}$$

$$\begin{aligned} v_{aov} &= 2 * \arctan\left(\frac{2.746\,mm/2}{3.53\,mm}\right) \\ &= 2 * \arctan(0.3889518) * (180/\pi) \\ &= 2 * 22.2853^{\circ} \\ &\approx 44.6^{\circ} \end{aligned}$$

We additionally multiply by $(180/\pi)$ to convert radians to degrees.

4.4.1.2 Distance and Bearing

In our mathematical model, we also need to include the determination of the distance in meters between the current GPS location and the locations of real-world objects. There exist many different ways to calculate the distance between two geographical points, whereby they vary in complexity and accuracy.

For instance, T. Vincenty [93] published in the year 1975 two iterative methods for the calculation of the distance between two locations on the surface of a spheroid (the earth's shape is seen as a globe that is sparsely flattened on its axes): the *direct* and *inverse* formula. The direct formula determines a destination point representing the distance traveled and the azimuth to the start point. With the help of the inverse formula it is possible to calculate the distance between two geographical points and the bearing angle.

In our approach, we use Vincenty's inverse formula for gaining the distance d (cf. Definition 4.3 and Definition 4.4) between two locations because the formula's accuracy lies within 0.5 mm on the ellipsoid being used (e.g. WGS84) [94]. Simpler distance calculation methods, such as the Haversine formula (which assumes that the earth is a sphere), exhibit less accuracy.

Definition 4.3. Let $d(res, dl)$ denote the geographic distance between two points $res \in RES$ and $dl \in DL$. Let a and b be the major and minor semi-axes of the ellipsoid. Then, we get the ellipsoidal flattening $flat$ through Equation 4.3.

$$flat = \frac{a - b}{a} \tag{4.3}$$

Let (φ_{res},λ_{res}) and (φ_{dl},λ_{dl}) be two geographical points. The reduced latitude $\phi(\varphi)$ (latitude on the auxiliary sphere) for the two points is defined by:

$$\phi_{dl} = \arctan\big((1-flat)\ \tan\varphi_{dl}\big) \tag{4.4}$$

$$\phi_{res} = \arctan\big((1-flat)\ \tan\varphi_{res}\big) \tag{4.5}$$

Furthermore, let $\Delta\lambda$ be the difference in longitude (λ_{res}-λ_{dl}) of two points res and dl. Now, we set the initial value of $\lambda = \Delta\lambda$. Then, we iterate over Equations 4.6-4.12 until λ converges.

$$\sin\sigma = \sqrt{(\cos\phi_{res}\ \sin\lambda)^2 + (\cos\phi_{dl}\ \sin\phi_{res} - \sin\phi_{dl}\ \cos\phi_{res}\ \cos\lambda)^2} \tag{4.6}$$

$$\cos\sigma = \sin\phi_{dl}\ \sin\phi_{res} + \cos\phi_{dl}\ \cos\phi_{res}\ \cos\lambda \tag{4.7}$$

$$\sigma = \arctan\frac{\sin\sigma}{\cos\sigma} \tag{4.8}$$

Now, let con_c be Vincenty's constant for computation of longitude difference:

$$\sin\alpha = \cos\phi_{dl}\ \cos\phi_{res}\ \sin\lambda/\sin\sigma \tag{4.9}$$

$$\cos 2\sigma_m = \cos\sigma - 2\ \sin\phi_{dl}\ \sin\phi_{res}/\cos^2\alpha \tag{4.10}$$

$$con_c = \frac{flat}{16}\ \cos^2\alpha\ \big(4 + flat\ (4 - 3\ \cos^2\alpha)\big) \tag{4.11}$$

$$\Delta\lambda = \lambda - (1-con_c)\ flat\ \sin\alpha\ \Big(\sigma + con_c\ \sin\sigma\ \big(\cos 2\sigma_m + con_c\ \cos\sigma\ (-1+2\ \cos^2 2\sigma_m)\big)\Big) \tag{4.12}$$

where σ denotes the angular distance of two points on the sphere and σ_m is the angular distance on sphere from equator to line midpoint.

Vincenty developed this formula with the consideration to efficiently be applied in implementations. Therefore, in each iteration of the formula only each of sin, cos, *sqrt*, and arctan are used. Thus, it usually takes four iterations to gain a result [94]. Is the change in λ negligible, we are able to compute the geodesic distance d between the two points in the ellipsoid (cf. Definition 4.4).

Definition 4.4. Let con_a and con_b be Vincenty's constants for the computation of σ, and let u^2 be a geodesic constant u-squared. We obtain the distance d through Equations 4.13-4.17.

$$u^2 = \frac{\cos^2\alpha\ (a^2 - b^2)}{b^2} \tag{4.13}$$

$$con_a = 1 + \frac{u^2}{16384}\ \Big(4096 + u^2\ \big(-768 + u^2\ (320 - 175\ u^2)\big)\Big) \tag{4.14}$$

$$con_b = \frac{u^2}{1024}\ \Big(256 + u^2\ \big(-128 + u^2\ (74 - 47\ u^2)\big)\Big) \tag{4.15}$$

$$\Delta\sigma = con_b\ \sin\sigma\Big(\cos 2\sigma_m + \frac{1}{4}\ con_b\ \big(\cos\sigma(-1 + 2\ \cos^2 2\sigma_m) - \frac{1}{6}\ con_b\ \cos 2\sigma_m\ (-3 + 4\ \sin^2\sigma)\ (-3 + 4\ \cos^2 2\sigma_m)\big)\Big) \tag{4.16}$$

$$d = b\ con_a\ (\sigma - \Delta\sigma) \tag{4.17}$$

where distance $d \in [0, r]$ (r is the range in meters which we define to restrict the number of real-world objects that are visualized within the AR interface).

The model for the earth ellipsoid, which we apply within the inverse formula for the distance calculation in our approach, is WGS-84. Therefore, we define the semi-major axis (equatorial radius) a = 6,378,137 m and the semi-minor axis (polar radius) $b \approx$ 6,356,752.314245 m and thus we obtain $f \approx 1\ /\ 298.257223563$.

We mentioned earlier, that Vincenty's inverse formula also provides initial and final bearing values. According to [95], bearing is the measurement of direction between two points. In Definition 4.5, we define Vincenty's additional formulas for obtaining the initial and final bearing angles. In our approach, we need the bearing angle β between the current position of the device $dl \in DL$ to each loaded resource $res \in RES$ in order to be able to check if a resource lies within the current field of view (cf. Section 4.4.1.4 and Defintion 4.8).

Definition 4.5. Let β_1 be the initial bearing angle and let β_2 be the final bearing angle from $dl = (\lambda_{dl}, \varphi_{dl}, \psi_{dl})$ to $res = (\lambda_{res}, \varphi_{res}, \psi_{res})$. They are defined by

$$\tan\beta_1 = \frac{\cos\phi_{res}\ \sin\lambda}{\cos\phi_{dl}\ \sin\phi_{res} - \sin\phi_{dl}\ \cos\phi_{res}\ \cos\lambda} \tag{4.18}$$

$$tan\beta_2 = \frac{\cos\phi_{dl}\ \sin\lambda}{-\sin\phi_{dl}\ \cos\phi_{res} + \cos\phi_{dl}\ \sin\phi_{res}\ \cos\lambda} \tag{4.19}$$

where we use ϕ_{dl} and ϕ_{res} from Equation 4.4 and Equation 4.5 and λ from Equations 4.6-4.12.

4.4.1.3 Inclination Angle

The inclination angle θ defines the angle from the current position on earth to the altitude ψ_{res} (in meters) of a resource location. Altitude or elevation is the height above sea level of a resource $res \in RES$ or device location $dl \in DL$. We use the inclination value later to check if a resource lies within the vertical angle of view v_{aov} of the camera (cf. Equation 4.29). We determine the

inclination angle θ by applying a basic trigonometry function. Figure 4.4 illustrates a triangle with its three sides called hypotenuse, adjacent, opposite, and the angle θ. In Definition 4.6, we define the respective inclination function.

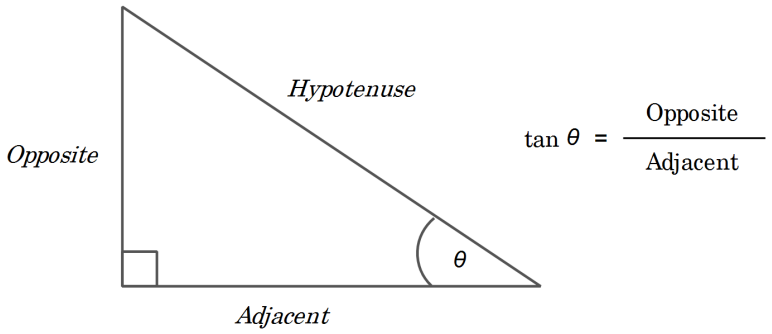


FIGURE 4.4: Basic trigonometry triangle

Definition 4.6. Let $\Delta\psi$ be the difference in altitude

$$\Delta\psi = \begin{cases} \psi_{res} - \psi_{dl} & \text{if } \psi_{res} > \psi_{dl} \\ (\psi_{dl} - \psi_{res}) * -1 & \text{otherwise} \end{cases} \tag{4.20}$$

of two points $dl = (\lambda_{dl}, \varphi_{dl}, \psi_{dl})$ and $res = (\lambda_{res}, \varphi_{res}, \psi_{res})$. Now, let $\Delta\psi$ be the opposite side of the triangle in Figure 4.4 and let the distance d (cf. Definition 4.3 and Definition 4.4) between the points dl and res be the adjacent side of the triangle in Figure 4.4. We define an inclination function $\theta(res, dl)$ as

$$f_\theta(res, dl) = \arctan\left(\frac{\Delta\psi}{d}\right) \tag{4.21}$$

By example, let the altitude of the current location be 342 m and the altitude of a resource 394 m. Now, if we have a distance of 4,496.86 m between the two locations, we are able to obtain:

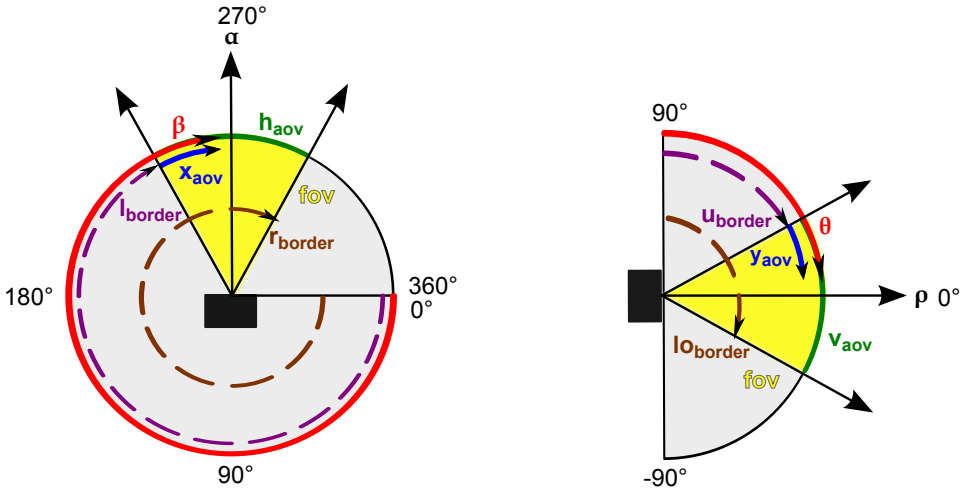
$$\theta = \arctan\left(\frac{394\,m - 342\,m}{4,496.86\,m}\right) * (180/\pi) \approx 0.6625^\circ$$

4.4.1.4 Field of View

The field of view presents the actual scene which the user observes through the display of the device. If a resource appears in the current field of view, a marker will be visualized on top of it. The virtual content and the real-world object have to stay aligned. Thus, every time the user moves, the current field of view is re-calculated. Therefore, we use azimuth α and pitch ρ values, as well as the horizontal h_{aov} and vertical v_{aov} view angles of the camera lens to calculate the borders of the field of view fov (cf. Definition 4.7). We obtain azimuth α and pitch ρ by the

device's built-in magnetometer and the accelerometer sensors. Azimuth and pitch define the direction and orientation of a device. In our model, the azimuth angle α presents the direction angle (the current heading of the device) $\alpha \in [0°, 360°]$ and the pitch angle $\rho \in [-90°, 90°]$ presents the rotation of the device forwards or backwards.

In Figure 4.5 (a), we illustrate the camera's top view and in Figure 4.5 (b), there is the camera's side view visible. The green angle presents the horizontal or vertical angel of view h_{aov} and v_{aov} of the camera. The purple and the brown angles are the left, right, upper, and lower borders of the field of view fov of the camera. The field of view itself is outlined by the yellow colored area between the black arrows. The red angle presents the bearing angle β in Figure 4.5 (a) and the inclination angle θ of a resource in Figure 4.5 (b). The black arrow in the middle of the field of view presents the current heading direction α in Figure 4.5 (a) and the current pitch angle ρ of the device in Figure 4.5 (b).



(a) Camera Top View (b) Camera Side View

FIGURE 4.5: h_{aov}, v_{aov} and borders of field of view

Definition 4.7. Let DS be the set of all device sensor parameters $ds \in DS$ that we receive from the device sensors. Then, let $ds = (\alpha,\rho,h_{aov},v_{aov},screen_w,screen_h)$ be a 6-tuple of device sensor parameters. We define two border functions

$$f_{l_{border}}(ds) = \alpha - \frac{h_{aov}}{2} \tag{4.22}$$

$$f_{r_{border}}(ds) = \alpha + \frac{h_{aov}}{2} \tag{4.23}$$

for obtaining the left and right borders of the horizontal field of view of the device's camera. The borders have to be between 0° and 360°. Thus, let con_d be a constant

of 360° for the determination of the borders for the horizontal field of view. We additionally define

$$l_{border} = \begin{cases} l_{border} + con_d & \text{if } l_{border} < 0 \\ l_{border} & \text{otherwise} \end{cases} \tag{4.24}$$

$$r_{border} = \begin{cases} r_{border} - con_d & \text{if } r_{border} > con_d \\ r_{border} & \text{otherwise} \end{cases} \tag{4.25}$$

to obtain $l_{border} \in [0, con_d]$ and $r_{border} \in [0, con_d]$. Analogous, we define two further border functions

$$f_{u_{border}}(ds) = \rho + \frac{v_{aov}}{2} \tag{4.26}$$

$$f_{lo_{border}}(ds) = \rho - \frac{v_{aov}}{2} \tag{4.27}$$

for obtaining the upper border $u_{border} \in [-90, 90]$ and lower border $lo_{border} \in [-90, 90]$ of the vertical field of view of the camera.

Now, with the help of these borders we are able to determine if a resource lies in the current field of view. Therefore, we have to map the bearing angle β (cf. Equation 4.18) onto the horizontal angle of view h_{aov} (cf. Equation 4.1) and the inclination angle θ on the vertical angle of view v_{aov} (cf. Equation 4.2). For instance, we illustrate the mapping of the bearing angle β in Figure 4.5 (a). The length of the blue arrow presents the mapped degree value of the current position of the resource x_{haov} on the horizontal angle. In Definition 4.8, we formally define the mapping of bearing angle β and inclination angle θ on the device's horizontal and vertical camera angle of view.

Definition 4.8. Let FOV be the set of all border parameters of the field of view $fov \in FOV$. Then, let $fov = (l_{border}, r_{border}, u_{border}, lo_{border}) \in FOV$ be a 4-tuple of field of view borders. Moreover, let RP be a set of points $rp \in RP$ on the angle of view of the camera, where rp is defined as a tuple (x_{haov}, y_{vaov}). Let con_x be a constant of constant of -1000° and let con_y be a constant of -180° in order to obtain a negative value if rp does not lie in the angle of view. We define two functions

$$f_{x_{haov}}(\beta, fov) = \begin{cases} \beta - l_{border} & \text{if } \beta \geq l_{border} \\ con_d - l_{border} + \beta & \text{if } \beta \leq r_{border} \\ con_x & \text{otherwise} \end{cases} \tag{4.28}$$

$$f_{y_{vaov}}(\theta, fov) = \begin{cases} u_{border} - \theta & \text{if } \theta \geq lo_{border} \text{ and } \theta < u_{border} \\ con_y & \text{otherwise} \end{cases} \tag{4.29}$$

where the coordinate $x_{haov} \in [0, h_{aov}]$ and the coordinate $y_{vaov} \in [0, v_{aov}]$ that determine a resource position within the camera's field of view in degrees.

To continue our example scenario, we already have the the horizontal and vertical angle of view of the camera h_{aov}= 54.8° and v_{aov}= 44.6°. Additionally, we assume that we have a camera azimuth of $\alpha = 346°$, a camera pitch of $\rho = 1°$, a device screen width of $screen_w$= 480 px, and a screen height of $screen_h$= 320 px. Thus, we have device sensor parameters ds=(346.0, 1.0,54.8,42.5,480,320)$\in DS$. Now, we determine the borders of the field of view through the following calculations:

$$l_{border} = 346° - \frac{54.8°}{2} = 318.6°$$

$$r_{border} = 346° + \frac{54.8°}{2} = 373.4°$$

$$r_{border} > 360° : r_{border} = 373.4° - 360° = 13.4°$$

$$u_{border} = 1° + \frac{44.6°}{2} = 23.3°$$

$$lo_{border} = 1° - \frac{44.6°}{2} = -21.3°$$

and obtain fov=(318.6,13.4,23.3,-21.3) $\in FOV$.

Now, let the bearing angle β be 342.96°. Considering Equation 4.28, we have $\beta \geq l_{border}$, so we proceed with

$$x_{haov} = 342.96° - 318.6° \approx 24.4°$$

and obtain $x_{haov} \in$ [0,54.8].

Then, we assume to have an inclination angle θ=0.6625°. Considering Equation 4.29, we have $\theta \geq lo_{border}$ and $\theta < u_{border}$, so we proceed with

$$y_{vaov} = 23.3° - 0.6625° \approx 22.6°$$

and obtain $y_{vaov} \in$ [0,44.6]. Finally, we obtain a point on the angle of view of the camera rp=(24.4,22.6) $\in RP$.

4.4.1.5 Screen Coordinates

As we need to properly align the virtual content and the resources of the current field of view on the screen of a device, we need screen coordinates in pixels to place the virtual content on

the respective position onto the screen. Therefore, we have to transform angle coordinates x_{haov} and y_{vaov} into screen coordinates (pixel units) which we define in Definition 4.9.

Definition 4.9. Let $screen_w$ be the screen width and let $screen_h$ be the screen height of $ds \in DS$. Moreover, let SP be a set of screen points $sp \in SP$ where a screen point sp is defined as a tuple (x_{cor},y_{cor}). Then, we define two functions

$$f_{x_{cor}}(rp, ds) = \frac{x_{haov} * screen_w}{h_{aov}} \tag{4.30}$$

$$f_{y_{cor}}(rp, ds) = \frac{y_{vaov} * screen_h}{v_{aov}} \tag{4.31}$$

where the coordinate $x_{cor} \in [0, screen_w]$ and the coordinate $y_{cor} \in [0, screen_h]$, to obtain the position of a resource on the screen of a mobile device.

If we consider again our example scenario, we already have device sensor parameters ds=(346.0,1.0,54.8,44.6,480,320) $\in DS$ and a resource point rp=(24.4,22.6) $\in RP$ on the angle of view. Now, we calculate

$$x_{cor} = \frac{24.4^\circ * 480\ px}{54.8^\circ} \approx 213.7\ px$$

$$y_{cor} = \frac{22.6^\circ * 320\ px}{44.6^\circ} \approx 162.2\ px$$

to finally obtain the screen point sp=(213.7,162.2) $\in SP$.

4.4.2 Tracking Algorithm

In the previous subsection, we introduced and defined all the calculations we need to obtain screen coordinates for the virtual content of our AR system. In this section, we put all the calculations into a single algorithm (cf. Table 4.3). Therefore, you may have to consider the previous section to clarify the meaning of each symbol. In order to map LOD resource locations to screen coordinates, we use the device location $dl = (\lambda_{dl}, \varphi_{dl}, \psi_{dl}) \in DL$, the loaded LOD resources $res = (\lambda_{res}, \varphi_{res}, \psi_{res}) \in RES$ of the system, and the device sensor parameters $ds = (\alpha, \rho, h_{aov}, v_{aov}, screen_w, screen_h) \in DS$ as input parameters for the algorithm. The output of the algorithm is a screen point $sp = (x_{cor}, y_{cor}) \in SP$ which presents the position on the screen at which a virtual object will be visualized. The input parameters which hold the sensor data for orientation and direction information ($\alpha \in ds$ and $\rho \in ds$) are updated constantly by the system. This is necessary because the visualized objects on the display have to stay aligned with the respective real-world resources to assure a proper AR experience.

Starting with line 1 of Table 4.3, we define a loop to guarantee that the system repeats the calculations for each resource. In lines 2 and 3, we calculate the distance in meters and

the bearing angle in degrees between the device location dl and the current resource res using Vincenty's inverse formula (cf. Definition 4.3- 4.5). The next calculation step (line 4) comprises the calculation of the inclination angle θ between the device dl and resource res by considering the value of their altitudes ψ_{dl} and ψ_{res} and by applying a basic trigonometry formula as defined in Definition 4.6 and Equation 4.21.

```
   input  : res, dl, ds, con_d
   output: sp = (x_cor, y_cor)
 1 foreach res ∈ RES do
 2     d(dl, res) = calculate distance;
 3     β(dl, res) = calculate bearing;
 4     θ(dl, res) = calculate inclination;
 5     l_border = subtract half of h_aov ∈ ds from α ∈ ds;
 6     if l_border < 0 then
 7         l_border = add con_d degrees to l_border;
 8     end
 9     r_border = add half of the h_aov ∈ ds to α ∈ ds;
10     if r_border > con_d then
11         r_border = subtract con_d degrees from r_border;
12     end
13     x_haov = con_x;
14     if β ≥ l_border then
15         x_haov = subtract l_border from current β;
16     end
17     else
18         if β ≤ r_border then
19             x_haov = take con_d, subtract l_border and add β;
20         end
21     end
22     x_cor = take x_haov multiply by screen_w ∈ ds and divide by h_aov;
23     u_border = add half of the v_aov to ρ ∈ ds;
24     lo_border = subtract half of the v_aov from ρ ∈ ds;
25     y_vaov = con_y;
26     if θ ≥ lo_border then
27         if θ < u_border then
28             y_vaov = subtract θ from u_border;
29         end
30     end
31     y_cor = take y_vaov multiply by screen_h ∈ ds and divide by v_aov;
32 end
```

TABLE 4.3: AR tracking algorithm

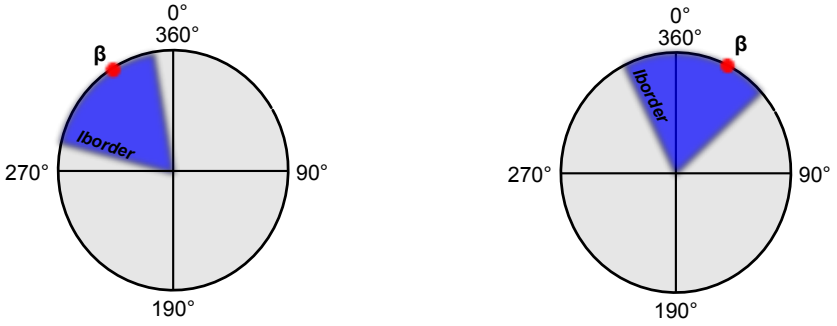
In lines 5-22, we define the calculation steps to obtain the x-screen coordinate. Starting with lines 5-12, we determine the left and right borders of the field of view as defined in Definition 4.7.

When we move a device left or right, the full range of heading reaches from 0° to 360°. But, the result value of the left border calculation may be a negative value (line 6). Thus, let con_d be a constant value of 360°, we add con_d to the left border value (line 7) to obtain a border value that lies in the range of [0°, 360°] (cf. Equation 4.24).

On the other hand, the right border calculation may lead to a value that is greater than 360° (line 10). Then, we subtract 360° from the right border value in line 11 (cf. Equation 4.25).

Continuing with line 13 of Table 4.3, we declare the variable x_{haov} and initialize it with con_x. con_x presents a constant value of -1000°. The negative value assures that no object is visualized on the screen, if the bearing angle does not lie between the left and right border of the field of view. That means that the resource is not visible and will not be visualized by the system.

With the help of the left and right borders l_{border} and r_{border} and the bearing angle β we are able to check if a resource lies within the field of view fov. Thus, in line 14 we check if the bearing angle β is greater than or equal to the left border, and then we map the bearing angle onto the horizontal angle of view as defined in Definition 4.8 to obtain the position of the resource on the horizontal angle of view. In Figure 4.6 (a), we illustrate an example case to this calculation step with the help of a compass rose and the respective heading degrees. The field of view is highlighted with a blue color. The red dot presents the value of the bearing angle. In this example, the bearing angle comprises a value that is greater than the left border of the current field of view. This means that the if-clause of line 14 is true.



(a) $\beta \geq l_{border}$ (b) $\beta < l_{border}$

FIGURE 4.6: Bearing angle within h_{aov}

In the case the bearing angle is not greater than the left border of the field of view, we additionally check if the bearing angle is lower than or equal to the right border of the field of view (line 18). Then, we obtain the position of the resource on the horizontal angle of view x_{haov} by subtracting the left border value from 360° and adding the bearing angle value in line 19, as defined in Definition 4.8. Figure 4.6 (b) illustrates this case: the bearing angle lies between 0° and the right border. Thus, β is lower than the right border. After the position of the resource on the horizontal angle of view is determined, we perform the transformation into the x-screen coordinate in line 22 of Table 4.3 (cf. Definition 4.9).

The last part of the algorithm comprises the y-screen coordinate computation. Starting with lines 23 and 24, we determine the upper and lower borders of the current field of view (cf. Definition 4.8). As before, we declare the variable y_{vaov} with a constant variable con_y = -180° (line 25). In consequence, we obtain a negative y-pixel value, if no re-initialization follows in the following lines. For determining the position of a resource on the vertical angle, we take the inclination angle and check if it is greater than or equal to the lower border, and we additionally check in line 27, if the inclination angle is lower than the upper border because then the resource lies within the current vertical viewing angle and we are able to map the inclination angle on the vertical angle of view (cf. Equation 4.29). Finally, we determine the y-screen coordinate y_{cor} in line 31 (cf. Definition 4.9).

4.5 Functional Model

This part gives an overview of the functional processes of the mobile AR system. Thereby, we provide three different activity models. The first model presents the control flow until the system visualizes virtual content within the AR interface, and the remaining models contain the control flow for loading content on a 2D map.

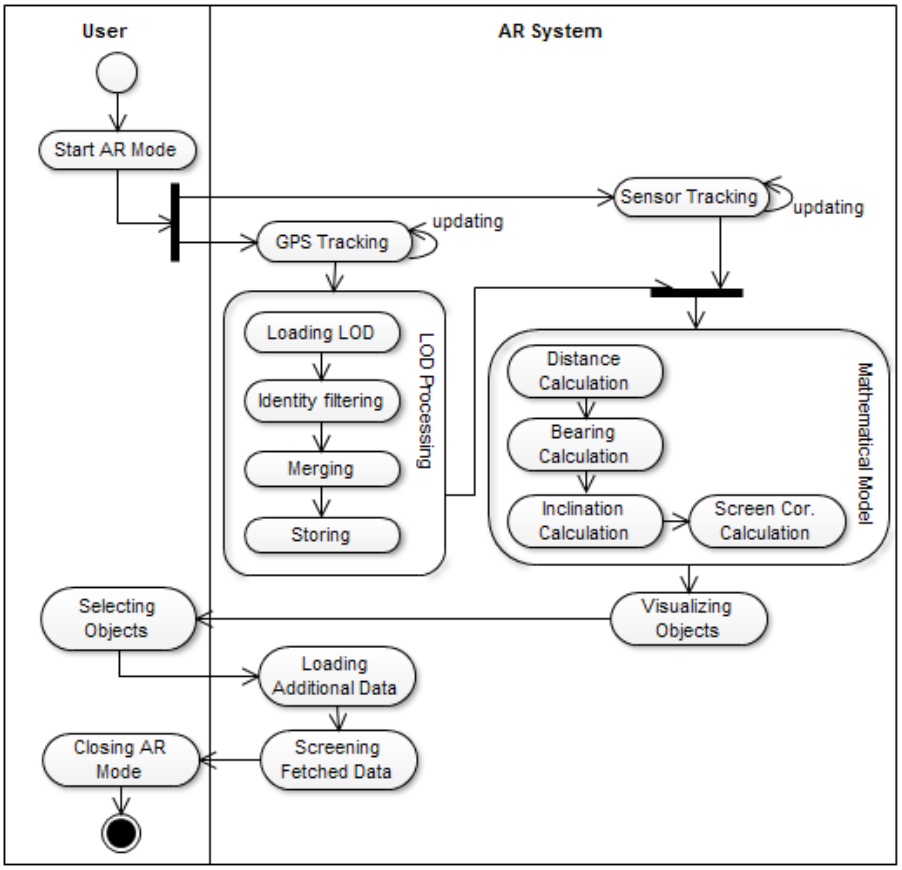


FIGURE 4.7: Functional model – loading content in AR interface

The model in Figure 4.7 is separated into two parts: the left side of the model contains activities that are triggered by the user and on the right side we present activities which are triggered by the AR system. Beginning at the top of Figure 4.7, the user selects the AR mode of the system. This provokes two activities simultaneously: the start of the sensor listening and the GPS tracking. The access to the device's sensors is necessary for recording the orientation and direction values of the device. We use the sensor data later within the mathematical model for the visualization of the AR content.

The GPS radio determines the current location of the device. At the first GPS fix, the loading of Linked Data resources starts. Once RDF data is available, the system filters identical resources to prevent duplicate resource occurrences and then they are merged and stored locally on the device. After the LOD processing, the system executes the mathematical model calculations: bearing, distance, inclination, and x/y coordinate calculations. Afterwards, the system visualizes virtual content on the screen. Now, the user has the possibility to select one of the virtual objects. The selecting triggers the loading of further LOD resource descriptions about the respective resource. In our application scenario, we load further resources description from the DBpedia data set (cf. Chapter 5). Our AR system displays the loaded RDF-based DBpedia resource descriptions on the screen, so that the user is able to observe additional information about specific resources. In addition, the user has the possibility to close the AR mode of the system.

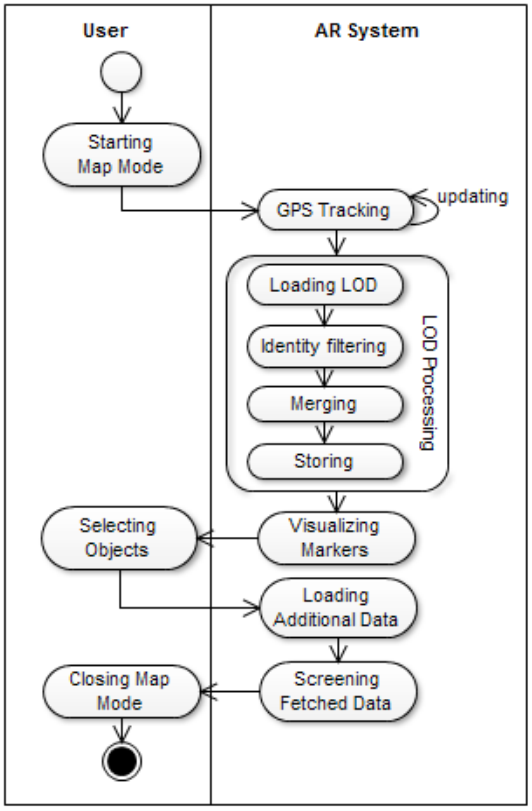


FIGURE 4.8: Functional model – loading LOD in map view

Figure 4.8 presents the flow of processes for loading virtual content on a 2D map. As soon as the user activates the map mode of the system, the system provokes the GPS radio to obtain the current location of the device. At the first location fix, the downloading of LOD resources for a specific area starts. In this case, the system does not use the mathematical model to calculate screen coordinates. This time, the system visualizes markers on the map after the process of downloading LOD resources is completed. As soon as markers are visible on the map, the user has the possibility to select a marker. The selection triggers again a downloading process of LOD data. Then, the system displays the loaded data on the screen. In the end, the user is able to close the map mode.

The user has also the possibility to load tours that are located near the device. In Figure 4.9, we illustrate the tour mode where the user has to set a range (km) in which tours will be searched, after she/he starts the tour mode. Then, the system triggers the GPS tracking to find a first fix. As soon as the location of the device is determined, the system searches after GPS tours at OpenStreetMap and presents received results in a list on the screen. Now, the user is able to select a GPS tour of the list. If a tour is selected, the system presents the tour on the map. Finally, the user is also able to close the tour mode of the system.

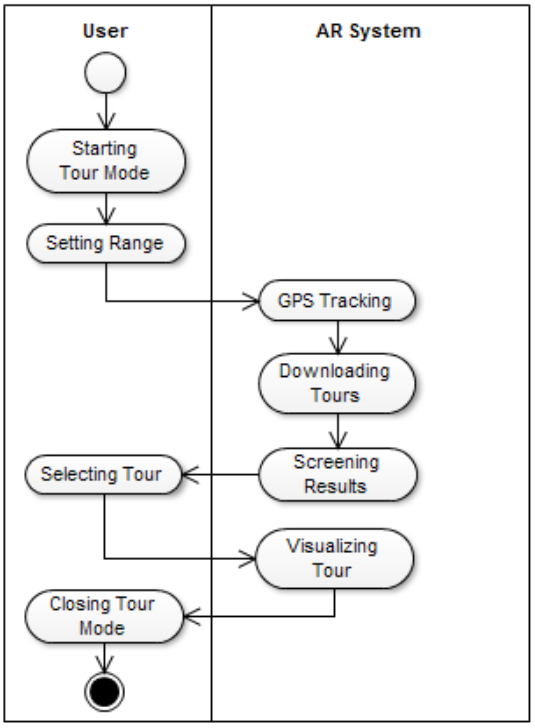


FIGURE 4.9: Functional model – loading tours in map view

4.6 Summary

Within this chapter, we presented design consideration and requirements for a mobile AR system. We presented this AR system on a conceptual basis where we outlined, amongst other

things, that we use three different Linked Data sets to access Linked Data resources: GeoNames, LinkedGeoData, and DBpedia. Whereby, we access GeoNames and LinkedGeoData resources during the loading process and thus use these resources as our AR content. If we need further information about certain resources, we fetch the data from DBpedia.

Fetching data from different sources causes the availability of duplicate resources. Thus, we described the Silk framework which includes Silk LSL a language that allows us to define rules for discovering related resources, as possible solution for avoiding duplicate data. Subsequently, it is possible to eliminate the discovered duplicate entries and merge the remaining resource descriptions into a single RDF graph. For our prototypical implementation of our AR system we do not use Silk, but we realized similar methods as the Silk framework provides to identify duplicate resources which we present in Chapter 5.

Furthermore, we outlined again that our AR system is based upon a sensor-based mathematical model. The model comprises several calculation steps which altogether compose an AR tracking algorithm. The algorithm requires several input parameters: the geographical locations of the LOD resources, the GPS location of the device, and sensor data representing heading and orientation information of the device. The model is then capable of determining the current angle of view and to check, if real-world resources lie within this angle of view. This is necessary in order to finally be able to calculate screen coordinates for every resource within the captured view, so that virtual content can be placed on top of the resources in the camera picture.

Chapter 5

Proof of Concept

This chapter describes the implementation process of the proof-of-concept Android AR application, based on the formal and conceptual constituents presented in Chapter 4. First, we start with the presentation of UML class diagrams of the AR prototype. These diagrams present an overview of the classes of the system and provide a short introduction into our AR system. Finally, we give a description of the most important implementation parts of our AR prototype including a precise look into specific classes and sequence processes, and a presentation of actual source code snippets.

5.1 Software Design

We explain the software design of our proof-of-concept prototype in this section by presenting class diagrams. Thereby, we begin with the presentation of an overview diagram and we describe the core classes of the system. Then, we present three diagrams for each mode (AR-, map-, and tour-mode) the prototype comprises.

We divide classes within the following UML class diagrams into different groups, whereby each group is marked by a different color. Yellow stands for classes which extend an activity, the red ones present helper classes that support other classes of the application, green stands for classes which extend an Android *View* (we describe views in Definition 5.1 according to the definition provided by [56]) and the purple one extends also the `View` class but represents a LOD resource. The attributes and methods of the classes are not presented and discussed in this section. But we further discuss the classes, which are most relevant for the realization of our approach, in Section 5.2 within the respective implementation parts of the prototype.

> **Definition 5.1** (View)**.** *Views are user interface (UI) elements that form the basic building blocks of a user interface. A view could be a button, label, text field, or many other UI elements. [...] Views are also used as containers for views, which means that there's usually a hierarchy of views in the UI.* (Komatineni et al., 2011)

5.1.1 Activities Overview

Figure 5.1 comprises all the activities our prototype includes and exhibits their relations to each other. Two helper classes are also presented because they are used to start activities within the application. In this figure, we additionally depict the sections in which we further discuss the specific classes. However, to understand the purpose of each presented class, we briefly describe them within the following paragraphs.

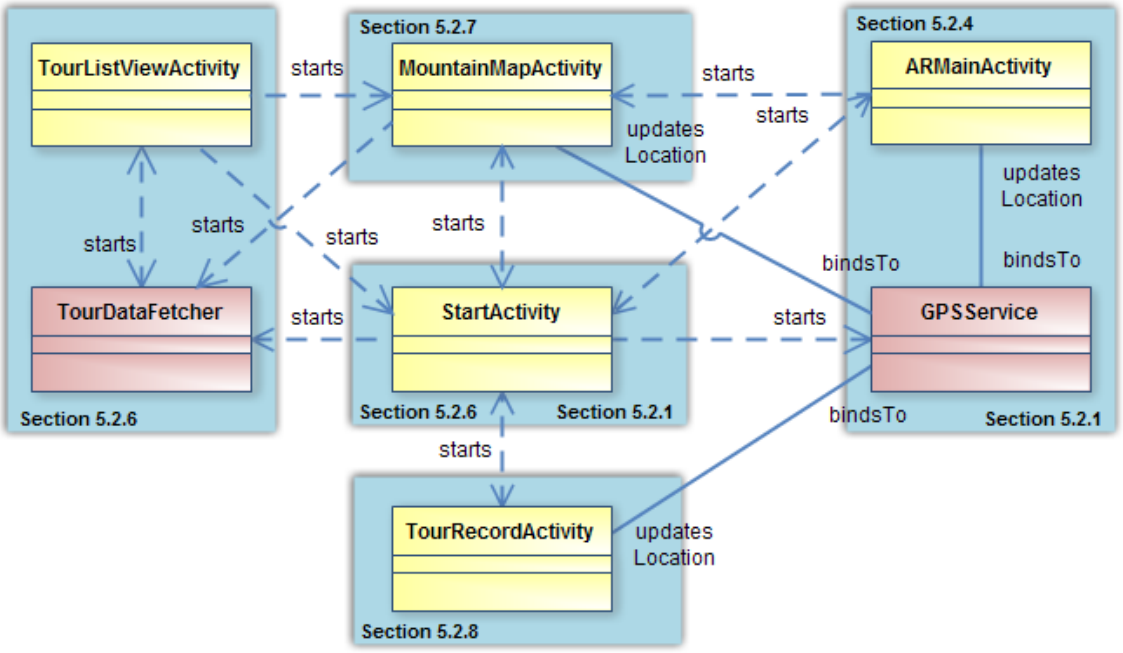


FIGURE 5.1: Core classes of AR prototype

StartActivity: Is the main activity of the application and presents the entry point to the AR application. It contains the implementation of the UI which appears on the display as soon as the user activates the application. The user has the possibility to start three different activities through this class: `ARMainActivity` to switch to the AR mode, `MountainMapActivity` to switch to the map mode, and `TourRecordActivity` to switch to the tour mode. `StartActivity` also triggers the `TourDataFetcher` to gather GPS tours from the Web and it activates the `GPSService`.

GPSService: Represents an Android service that is responsible for providing an activity, which binds itself to this service, with location updates. Therefore, this class implements Android's `LocationManager` and the `LocationListener` to be able to access the built-in GPS radio of the mobile device and to receive constantly location updates (cf. Section 5.2.1.1). The system starts `GPSService` as soon `StartActivity` is activated. But the receiving of location updates is not started until a certain activity binds itself to `GPSService`.

ARMainActivity: This class is the core heart of the AR mode of the prototype, which defines the AR interface for the displaying of recognized mountains in the near vicinity and it implements the access to the device's sensors. This activity binds itself to `GPSService` to

constantly obtain GPS location updates. Within the AR mode it is possible to activate the map mode (starting `MountainMapActivity`). Leaving this activity causes the renewed activation of `StartActivity`.

MountainMapActivity: This activity presents a map-viewing activity. We create map-based activities by using the Google Maps Android API[1]. Thus, this class displays a Google Maps interface element that allows us to display resources through special markers on it. `MountainMapActivity` enables the activation of `TourDataFetcher` for loading GPS tours from OpenStreetMap but also the renewed activation of `StartActivity`. Moreover, it receives location updates from the `GPSService`.

TourDataFetcher: This class extends the abstract class `AsyncTask`. It contains event handlers that allow to display a progress dialog on the screen to inform the user that the system currently fetches tour data from OpenStreetMap. After the searching of tours is finished the system starts the `TourListViewActivity` which is responsible for presenting the results on the screen.

TourListViewActivity: This class extends `ListActivity` which enables to display the results (received GPS tours) in a list, after the `TourDataFetcher` has finished the GPS tour searching process. Every item on this list is selectable. Selecting an item in the list causes the activation of `MountainMapActivity` and the system visualizes the tour on a map. `TourListViewActivity` also allows the activation of the `TourDataFetcher` for starting a new searching process to gather further GPS tracks. It is also possible to move back to `StartActivity`.

TourRecordActivity: Presents also a map-based activity. This class enables the visualization of a currently recorded GPS tracks on a map and to store these recorded tracks in a GPX file. This class also enables the displaying of GPS tracks which are stored in a GPX file within the local storage system of the device. This class binds itself to `GPSService` to receive location updates every time the user moves and it is possible to move back to the home screen by activating `StartActivity` again.

5.1.2 AR Mode

The class diagram in Figure 5.2, illustrates all the classes the system requires in the AR mode. We shortly introduce each class in the following paragraphs:

ARView: It contains the implementation of our AR tracking algorithm (Table 4.3). Thus, this class is responsible for visualizing virtual content (LOD resources which are instances of `MountainObject`) on top of the camera view. In Figure 5.2, we illustrate that `ARView` is able to comprise none or several objects of type `MountainObject`.

CamView: The purpose of this class is to implement the camera preview. Presents the base view to be able to visualize objects on top of it.

[1]Google Maps Android API : `https://developers.google.com/maps/documentation/android/`

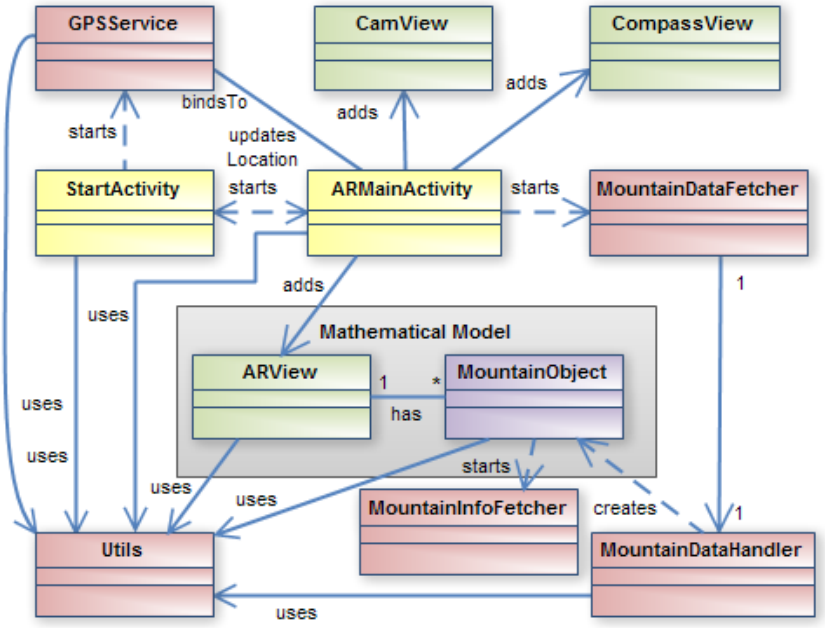


FIGURE 5.2: Class diagram overview – AR mode

CompassView: Implements an animated compass which appears on the left bottom side of the screen.

In Figure 5.2, we use the word "adds" for the associations between the `ARMainActivity` class and the view classes because in Android an activity adds views to the content view in order to generate an UI. In our AR system, we need an AR interface in which resources are visualized on top of the live camera view. Thus, we initialize `ARView`, `CamView`, and `CompassView` within `ARMainActivity`. `CamView` presents the content view and we add the two remaining views to this content view. As a result, `CamView` lies below `ARView` and `CompassView` which means that virtual content is placed upon the live camera preview. Figure 5.3 illustrates this method of piling views onto each other. These three layers constitute the AR interface of the prototype.

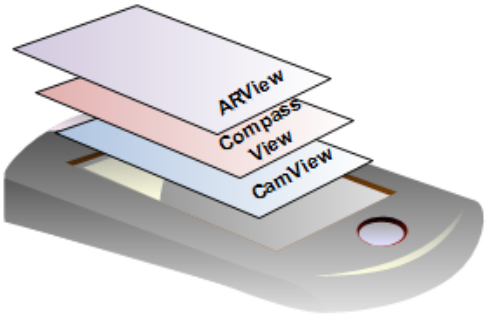


FIGURE 5.3: Views of the AR interface

MountainObject: Represents a mountain resource loaded from LOD sets. We create objects of type `MountainObject` within the `MountainDataHandler` class as soon as LOD resources are available. `MountainObject` also comprises calculations of the mathematical

model of which the results serve as input parameters for the tracking algorithm in the `ARView` class. Moreover, this class comprises a drawing method for drawing canvas objects on the right position of the screen.

Utils: This class comprises important helping methods used by other classes for instance methods which check the availability of the Internet connection. Moreover, `Utils` extends a SQLite[2] helper class to manage database creation for the loaded LOD resources. Thus, it implements methods for creating, opening, and upgrading the SQLite database on our mobile device but also methods for performing functions (e.g. adding new resources) on this SQLite database. In Figure 5.2, we illustrate which classes are in need of methods implemented in `Utils`.

MountainDataFetcher: The `MountainDataFetcher` extends the abstract class `AsyncTask`. Event handlers are used to display a progress dialog until a background process is finished. Within the background process the system triggers the `MountainDataHandler`. As visible in Figure 5.2, the system calls the `MountainDataFetcher` within the main activity `ARMainActivity` and the `MountainDataFetcher` makes use of methods of the `MountainDataHandler` during its background process.

MountainDataHandler: In this class, we define methods for downloading RDF data from the GeoNames and LinkedGeoData data sets. Additionally, `MountainDataHandler` is responsible for filtering identical resources, merging resource descriptions, and for saving resources as objects of type `MountainObject`. This class also saves the filtered resources within a SQLite database on the device, so that the application is later executable without a permanent network connection (cf. Section 5.2.3).

MountainInfoFetcher: Extends the abstract class `AsyncTask` for fetching further information about specific mountains. It is activated by the `MountainObject` class. A progress dialog indicates that in the background data is fetched from DBpedia. The system displays the loaded results within a dialog on the screen.

5.1.3 Map and Tour Mode

As our prototype presents an AR mountain guide, we included a map and tour mode to the prototype. In Figure 5.4 and in Figure 5.5, we present all classes our AR prototype additionally comprises in order to display a map to present mountain markers on it but also classes which enable the loading, screening, and recording of GPS tours. We already explained many of these classes in the previous sub-section. Thus, we only discuss the remaining classes within this section.

As visible in Figure 5.4, we use Android *Overlays* (we describe overlays in Definition 5.2 according to the definition provided by [57]) to annotate a map and to handle the user input (touch event).

[2]SQLite: a lightweight relational database for data storage, cf. `http://www.sqlite.org/`

Definition 5.2 (Overlay)**.** *Overlay is the class used to annotate your maps. Using Overlays, you can use a Canvas to draw onto any number of layers that are displayed on top of a Map View.* (Meier, 2010)

In Figure 5.4, we use again the word "adds" for the associations between the class `MountainMapViewActivity` and the overlay classes of our AR system because we have to add these overlays to the map view instance within `MountainMapViewActivity`. Then, the system is able to display mountain markers on top of the map.

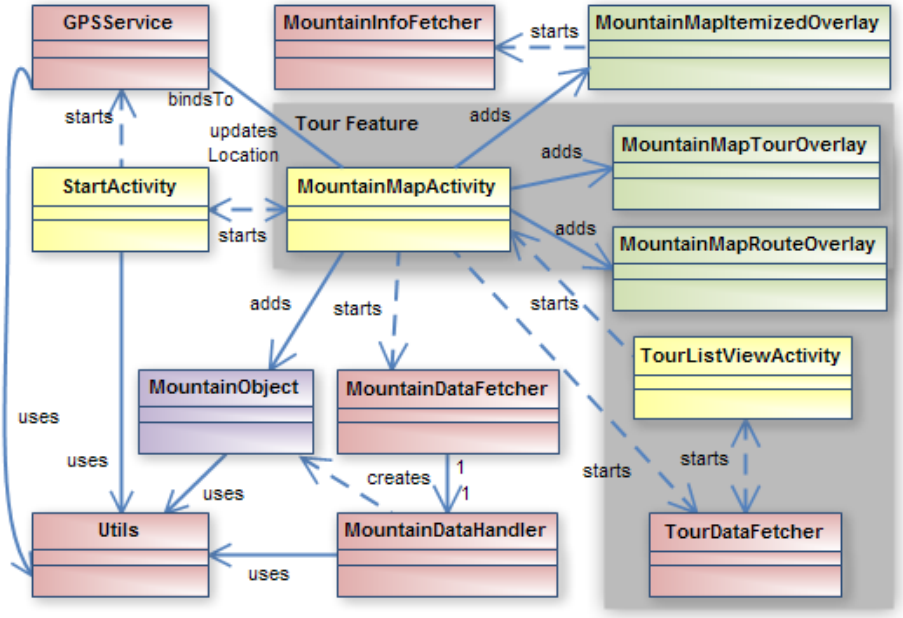


FIGURE 5.4: Class diagram overview – Map and tour mode

In our system, we define 4 different overlays which the system is able to add to the map view:

MountainMapItemizedOverlay: This class is responsible for visualizing markers onto a `MapView` (implemented inside the `MountainMapActivity`) for each object of the class `MountainObject`. It is also responsible for displaying the current user position on the map and it implements a method that handles touch events. If a user selects a marker, a dialog appears on the screen which contains information about the selected resource. The system uses, as in the AR mode, the `MountainInfoFetcher` for the loading of additional resource descriptions.

MountainMapTourOverlay: This class is responsible for visualizing a GPS tour track on a map by drawing a line between the geographical points of the track.

MountainMapRouteOverlay: This overlay class is responsible for additionally visualizing the route to the start point of the current displayed tour.

TourRecordOverlay: If the user records a tour, this class is responsible for visualizing the already passed path on a map by adding a line between the last recorded geographical point and the new geographical point received from `GPSService` on the map.

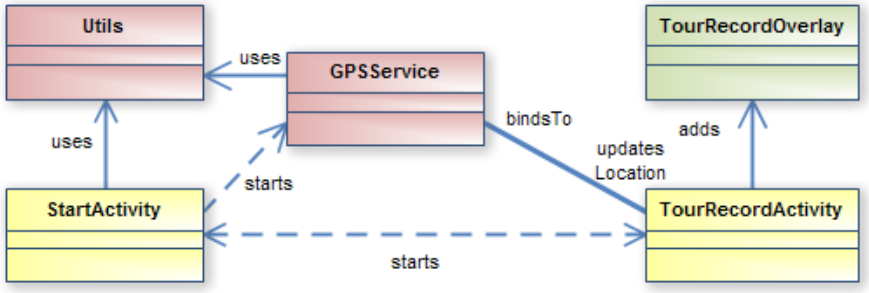


FIGURE 5.5: Tour recording mode

5.1.4 Comparison of Approach and Proof of Concept Design

To consider our approach of Chapter 4, we illustrate in Figure 5.6 and Figure 5.7 the components of our conceptual model of an AR system (cf. Figure 4.2) in respect of the classes designed for our proof-of-concept AR application.

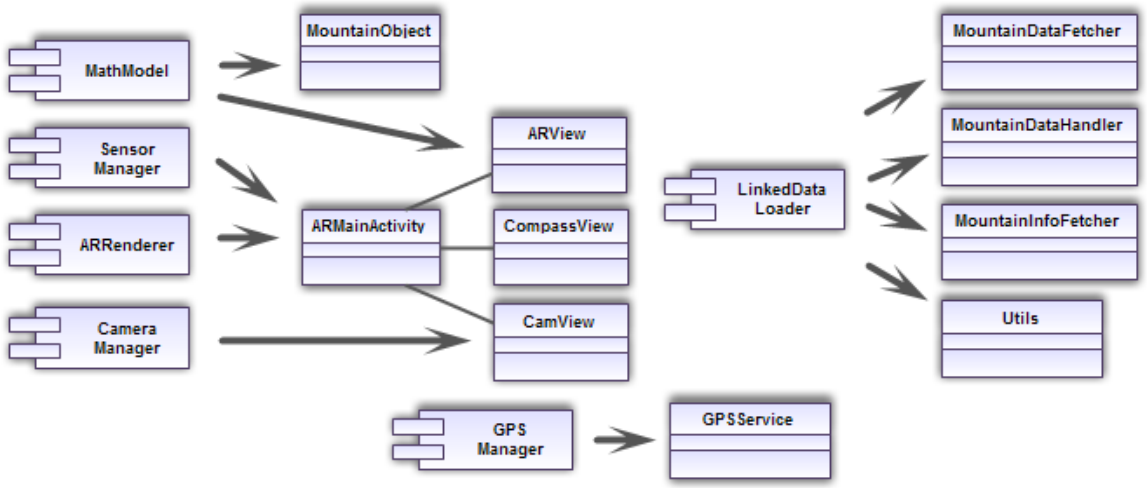


FIGURE 5.6: Comparison of AR mode components and classes

As we already explained earlier, we define the calculations of the mathematical model within the `MountainObject` and `ARView` classes. Thus, we realized the component *MathModel* through implementing these two classes. `ARMainActivity`, `ARView`, `CompassView`, and `CamView` compose the AR interface of our AR system and thus comply with the responsibilities of the *ARRenderer* component. Additionally, the `CamView` class takes over the responsibilities of the *CameraManager* component. The tasks of the *SensorManager* component are implemented within the `ARMainActivity` class, and the *GPSManager* component is realized through the `GPSService` class.

Finally, the tasks of the *LinkedDataLoader* component are handled by the classes `MountainDataFetcher`, `MountainDataHandler`, `MountainInfoFetcher`, and `Utils`. These classes comprise all the methods for loading, filtering, processing, merging, and storing LOD resources and for fetching further descriptions about specific resources if requested.

Figure 5.7 comprises the remaining two components *MapManager* and *TourManager* of our conceptual AR system of Chapter 4. We realized these two components through the implementation of the classes `MountainMapActivity`, `TourRecordActivity`, `TourListViewActivity`, and 4 different overlay classes (summarized as `Overlays` class) in our proof-of-concept AR prototype.

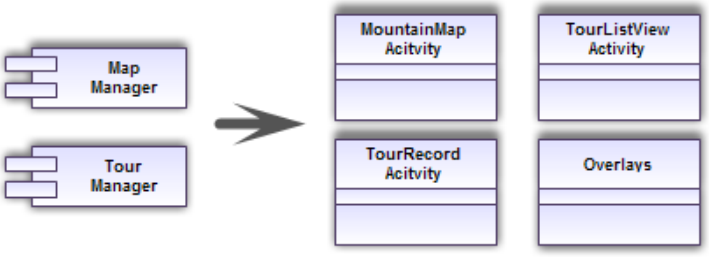


FIGURE 5.7: Comparison of map/tour mode components and classes

5.2 Software Implementation

This section contains the most relevant implementation parts of our AR prototype. Thereby, we present implementation details of the main activity of the application, methods of how we access GPS data, as well as sensor data for the estimation of orientation and direction of the device, and the realization of the mathematical model. We also present the methods that allow us to access Linked Data sets of the Web. In the last part of this section, we describe the realization of tour features such as presenting tours on a 2D map or recording tours.

Most of the implementation sections include a class diagram and a sequence diagram to provide a look into specific process flows of our system. We present only the most relevant attributes and methods of the presented classes, and the sequence diagrams do not contain every method that is actually executed. We only provide a rough overview of process flows of the different implementation scenarios.

5.2.1 Localization and Orientation

In this section, we explain the components of the system that are responsible for determining the current location on earth and the orientation of the device (cf. Figure 5.8).

StartActivity – Attributes/Methods

The class `StartActivity` of Figure 5.8, comprises the method `onCreate()` which is called when the activity is starting. The `onCreate()` method contains for instance the activation of `GPSService`. Within `onCreate()` we also define a method for starting the activity `ARMainActivity`. We use the two other methods `onPause()` and `onResume()` to handle actions in the case the user leaves and resumes the AR application.

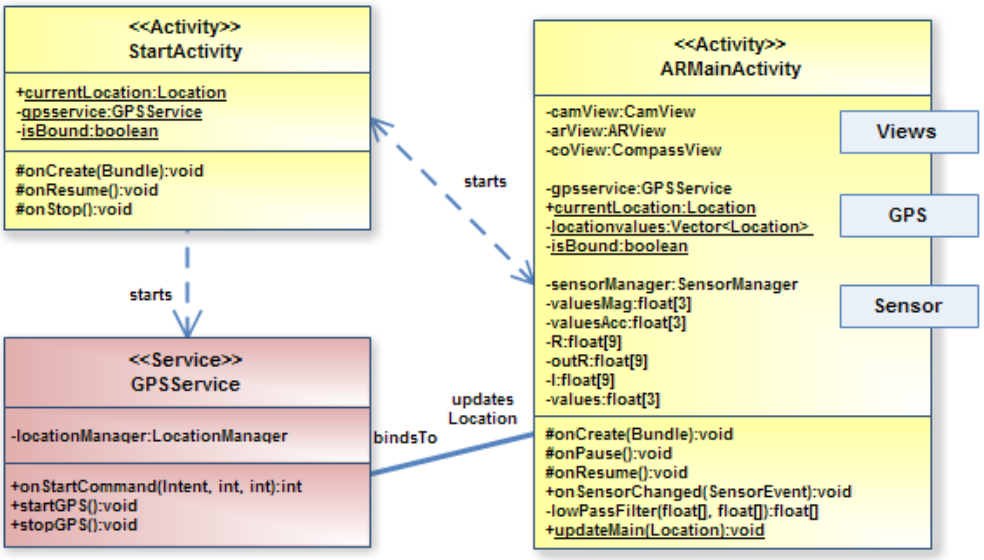


FIGURE 5.8: Classes responsible for determining localization and orientation of device

ARMainActivity – Attributes/Methods

As visible in Figure 5.8, we separate the attributes of the class `ARMainActivity` in different groups: *Views*, *GPS*, *Sensor*. Attributes of group *Views* present objects of the different views of the AR system. `ARMainActivity` is used to set up the AR interface of the application. An UI is provided by one or more view objects which are derived from the `View` class. Our system includes 3 different views: `CamView`, `ARView`, and `CompassView`. Within `onCreate()` the method `setContentView()` inflates the activity's UI. The content view of our system is the camera preview and the two other views are added to the content view.

Furthermore, we use the attributes of group *Sensor* for the process of reading sensor values because `ARMainActivity` implements a method for accessing the accelerometer and magnetic field sensors to retrieve values for the determination of the current heading and orientation of the device (cf. Section 5.2.1.2).

Finally, the attributes of the third group *GPS* are responsible for the location parameters delivered by `GPSService`. `gpsservice` is an instance of `GPSService` and is used for starting (`gpsservice.startGPS()`) and stopping (`gpsservice.stopGPS()`) location updates. `isBound` is a boolean variable which we use to check if the activity is currently bound or not bound to `GPSService`. The method `updateMain()` is triggered after every location update of `GPSService` (cf. Section 5.2.1.1). This method implements functions for handling the new location (e.g. `currentLocation` is re-initialized with the new location) and to trigger data downloads of Linked Data sets (cf. Section 5.2.2).

We use the methods `onCreate()`, `onPause()`, and `onResume()` for handling the life-cycle of this activity (cf. Section 2.4.3). We describe the methods `onSensorChanged()` and `lowPassFilter()` in Section 5.2.1.2.

GPSService – Attributes/Methods

`GPSService` contains an attribute `locationManager` which is an instance of the `LocationManager` class. The location manager provides access to the system location service and implements a location listener. The location listener receives notifications from the location manager the moment the location changes [60], and notifies the current activity periodically about the location updates. Thus, we use the attribute `locationManager` to initialize the location manager and to register for location updates as soon an activity asks for the current GPS location. If the service is started, the system calls `onStartCommand()`. We use the `startGPS()` and `stopGPS()` methods for registering and unregistering for location updates. In the next section, we give more information about the class `GPSService` and explain the most relevant methods of this class.

5.2.1.1 Reading GPS Coordinates

In this section, we explain the process of reading GPS coordinates by presenting a sequence diagram (cf. Figure 5.9). This diagram illustrates the interactions between the three classes of Figure 5.8 arranged in time sequence. We also present some of the methods as code snippets. In Figure 5.9, we define the interactions with identification numbers and we describe these interactions within Table 5.1.

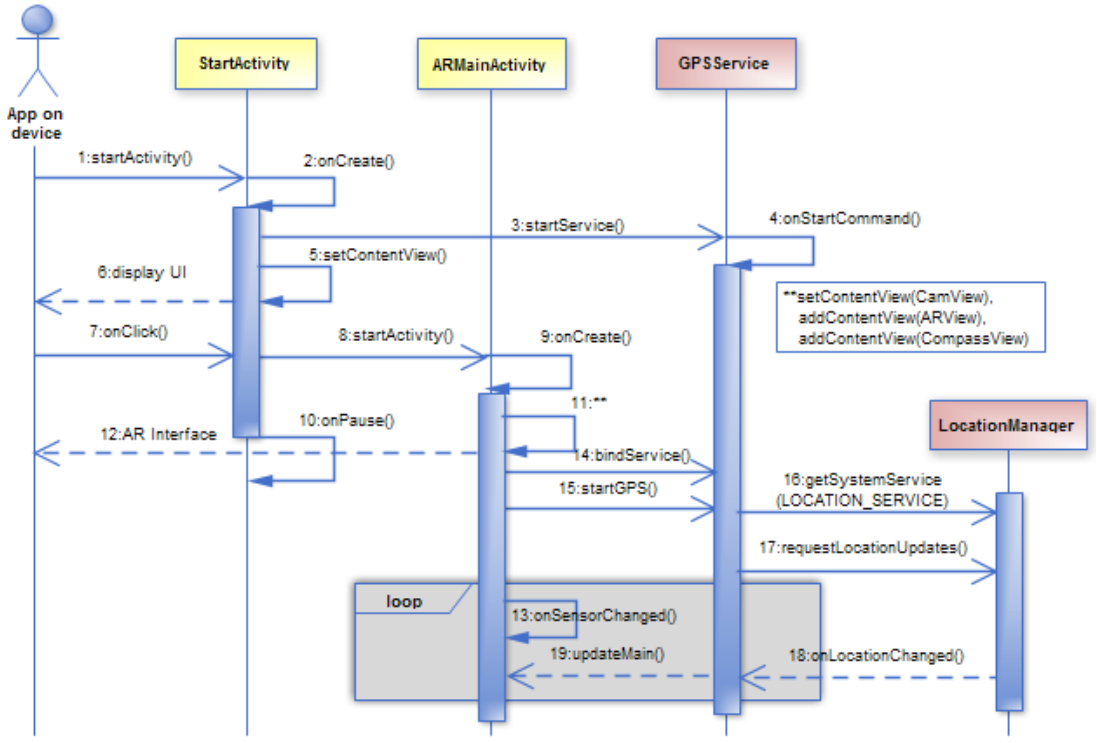


FIGURE 5.9: Localization process

ID	Description
1	The user activates `StartActivity` by launching the AR application on the device.
2	`onCreate()` initializes the activity.
3	`StartActivity` starts the service `GPSService`.
4	This method is called the moment the service is started.
5,6	To define an UI for the application we call `setContentView()`. Then, the system displays the desired content view.
7,8,9	The user selects a mode and triggers the start of `ARMainActivity` which is initialized by `onCreate()`.
10	`onPause()` is called when the system is about to start `ARMainActivity`.
11,12	We set the content view and add additional views to build the AR interface (cf. Figure 5.3).
13	Is used to monitor sensor values and is called each time the sensor values change. (cf. Listing 5.3)
14	We use `bindService()` to obtain a persistent connection to `GPSService`.
15	`startGPS()` (cf. Listing 5.1) triggers the location updates from `GPSService`.
16, 17	We request an instance of `LOCATION_SERVICE` to be able to access the `LocationManager` in order to request location updates by the use of a location listener (cf. Listing 5.1).
18,19	`onLocationChanged()` is called for each location update and it triggers the `updateMain()` method to inform `ARMainActivity` about the new location update. We put these two methods in a loop box because they are repeated as long as new location updates are received.

Table 5.1: Description of interactions from Figure 5.9

In Listing 5.1, we present the method `startGPS()` of `GPSService`. It contains the initialization of the `LocationManager` with the Android system service `LOCATION_SERVICE`. The `LocationManager` registers the `LocationListener` called `gpsListener` with the method `requestLocationUpdates()`.

```
1 public void startGPS() {
2  LocationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);
3  LocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,100,1,gpsListener);
4 }
```

Listing 5.1: A code snippet of `GPSService` – Registration for location updates

The `requestLocationUpdates()` method requires four parameters comprising the name of the GPS location provider the application uses (`LocationManager.GPS_PROVIDER`), the minimum time (100 ms) followed by the minimum distance change (1 meter) before the next update is made, and at least, the name of the `LocationListener` itself. A `LocationListener` comprises 4 methods. One of them is `onLocationChanged()`. It presents a callback method called by the `LocationManager` every time the location changes. The parameter of this

method is a new `Location` object that contains, among other things, information about longitude, latitude, and altitude values of the current location.

5.2.1.2 Accessing Accelerometer and Magnetic Field Sensors

The AR system requires information about heading and orientation of the mobile device to be able to accurately register visualized resources in the AR interface. We are able to access orientation and heading information of specific sensors offered by Android mobile phones.

In our AR system, we implement the sensor access within the `ARMainActivity` class. In Listing 5.2, we present the initialization of the `SensorManager` with the Android system service (`SENSOR_SERVICE`) to be able to access the sensors. The `SensorManager` registers the `SensorEventListener` for a specific sensor with the method `registerListener()`. `SENSOR_DELAY_UI` specifies an update rate for UIs [57].

```
1 sensorManager= (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
2 sensorManager.registerListener(this,sensorManager.getDefaultSensor(Sensor.
      TYPE_MAGNETIC_FIELD),SensorManager.SENSOR_DELAY_UI);
3 sensorManager.registerListener(this,sensorManager.getDefaultSensor(Sensor.
      TYPE_ACCELEROMETER),SensorManager.SENSOR_DELAY_UI);
```

Listing 5.2: A code snippet of `ARMainActivity` – Registration of sensor listener

A `SensorEventListener` requires the two methods: `onAccuracyChanged()` and `onSensorChanged()`. In our work, we concentrate on the `onSensorChanged()` method (cf. Listing 5.3) which the system calls every time the sensor values of the device change.

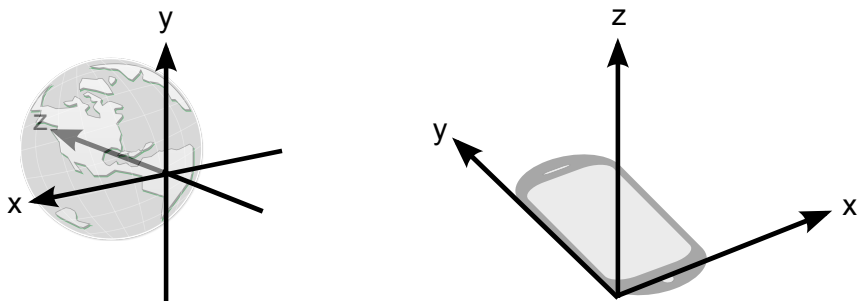
```
1  public void onSensorChanged(SensorEvent event) {
2    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
3      valuesAcc = event.values.clone();
4    if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
5      valuesMag = event.values.clone();
6    filtvaluesMag = lowPassFilter(valuesMag, filtvaluesMag);
7    filtvaluesAcc = lowPassFilter(valuesAcc, filtvaluesAcc);
8    SensorManager.getRotationMatrix(R, I, filtvaluesAcc, filtvaluesMag);
9    SensorManager.remapCoordinateSystem(R,SensorManager.AXIS_X,SensorManager.AXIS_Z,outR);
10   SensorManager.getOrientation(outR, values);
11   azimuth = (float) Math.round(Math.toDegrees(values[0]));
12   pitch = (float) Math.round(Math.toDegrees(values[1]));
13   if (azimuth < 0)
14     azimuth += 360;
15   if (pitch != 0)
16     pitch *= -1;
17   arView.updateIncomingSensorValues(azimuth, pitch);
18   coView.updateIncomingSensorValues(azimuth);
19 }
```

Listing 5.3: A code snippet of `ARMainActivity` – `onSensorChanged()`

The parameter of the `onSensorChanged()` method in Listing 5.3 is a `SensorEvent` variable which holds the sensor data as well as the sensor type. Thus, it is possible to check which sensor

object actually triggers the event (lines 2 and 4). After the sensor type is identified, the sensor data can be fetched by using the values of the passed event. As the sensors of type magnetic field and accelerometer deliver values for x-, y- and z-axes, sensor data is passed to float arrays with a length of 3 (line 3 and 5). Before we further process the sensor values, they undergo a low pass filtering (lines 6 and 7). We explain the reason for the filtering of the sensor data later within this section.

In line 8 of Listing 5.3, we define the `getRotationMatrix()` method which enables the transformation of device coordinate system values to world coordinate system values. In the world coordinate system (cf. Figure 5.10 (a)), the x-axis roughly points East being tangential to the ground, the y-axis is again tangential to the ground at the current phone position but points towards the magnetic North Pole and finally, leaving the z-axis pointing to the sky being perpendicular to the ground [60]. In the coordinate system of the device (cf. Figure 5.10 (b)), the x-axis is horizontal, the y-axis vertical, and the z-axis points out of the screen.



(a) World (b) Device

FIGURE 5.10: Coordinate systems

The `getRotationMatrix()` method requires four parameters, where the rotation matrix `R` and the inclination matrix `I` present output arguments (are filled with values when this function returns) and the previously filtered accelerometer and magnetic field values (`filtvaluesAcc`, `filtvaluesMag`) are used as input arguments. The whole application is fixed in landscape mode. Thus, the x-axis of the device's coordinate system is vertical and points up instead of the y-axis. Android offers the `remapCoordinateSystem()` method for transforming values from portrait mode to landscape mode. It swaps the axes around and properly modifies the rotation matrix `R`. The `remapCoordinateSystem()` method also requires four parameters (line 9): the first parameter represents the input rotation matrix `R` (previously gained through `getRotationMatrix()`), followed by two integer values indicating how the axes should be swapped and finally, the output array `outR` which will be filled with values when the function returns back and represents the modified rotation matrix. In line 10 of Listing 5.3, we define the method `getOrientation()` which uses the previously modified rotation matrix to calculate the phone orientation. The `getOrientation()` method returns an array containing the

device's rotation relative to the earth's magnetic north (azimuth) value, and also the pitch and roll values which are relative to the ground [56]. We only use azimuth and pitch values within our approach (cf. Section 5.2.4).

As the orientation method returns radiant values, we transform them into degrees (Listing 5.3 lines 11 and 12). To get values in the range of 0° to 360° we add 360° if the calculated azimuth value is less than null (lines 13-14). Furthermore, we multiply the pitch value by -1 (line 16) to be able to reverse the sign of the pitch value (plus to minus and minus to plus). Thus, visualized objects on the screen move against the up and down movement of the device and the virtual objects stay aligned with the respective real-world objects. Finally, we pass the recorded values to other classes of the project by calling the `updateIncomingSensorValues()` methods.

Problems

Unfortunately, measurements returned by the accelerometer and magnetic field sensors are coupled with noise. The earths magnetic field is a very weak signal. Thus, compasses are vulnerable to distortions [96]. Distortions can be caused by magnetic substances (e.g. cars, metal constructions of buildings) that are located near the compass and which disturb the signal of the magnetic field. As a result, we implemented a filter for reducing noise. In lines 6 and 7 of Listing 5.3, the raw accelerometer and magnetic field values undergo a low-pass filtering (cf. Listing 5.4, as seen in [97]), to smooth out noisy sensor readings. Otherwise, the values are unsteady and lead to inaccurate visualizations of resources on the screen.

```
1 protected float[] lowPassFilter( float[] raw, float[] filtered ) {
2   if ( filtered == null ) {
3     return raw;}
4   for ( int i=0; i<raw.length; i++ ) {
5     filtered[i] = filtered[i] + alpha * (raw[i] - filtered[i]);}
6   return output;
7 }
```

Listing 5.4: A code snippet of `ARMainActivity` – `lowPassFilter()`

The `lowPassFilter()` method in Listing 5.4, contains two arguments presenting float arrays of length 3. The input array contains the raw sensor data and the output array contains the filtered data. This method forces that each new raw value is normalized against the smoothed value. Thereby, the new sensor value is weighted with a factor alpha ($0 \leq \alpha \leq 1$ - smaller values cause more smoothing [97]) and added to the absolute sensor value. In Chapter 6, we provide diagrams which illustrate the effects of this low pass filter.

5.2.2 Loading and Processing LOD Resources

In this section, we present the realization of the loading and processing of LOD resources within our AR system. We start with the presentation of the classes and methods that we use to

handle LOD resources in our system. Then, we explain the processing steps of the data in the following sections.

In our AR system, we use two classes for loading data from LOD sets which we present in Figure 5.11.

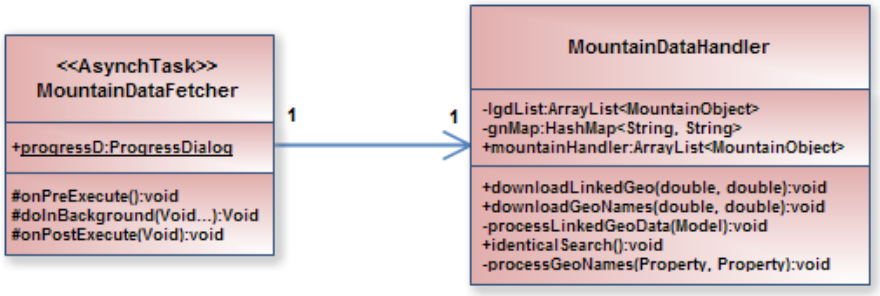


FIGURE 5.11: Classes responsible for data download

MountainDataFetcher – Attributes/Methods

The `MountainDataFetcher` class of Figure 5.11 extends the abstract class `AsyncTask` and is responsible for displaying a progress dialog that indicates that we fetch data from LOD sets in the background. Thus, the `MountainDataFetcher` contains an attribute called `progressD` of type `ProgressDialog`. We present the three methods of this class within the sequence diagram in Figure 5.12 and we describe them in Table 5.2.

MountainDataHandler – Attributes/Methods

The `MountainDataHandler` of Figure 5.11 comprises 3 attributes `lgdList`, `gnMap`, and `mountainHandler`. `lgdList` is an array list that we use to save LinkedGeoData resources as objects of type `MountainObject`. We do not transform GeoNames resources into Java objects until we compare them with the resources of LinkedGeoData to identify identical resources. Thus, we generate a hash map called `gnMap` to only store the URL of the resources as key and we add the label of the resource to be able to compare the labels between the resources of the two different LOD sets. Using a hash map, allows us to easily remove a GeoNames resource form the map that is equal to a certain LinkedGeoData resource. Then, the system iterates over the reduced `gnMap` and transforms the remaining GeoNames resources into Java Objects. Finally, we use the array list `mountainHandler` to merge the generated LinkedGeoData and GeoNames objects. This list contains all the mountains for a certain range and location that the system has to visualize on the screen. In Figure 5.11, there are also five methods visible which we explain within Table 5.2.

We present the interactions between the `MountainDataFetcher` and the `MountainDataHandler` classes of our AR system in the sequence diagram in Figure 5.12. The descriptions to these interactions follow in Table 5.2.

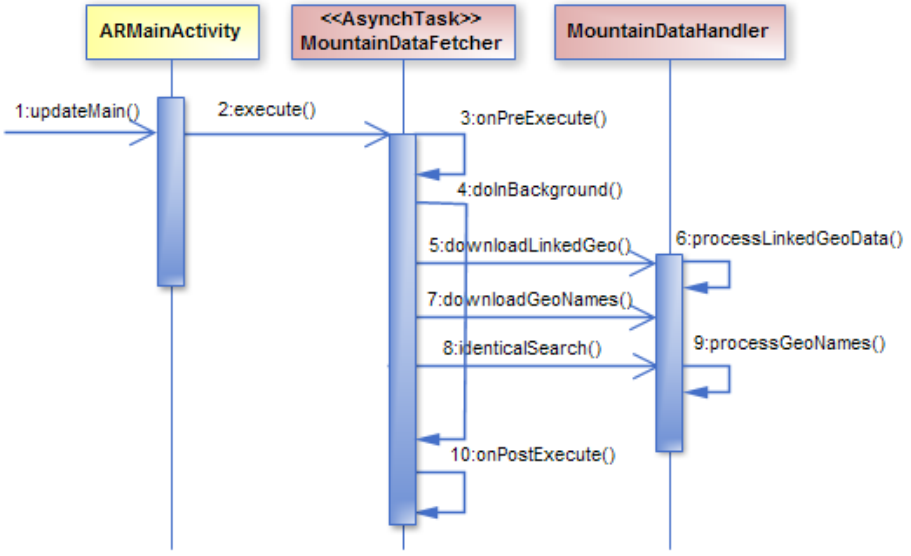


FIGURE 5.12: LOD download

ID	DESCRIPTION
1,2	Within `updateMain()` we start the LOD download by calling the method `execute()`.
3	In this method, we initialize a progress dialog that informs the user that currently data is loaded.
4	This method is responsible for the data download. It triggers the methods `downloadLinkedGeo()` and `downloadGeoNames()` of the `MountainDataHandler`.
5,6	These methods contain functions for loading data from LinkedGeoData and for processing the loaded resources to generate objects of type `MountainObject`. (cf. Section 5.2.2.1).
7	This method contains functions for loading data from GeoNames (cf. Section 5.2.2.2).
8	The resources of both LOD sets are compared and descriptions are merged. (cf. Section 5.2.2.3).
9	The system processes the remaining GeoNames resources and generates objects of type `MountainObject`.
10	After the background work finishes, the system invokes this method. Within this method, we define a toast notification (using the `Toast`[3] class of Android) that pops up on the surface of the window after the data is loaded, informing the user about the number of loaded resources.

TABLE 5.2: Description of interactions from Figure 5.12

5.2.2.1 LinkedGeoData - Integration

LinkedGeoData provides two possibilities to access LOD resources [98]:

REST API: allows access to all nodes and ways stored within a database which is set up by loading an OSM planet file[4]. This file contains all the map data (nodes, ways, and relations)

[3] `Toast`: helps to create and display a view containing a quick little message for the user, cf. `http://developer.android.com/reference/android/widget/Toast.html`

[4] OSM planet file: `http://wiki.openstreetmap.org/wiki/Planet.osm`

of OpenStreetMap.

SPARQL endpoints: There exists a static and live endpoint, whereby the former enables queries on data sets extracted from an OSM planet file and the latter queries on a copy of that OSM planet file, which is additionally synchronized with updates from OSM.

We make use of a SPARQL endpoint for downloading LOD resources from LinkedGeoData because the REST API provides only limited query capabilities. After testing the API, we also noticed that the adding of randomly chosen coordinates to the API URL led most of the time to internal server errors. The SPARQL endpoint[5] that we use in our work is offered by OpenLink's Virtuoso[6] database system. The LinkedGeoData ontology comprises a category defining peaks (`lgdo:Peak`) of which we make use to retrieve resources of type mountain. In Listing 5.5, we illustrate the SPARQL query that we use to fetch data from LinkedGeoData. The query includes a pattern to fetch resources of type `lgdo:Peak` (line 11). We also define patterns to retrieve the label (line 12) and the geo-coordinates (line 13-15) of this resource. Not for every resource there is a property `lgdo:ele` available at the LinkedGeoData data set. Thus, we define an optional graph pattern (line 16) which is only bound when a `lgdo:ele` property exists.

```
1  Prefix lgdo: <http://linkedgeodata.org/ontology/>
2  Prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  Prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
4  Construct {
5      ?resource a lgdo:Peak ;
6                rdfs:label ?label ;
7                geo:lat ?lat ;
8                geo:long ?long ;
9                lgdo:ele ?ele . }
10 Where {
11  ?resource a lgdo:Peak .
12  ?resource rdfs:label ?label .
13  ?resource geo:lat ?lat .
14  ?resource geo:long ?long .
15  ?resource geo:geometry ?point .
16  Optional{ ?resource lgdo:ele ?ele . }
17  Filter(bif:st_intersects (?point, bif:st_point (12.683,47.217), 10)).}
```

Listing 5.5: SPARQL query against LinkedGeoData

Finally, the query includes a filter method (line 17) that allows to fetch those resources which are located in the near vicinity of the current device location. Thereby, we use the `bif:st_intersects` function which is a geometry function provided by the OpenLink Virtuoso database system. This function makes use of three arguments: the first two represent longitude and latitude of the geo-location for which we search intersects, the third argument represents the allowed maximum distance in kilometers between them [99]. By using the query

[5]LinkedGeoData SPARQL endpoint: `http://linkedgeodata.org/sparql`
[6]OpenLink - Virtuoso: `http://virtuoso.openlinksw.com`

form `Construct` (line 4), it is able to return an RDF graph that is based on the template defined in lines 5 to 9.

Listing 5.6 exemplifies a resulting RDF graph if we execute the SPARQL query of Listing 5.5 against the SPARQL endpoint of LinkedGeoData. It contains the resource description of the peak "Bambachkopf" in the RDF/XML serializing format.

```
1 <rdf:RDF>
2  <rdf:Description rdf:about="http://linkedgeodata.org/triplify/node494054611">
3     <rdf:type rdf:resource="http://linkedgeodata.org/ontology/Peak"/>
4     <rdfs:label>Bambachkopf</rdfs:label>
5     <geo:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#double">47.2301061</geo:lat>
6     <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">12.6812105</
          geo:long>
7     <lgdo:ele rdf:datatype="http://www.w3.org/2001/XMLSchema#float">2516.0e0</lgdo:ele>
8  </rdf:Description>
9 </rdf:RDF>
```

LISTING 5.6: An example of a SPARQL query result received from the LinkedGeoData SPARQL endpoint

We use Apache's `HttpClient`[7] to execute the SPARQL query and to receive the resources from the LinkedGeoData platform (in RDF/XML format) as input stream. Furthermore, we use Androjena[8] to manipulate the RDF/XML-based data. Within Listing 5.7, we present the method of processing the RDF graph obtained from LinkedGeoData.

```
1  Model lgdModel = ModelFactory.createDefaultModel();
2  lgdModel.read(inputstream, "RDF/XML");
3  ResIterator i = lgdModel.listSubjects();
4  while (i.hasNext()) {
5    Resource resource = i.next();
6    String id = resource.getURI();
7    String lgdlabel = resource.getProperty(RDFS.label).getLiteral().toString();
8    Double lgdlat = resource.getProperty(lgdproplat).getDouble();
9    Double lgdlong = resource.getProperty(lgdproplong).getDouble();
10   Location lgdlocation = new Location("linkedgeodata");
11   lgdlocation.setLatitude(lgdlat);
12   lgdlocation.setLongitude(lgdlong);
13   if(resource.hasProperty(lgdpropelev)){
14     Double lgdalt = resource.getProperty(lgdpropelev).getDouble();
15     lgdlocation.setAltitude(lgdalt);
16   }else{
17     lgdlocation.setAltitude(Double.parseDouble(Utils.fetchAltitude(lgdlat.toString(),
           lgdlong.toString())));
18   }
19   MountainObject lgdmo = new MountainObject(id, lgdlocation, lgdlabel);
20   lgdList.add(lgdmo);}
```

LISTING 5.7: A code snippet of `MountainDataHandler` – `downloadLinkedGeo()`

[7] HttpClient: `http://hc.apache.org/httpcomponents-client-ga/`

[8] Androjena: Java API for Android which provides object classes for manipulating RDF graphs, resources, properties, and literals. cf. `http://code.google.com/p/androjena/`

In line 1 of Listing 5.7, we create an empty model called `lgdModel`. Then, we parse the statements of the loaded RDF graph into `lgdModel` (line 2). To be able to access specific data components, we create an iterator over `lgdModel` (line 3) by listing all the resources of the model. Then, we iterate the model to access specific properties. We fetch the id, the label, and the geo-coordinates of each resource and we use these parameters to generate a new object of type `MountainObject` in line 19. Not for every resources there is an elevation value available at the LinkedGeoData platform. Thus, we have to fetch this information with an extra method implemented in our `Utils` class (line 17).

Finally, we add each new object to the array list `lgdlist` (line 20). This list will later be used within the process of filtering identical resources (cf. Section 5.2.2.3).

5.2.2.2 GeoNames - Integration

GeoNames offers several Web services for accessing geo-spatial data contents, but no SPARQL endpoint. Most of the Web services represent REST APIs and are in need of the parameter "username". Thus, a developer has to register a GeoNames account to be able to test and implement one of the services. In our approach, we use the Web service called "Find Nearby Toponym". This service retrieves any resource of the GeoNames data set for a given geographical coordinate within a given radius (in km). The URL to this API[9] allows the adding of parameters to restrict the search process. Thus, it is possible to add a feature code parameter (cf. Section 4.3.1) to be able to retrieve resources of type mountain. This API returns results as XML, JSON or RDF/XML document [35].

In Listing 5.8, we present the REST API URL we use to access resources. We define several parameters: the current location of the device, the feature class and feature codes the resources have to exhibit, a fixed radius of 5 kilometer around the defined location in which resources have to be filtered, the type of format the resulting data have to be in, the maximal number of allowed result rows, and finally, for the request to work, our username.

```
1  String geonamesUrl = "http://api.geonames.org/findNearby?lat="+new Double(latitude).
       toString() +"&lng="+new Double(longitude).toString()+ "&featureClass=T&featureCode=
       MT&featureCode=PK&radius="+5+"&type=rdf&maxRows=50&username=botany";
2  gnModel.read(geonamesUrl, "RDF/XML");
3  ResIterator i = gnModel.listSubjectsWithProperty(propName);
4  while (i.hasNext()) {
5     Resource resource = i.next();
6     String id = resource.getURI();
7     String gnlabel = resource.getProperty(propName).getLiteral().toString();
8     gnMap.put(id, gnlabel); }
```

Listing 5.8: A code snippet of `MountainDataHandler` – `downloadGeoNames()`

We parse the result into an Androjena model and iterate the model to access the id and the label of the fetched resources and to put these two parameters into a hash map called `gnMap`.

[9] GeoNames - FindNearby API: `http://api.geonames.org/findNearby?`

We only filter the id and the label to prevent from unnecessary data processing. We generate objects of type `MountainObject` not until we performed the filtering of identical resources. Thus, we use `gnMap` during the identical search process which we explain in the next section.

5.2.2.3 Filtering Identical Resources

In Chapter 4, we described the Silk framework which is capable of discovering identical resources when comparing different LOD sources. We were not able to implement the offered Silk jar file within our Android project. Each Android application runs as an instance of the Dalvik VM (cf. Section 2.4.1). The Dalvik VM converts the Java class files into a Dalvik executable (dex) file [56]. Now, such a dex file has a limit of about 65,000 method references and as the Silk jar file includes a huge amount of methods the Dalvik VM was not able to convert all Java classes. Thus, we realized our own method (cf. Listing 5.9) for identifying identical mountain resources between the GeoNames and LinkedGeoData data sets instead of using the Silk framework.

In Listing 5.9, we iterate the resources (saved as `MountainObject`) of the array list `lgdList`. This is necessary so that we are able to compare every LinkedGeoData resource with the GeoNames resources. As explained in the previous section, the label and id of all GeoNames resources are saved within a hash map. Now, we iterate over this hash map and access the label of each GeoNames resource.

To be able to compare the labels of resources from the two different data sets, we use the Jaro-Winkler Distance metric. The LingPipe[10] Java API offers a `JaroWinklerDistance` class that includes a method for determining the Jaro-Winkler string comparison to obtain a distance measure. We use this distance measure to decide if two labels correspond to the same resource. `JaroWinklerDistance` returns a distance value in the range from 0 to 1, where 0 equals an exact string match and a value of 1 means that the two labels do not contain any matching characters [101].

In our implementation, we put all resources which exhibit a perfect score of 0 into the hash map `equals` (line 31). As some of the resources which result into a distance measure close to 0 may also be equals, we have to consider scores higher than 0. For instance, if we have a mountain named “Grieskogel” at LinkedGeoData and the same mountain named “Griesskogl” at GeoNames, then we obtain a Jaro-Winkler distance measure of 0.06666666666666676. In this case, we use a geometrical distance metric to decide whether these two labels describe the same real-world mountain or not. We defined that a Jaro-Winkler distance measure up to 0.11 causes the execution of the additional geometrical distance measurement because with a score of about 0.11 the difference between to labels is very minimal and the probability that these two labels describe the same mountain is very high. Thus, all the resources which result into a distance measure between 0 and 0.11 undergo the geometrical distance calculation in line 14 of Listing 5.9. If the geographical distance between the two resource locations lies beneath 200

[10]LingPipe is a tool kit for processing text using computational linguistics. cf. [100]

meters (line 15) the two resources are equal. Thus, the system puts the GeoNames resource that was currently compared with a LinkedGeoData resource into the `equals` hash map (line 26).

```
1  HashMap<String, String> equals = new HashMap<String, String>();
2  for(MountainObject lgdresource: lgdList){
3     for(String gnresource : gnmap.keySet()){
4       JaroWinklerDistance rwd = new JaroWinklerDistance();
5       String gnlabel =  gnmap.get(gnresource);
6       double d=rwd.distance((CharSequence)gnlabel,(CharSequence)lgdresource.getTitle());
7       if(d < 0.11){
8         if(d>0){
9           Double gnlat = gnModel.getResource(gnresource).getProperty(gnproplat).
                getDouble();
10          Double gnlong = gnModel.getResource(gnresource).getProperty(gnproplong).
                getDouble();
11          Location gnlocation = new Location("geonames");
12          gnlocation.setLatitude(gnlat);
13          gnlocation.setLongitude(gnlong);
14          double dist = lgdresource.getMountainLocation().distanceTo(gnlocation);
15          if(dist < 200){
16            if(!equals.containsKey(gnresource)){
17              ARMainActivity.db.insertDuplicate( lgdresource.getURL(),  gnlabel,  gnlat,
                      gnlong,  gnalt);
18              Double lgdlat = lgdresource.getLatitude();
19              if (lgdresource.getDistanceToUser() < (ARView.range * 1000)) {
20                  if(lgdlat.toString().length()>=gnlat.toString().length()){
21                    mountainsHandler.add(lgdresource);
22                  }else{
23                    MountainObject mo=new MountainObject(lgdresource.getURL(),gnlocation
                          ,lgdresource.getTitle());
24                    mountainsHandler.add(mo);
25                  }}
26                  equals.put(gnresource.toString(), gnlabel);
27                  break;}}
28        }else if(d==0){
29          if(!equals.containsKey(gnresource)){
30              ...
31              equals.put(gnresource.toString(), gnlabel);}
32              break;}}
33      ...
34 }
35 for (String id : equals.keySet()) {
36    gnmap.remove(id); }
```

LISTING 5.9: A code snippet of `MountainDataHandler` – Filtering identical resources using Jaro-Winkler Distance

During the identification of identical resources, we merge identical resource descriptions. Thus, we insert the new resource descriptions to the local database with the method `insertDuplicate()` (line 17) (cf. Listing 5.12). Thereby, we use the URL of the LinkedGeoData resource to be able to insert the new additional values to the respective resource in the database. Additionally, we add the resources to the array list `mountainHandler` in order to visualize the resources which are in a certain range (lines 19-25) on the display. Before we are able to do this,

we have to decide which geo-coordinates should be used for the further calculation steps within the mathematical model. Thus, we compare the length of the digits of the latitude information of the geographical coordinates. The latitude information which exhibits more decimal places is the one used for further visualization processing steps in our system, as they are more accurate than a latitude information with less decimal places [102].

After we are aware of all the identical resources, we iterate over the hash map `equals` by fetching the URL of each resource and comparing it with the URLs of the resources saved within the `gnMap` (lines 35-36). This way we are able to remove the resources from `gnMap` that have a matching URL. In the end, we transform the remaining GeoNames resources like the LinkedGeoData resources in Listing 5.7 into Java objects within the `processGeoNames()` method, and we add the generated `MountainObject` objects to the `mountainHandler` array list. Thus, LinkedGeoData and GeoNames resources are merged and ready to be visualized within the AR interface.

5.2.3 Storing Data

Parallel to the processing of resources within the `MountainDataHandler`, we also store the parameters of the `MountainObject` objects into a database on the local storage system of the device. Thereby, we make use of the SQLite library provided by Android. The SQLite database allows us to execute SQL commands for creating a table and for inserting new resources into this table [57]. In order for this to work, the `Utils` class of our Android project extends the abstract class `SQLiteOpenHelper` of the SQLite library. Thus, `Utils` acts as helper class to simplify our database interactions. In Listing 5.10, we present the method for creating a table within our database.

```
1     private static final String TABLE_NAME = "mountains";
2     private static final String KEY_ID = "id";
3     private static final String KEY_URL = "url";
4     private static final String KEY_NAME = "label";
5     private static final String KEY_LAT = "latitude";
6     private static final String KEY_LONG = "longitude";
7     private static final String KEY_ALT = "altitude";
8     private static final String KEY_ALT_NAME = "altname";
9     private static final String KEY_ALT_LAT = "altlatitude";
10    private static final String KEY_ALT_LONG = "altlongitude";
11    private static final String KEY_ALT_ELE = "altelevation";
12
13    public void onCreate(SQLiteDatabase db) {
14      String CREATE_TRIPLESTORE_TABLE = "CREATE TABLE " + TABLE_NAME + "("+ KEY_ID + "
            INTEGER PRIMARY KEY," + KEY_URL + " TEXT unique," + KEY_NAME + " TEXT," +
            KEY_LAT + " TEXT," + KEY_LONG + " TEXT," + KEY_ELE + " TEXT," + KEY_ALT_NAME +
             " TEXT," + KEY_ALT_LAT + " TEXT," + KEY_ALT_LONG + " TEXT," + KEY_ALT_ELE + "
             TEXT"+ ")";
15      db.execSQL(CREATE_TABLE);
16    }
```

Listing 5.10: Creating table in SQLite database

First, we define several constants that present the various fields of the table in lines 2-7. We additionally, define constants for an alternative name and geographical information in the case the system identifies duplicate resources during the LOD resource processing (cf. Section 5.2.2.3).

Within the `onCreate()` method of Listing 5.10, we define the string variable called `CREATE_TABLE` that contains the SQL statement for creating a table within our database. `onCreate()` is called by the system, if the database does not exists. We use an identifier as primary key of the table and the URL is set to be unique to allow unique resource entries. Thus, if a new data download is executed in the future and some of the resources are already stored within the database the URL allows to identify identical resources and to prevent the renewed storage of these resources. The `db` object of type `SQLiteDatabase` presents our physical database on the local storage system of the device and the `execSQL()` method (line 15) allows us to execute the SQL statement defined within `CREATE_TABLE`.

In Figure 5.13, we illustrate the sequence of interactions if the system is activated and new data is stored. The description of these interactions follows in Table 5.3.

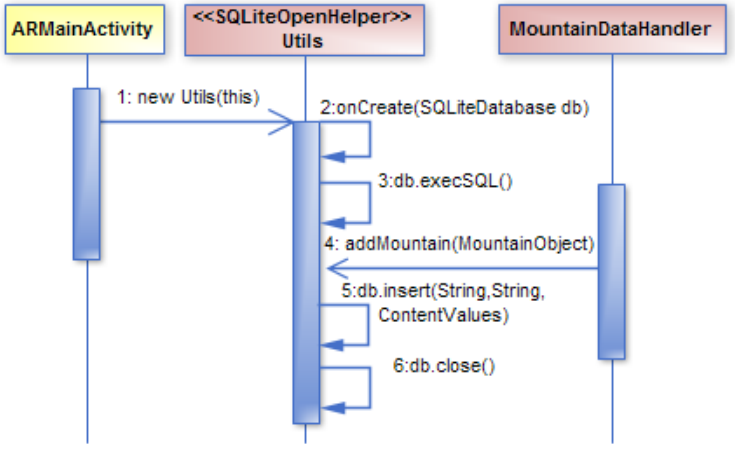


FIGURE 5.13: Storing LOD resources

ID	DESCRIPTION
1	We create a new instance of the `Utils` class to be able to access our database.
2	Is called once, if the database does not exist.
3	We use this method to create a table for the mountain resources within our database.
4	We use this method to add new values to the table of our database (cf. Listing 5.11).
5	This method allows to insert a new row to the table of our database.
6	The system closes the database connection.

TABLE 5.3: Description of interactions from Figure 5.13

The `addMountain()` method of Listing 5.11 accepts an object of type `MountainObject` as parameter. In line 2, we get writable access to our database. The `ContentValues` class

allows us to define the values that have to be inserted into the table. For instance, we primarily define the table column identifier and then we define the new content for the table in this column in line 4.

```
1   public void addMountain(MountainObject mo) {
2     SQLiteDatabase db = this.getWritableDatabase();
3     ContentValues values = new ContentValues();
4     values.put(KEY_NAME, mo.getTitle());
5     values.put(KEY_URL, mo.getURL());
6     values.put(KEY_LAT, mo.getMountainLocation().getLatitude());
7     values.put(KEY_LONG, mo.getMountainLocation().getLongitude());
8     values.put(KEY_ALT, mo.getMountainLocation().getAltitude());
9     db.insert(TABLE_NAME, null, values);
10  }
```

LISTING 5.11: A code snippet of `Utils` – Adding row to database table

5.2.3.1 Update Database Table

During the process of identifying duplicate resources (cf. Listing 5.9) we update specific table rows of the SQLite database. The system merges GeoNames resource descriptions with the LinkedGeoData resource description if the two resources are identical. In this case, we have to update the already stored LinkedGeoData resource descriptions within the database. In Listing 5.12, we present the method `insertDuplicate()` that manages the updating of specific table rows. The parameters of this method are the ID of the LinkedGeoData resource that will be updated and the name and the geo-coordinates of the GeoNames resource that is identical to the resource that exhibits the LinkedGeoData ID in the database. In the previous section, we explained that the database table includes cells for alternative resource descriptions (alternative name, latitude, longitude, and altitude). Now, we fill these cells of the table with the incoming values of the method. The function `update()` enables the updating process. Thereby, we have to define the name of the table, the new content values (GeoNames resources descriptions) that have to be inserted, and the LinkedGeoData ID in order that the system finds the respective row in the table that has to be modified.

```
1 public void insertDuplicate(String lgdid,String name,Double lat,Double lng,Double alt){
2     SQLiteDatabase db = this.getWritableDatabase();
3     ContentValues values = new ContentValues();
4     values.put(KEY_ALT_NAME, name);
5     values.put(KEY_ALT_LAT, lat.toString());
6     values.put(KEY_ALT_LONG, lng.toString());
7     values.put(KEY_ALT_ELE, alt.toString());
8     db.update(TABLE_NAME, values, KEY_URL+" = ?", new String[] {lgdid});
9 }
```

LISTING 5.12: A code snippet of `Utils` – `insertDuplicate()`

5.2.4 Mathematical Model

We implemented the mathematical model for the AR prototype based on the constructed model presented in Chapter 4.4. Thereby, we separated the implementation of the mathematical model in two different Java classes: the system prepares mountain specific data in the `MountainObject` class and we defined the methods for the calculation of screen coordinates for the virtual content within the `ARView` class. We present the most relevant attributes and methods of `ARView` and `MountainObject` in Figure 5.14.

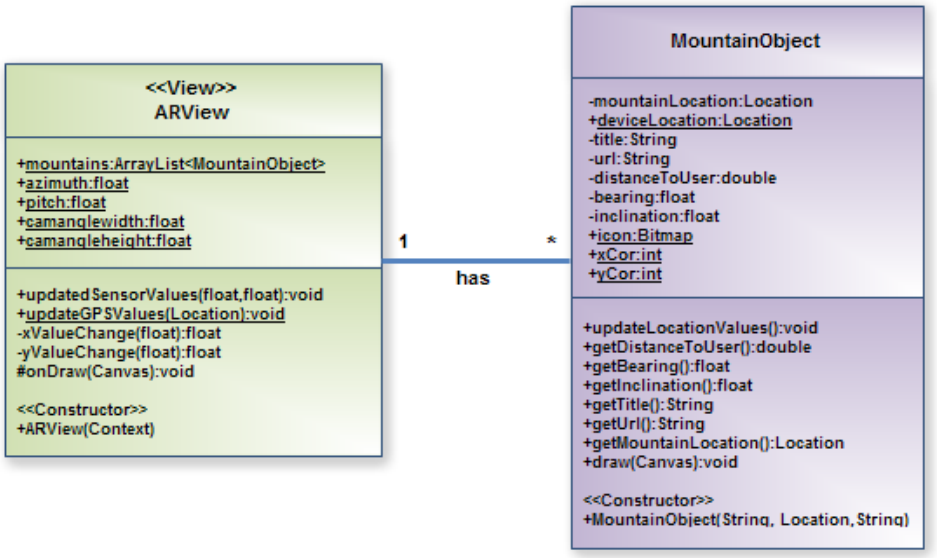


FIGURE 5.14: Classes responsible for screen coordinates calculation

ARView – Attributes/Methods

`ARView` contains an array list attribute called `mountains` that holds the downloaded mountain resources as an instance of `MountainObject`. The two attributes `azimuth` and `pitch` of type float hold the information about direction and orientation of the device. We need them to determine the current field of view (cf. Section 4.4.1.4). The two attributes `camanglewidth` and `camangleheight` are also of type float and they hold the values of the horizontal and vertical angle of the devices's digital camera (cf. Section 4.4.1.1).

The `ARView` class comprises the constructor method `ARView(Context)` and several other methods which we present in the sequence diagram of Figure 5.15 and we explain them in Table 5.4.

MountainObject – Attributes/Methods

The `MountainObject` class of Figure 5.14 comprises attributes for describing a LOD resource. We use the attributes of type `Location` for the calculation of the distance between

the mountains and the device location. The result of this calculation is saved within the attribute `distanceToUser`. A `MountainObject` instance also contains information about the title of the resource (`title`), the Web link to this resource (`url`), the bearing (`bearing`) value between current location and the location of the resources, and the inclination value (`inclination`). Another attribute presents an instance of the class `Bitmap` called `icon`. This is the icon that the system visualizes within the AR interface on the position (`xCor`,`yCor`).

The `MountainObject` class includes the get methods `getDistance()`, `getBearing()`, `getInclination()`, `getTitle()`, `getUrl()`, and `getMountainLocation()` which the system uses to access the variables of `MountainObject`. The constructor of the `MountainObject` class `MountainObject(String, Location, String)` comprises three parameters for the URL, the location, and the title of a loaded LOD resource. We describe the remaining methods of the class `MountainObject` (`updateLocationValues()` and `draw()`) in Table 5.4.

In Figure 5.15, we illustrate the sequence of interactions between the `ARView` and `MountainObject` classes and we describe them in Table 5.4.

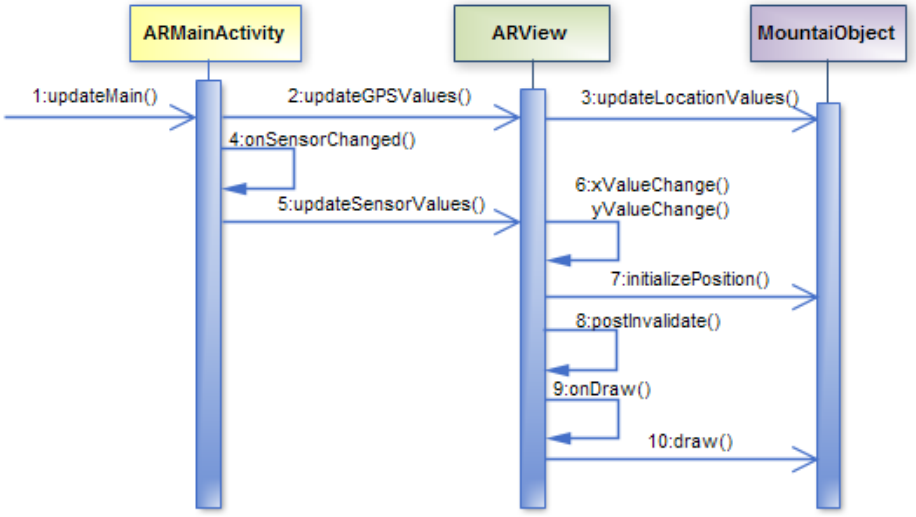


FIGURE 5.15: Determination of screen coordinates

The mathematical model comprises four important calculation steps (distance, bearing, inclination, and screen coordinates calculation) which we explain in the following sections.

5.2.4.1 Distance Calculation

Within the `MountainObject` class, we define the distance calculation between the current device location and the location of a resource. Android's `Location` package contains a method called `distanceTo()`. This method requires the target location as argument. In the case of this application, the target location is the location of a resource. `distanceTo()` implements Vincenty's inverse formula [93] for gaining the distance value. In Section 4.4, we presented and explained Vincenty's inverse formula.

ID	Description
1,2,3	The `updateMain()` method of `ARMainActivity` triggers the `updateGPS-Values()` method of `ARView` to update the location object of `ARView`. `updateLocationValues()` is called to force the re-calculation of distance and bearing values of an object of type `MountainObject`.
4,5	`onSensorChanged()` monitors sensor values and is called as soon as these values change. The new values are transmitted to `ARView` through the method `update-SensorValues()`. (cf. Listing 5.3)
6	Methods for determining x and y screen coordinates in pixels. (cf. Listing 5.14 and Listing 5.15)
7	After the screen coordinates determination, the positions of the mountain icons are re-initialized by calling this method.
8	Used to force the view to draw and re-draw.
9,10	`onDraw()` forces the system to render virtual content. Thereby, the `draw()` method of each `MountainObject` object is called. `draw()` defines functions for drawing a bitmap on the right position of the screen.

Table 5.4: Description of Interactions from Figure 5.15

5.2.4.2 Bearing Calculation

We also define the bearing calculation inside the `MountainObject` class. Android provides a method called `bearingTo()` for the calculation of the initial bearing in degrees East of true North when traveling along the shortest path between the current location and the target location [60]. Android's `bearingTo()` method determines the bearing value also with the help of Vincenty's inverse formula which we explained in Section 4.4. The bearing value is later used within the `ARView` class during the x-coordinate calculation for visualizing mountain icons on the screen.

5.2.4.3 Inclination Calculation

As mentioned in Chapter 4, the calculation of the inclination angle between the current location and a resource is based on a basic trigonometry function (cf. Section 4.4.1.3). We realized the inclination angle calculation in our AR project (cf. Listing 5.13) in reference to Definition 4.6. In Listing 5.13 line 2, we determine the opposite value by subtracting the altitude value of the current device location from the altitude value of the mountain location. If the altitude of the mountain is lower than the altitude of the device location, we obtain the opposite value by subtracting the altitude value of the mountain location from the altitude value of the device location. In lines 3 and 6, we use Equation 4.21 to obtain the inclination angle. The inclination angle is later used in the class `ARView` for the calculation of the y screen coordinate.

```
1 if (mountainLocation.getAltitude() > deviceLocation.getAltitude()) {
2   opposite = mountainLocation.getAltitude()- deviceLocation.getAltitude();
3   inclination = (float)Math.toDegrees(Math.atan(opposite/(double)getDistanceToUser()));
4 } else {
5   opposite = deviceLocation.getAltitude()- mountainLocation.getAltitude();
6   inclination = (float)Math.toDegrees(Math.atan(opposite/(double)getDistanceToUser()))
        *-1; }
```

Listing 5.13: Inclination angle calculation

5.2.4.4 Screen Coordinates Calculation

If the user moves the device the virtual objects have to stay aligned with the real world objects. Thus, new screen coordinates have to be calculated every time sensor values change. The determination of the screen coordinates takes place in the `ARView` class.

Listing 5.14 contains the method for determining the x screen coordinate. This code is based on the algorithm presented in the approach part of this thesis (Section 4.4.2). The method requires the bearing value (cf. Section 5.2.4.2) as input argument. As mentioned in Section 4.4, we have to determine the current field of view of the device to be able to check if a mountain is located within the captured view. Thus, the horizontal view angel of the device's camera and the direction value are used to calculate the left and right borders of the view angle (cf. Definition 4.7) in lines 2 and 3 of Listing 5.14. The field of view is always up-to-date because the system re-initializes `azimuth` constantly through the `onSensorChanged()` method within `ARMainActivity`.

```
1  private static float xValueChange(float bearingToMountain) {
2    double leftside = (azimuth-camanglewidth/2)<0 ? (azimuth-camanglewidth/2) + 360 : (
         azimuth-camanglewidth/2) ;
3    double rightside = (azimuth+camanglewidth/2)>360 ? (azimuth+camanglewidth/2) - 360 :
         (azimuth+camanglewidth/2);
4    float xDegreeValue = -1000;
5    if(bearingToMountain >= leftside){
6       xdegreeValue = (float) (bearingToMountain - leftside);
7    }else{
8       if(bearingToMountain <= rightside){
9          xDegreeValue = (float) (360 - leftside + bearingToMountain);}
10   }
11   float newXValue = xDegreeValue/camanglewidth*Utils.getScreenWidth();
12   return newXValue;
13 }
```

Listing 5.14: Calculation of x screen coordinate

The next step within the `xValueChange()` method (lines 5-10) is the mapping of the resource onto the horizontal angle of view which we defined in Definition 4.8. Then, we obtain the position on the horizontal screen angle (x_{haov}) of the current field of view of the camera. To be able to draw an icon on the screen, we transform x_{haov} into a screen coordinate in line 11 (cf. Definition 4.9).

We present the calculation of the y screen coordinate in Listing 5.15.

```
1  private float yValueChange(float m_inclination) {
2   float upperside = (camangleheight / 2) + pitch;
3   float lowerside = pitch - (camangleheight / 2);
4   float yDegreeValue = -180;
5   if (m_inclination >= lowerside) {
6       if (m_inclination < upperside) {
7         yDegreeValue = upperside - m_inclination; }
8   }
9   float newYValue = yDegreeValue/camangleheight*Utils.getScreenHeight();
10  return newYValue;
11 }
```

Listing 5.15: Calculation of y screen coordinate

We explained the algorithm of `yValueChange()` in the approach part of this work in Section 4.4.2. This method requires the inclination angle (cf. Section 5.2.4.3) as input argument and we have to determine the lower and upper borders (lines 2-3) using the pitch value and the vertical angle of view of the camera based on Definition 4.7. Then, we map the resources on the vertical angle of view (lines 5-8) as defined in Definition 4.8 to obtain the position on the vertical screen angle (y_{haov}). Finally, we calculate the y screen coordinate (line 9) based on Equation 4.31.

5.2.5 Querying Mountain Information

To be able to obtain more information about a specific resource within our AR system, the user has to select visualized mountain icons on the screen to trigger the `MountainInfoFetcher` which fetches the respective data from DBpedia.

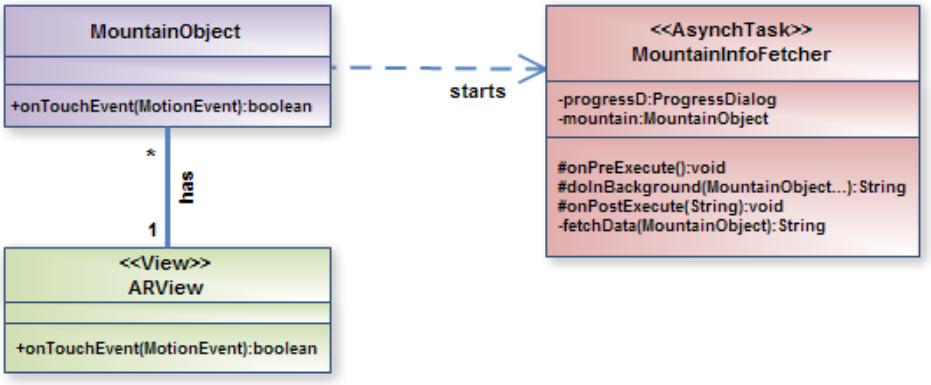


Figure 5.16: Classes responsible for loading mountain information

ARView and MountainObject – Attributes/Methods

In Figure 5.16, we present the `MountainInfoFetcher` in combination with the two classes `ARView` and `MountainObject` which we already described earlier in this chapter. As visible,

`ARView` is able to comprise none or several objects of type `MountainObject`. Both of these classes include the `onTouchEvent()` method. If a user touches the screen, the AR system delivers a touch event to `ARView`. These methods implement the tasks that the system has to perform in the case of a touch event. We further describe them in Table 5.5.

MountainInfoFetcher – Attributes/Methods

We implemented the loading of resource information as asynchronous task. Thus, `MountainInfoFetcher` extends `AsynchTask` and therefore comprises 3 methods for three different task states: `onPreExecute()`, `doInBackground()`, and `onPostExecute()`. We describe all these methods including `fetchData()` within Table 5.5.

`MountainInfoFetcher` also contains two attributes, whereby one of them is an instance of `MountainObject`. This attribute presents the resource for which additional data is requested. The other attribute `progressD` is of type `ProgressDialog`. We need it to display a progress dialog while the system loads additional information in the background.

In Figure 5.17, we present the sequence of interactions the system performs in the case the user touches a mountain icon within the AR interface. The description of these interactions follows in Table 5.5.

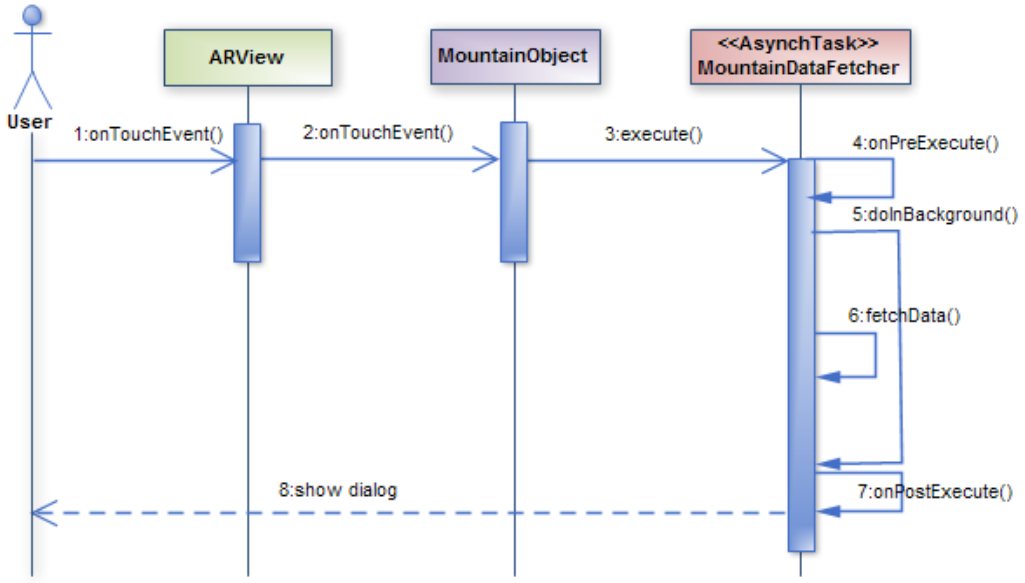


FIGURE 5.17: Mountain information loading

ID	Description
1,2	If we select a mountain icon a touch screen motion event occurs and the `onTouchEvent()` method is called on that object. Within the `onTouchEvent()` of `ARView`, we use the `dispatchTouchEvent(event)` method of the respective `MountainObject` instance, to pass the touch screen motion event down to the target view. This causes the activation of the `onTouchEvent()` of the `MountainObject` class.
3	We start the data download with the method `execute()`, whereby we load data for a specific instance of `MountainObject`.
4	Is called before the thread is started. In this method, we initialize a progress dialog that informs the user that the system currently loads data.
5	In this method we load data from DBpedia for a specific resource.
6	This method extracts the loaded data from DBpedia and generates a HTML view. (cf. Listing 5.16)
7,8	Invokes after the background computation finishes. We load data in a dialog which we display on the screen. (cf. Listing 5.17)

Table 5.5: Description of interactions from Figure 5.17

In Listing 5.16, we present the `fetchData()` method. With the help of the Androjena framework we create an empty model in line 1. Then, we initialize an instance of the `RDFReader` class which is able to read a RDF/XML-serialized file. In line 3, we define the model into which the statements should be loaded and the resource URI which contains the statements.

```
1  Model model = ModelFactory.createDefaultModel();
2  RDFReader r = model.getReader("RDF/XML");
3  r.read(model,"http://dbpedia.org/data/"+title+".rdf");
4  StringBuffer html = new StringBuffer();
5  html.append("<html><head></head><body><table cellpadding='10'>");
6  StmtIterator it = model.listStatements(null,null, (RDFNode) null);
7  while (it.hasNext()) {
8    Statement s = it.nextStatement();
9    if(s.getPredicate().toString().contains("comment") && s.getObject().toString().
         contains("@en")){
10     html.append("<tr><td bgcolor='#F0F0F0'><font size='4'><b>More Facts:</b></font></
           td></tr>");
11     String[] tempAbs = s.getObject().toString().split("\\@");
12     html.append("<tr><td>"+tempAbs[0]+"</td></tr>");}
13     ... }
14 html.append("</table></body></html>");
```

Listing 5.16: A code snippet of `MountainInfoFetcher` – `fetchData()`

We display mountain information in form of generated HTML views, thereby we use a `StringBuffer` called `html` to append the extracted RDF data of DBpedia (line 4) within specific HTML tags. In the end, this string buffer comprises a complete HTML page. To be able to retrieve specific objects of the model, we use a statement iterator called `it`. Thereby, we

iterate over every statement saved within the model and filter for instance in line 9 the property `comment`. In addition, we define that the literal value of the respective object has to be in English. If this statement exists, the object is transformed into a string value and added to the string buffer `html` (lines 11-12). We also retrieve other objects of the model but we do not present them within Listing 5.16.

We present a code snippet of the `onPostExecute()` method in Listing 5.17. It comprises the creation of a `WebView` object (line 1) which we use to load data from the string buffer `html` (line 2). Finally, we define a dialog to be able to add `webview` as content view and to display the loaded data to the user.

```
1  WebView webview = new WebView(context);
2  webview.loadData(html.toString(), "text/html", "utf-8");
3  Dialog d = new Dialog(context, R.style.myBackgroundStyle) {
4  public boolean onKeyDown(int keyCode, KeyEvent event) {
5      if (keyCode == KeyEvent.KEYCODE_BACK)
6         this.dismiss();
7         return true;}};
8   d.setTitle("Mountain > " + mountain.getTitle());
9   d.addContentView(webview, new FrameLayout.LayoutParams(600, 300));
10  d.show();
```

LISTING 5.17: A code snippet of `MountainInfoFetcher` – `onPostExecute()`

5.2.6 Loading Tours

Our AR prototype additionally includes tour features: the user has the possibility to load GPS tours from OpenStreetMap[11] or to load self-recorded GPS tours from the local storage system of the device on a 2D map. We present the responsible classes of our AR system for loading tracks in Figure 5.18.

StartActivity – Attributes/Methods

We already described the methods and most of the attributes of `StartActivity` in Section 5.2.1. If the user selects the option to load tours from the Web, the system displays a dialog on the screen with a seek bar on it. Thus, this class includes attributes of type `Dialog` and `SeekBar`[12]. On the seek bar the user has to choose a progress value (kilometers) to define the area in which GPS tours have to be searched at OpenStreetMap.

TourDataFetcher – Attributes/Methods

The `TourDataFetcher` class extends the abstract class `AsynchTask` and comprises three methods for the three different task states: `onPreExecute()`, `doInBackground()`, and

[11] OSM GPS tracks available at: `http://www.openstreetmap.org/traces`
[12] A bar representing how far a operation has progressed.

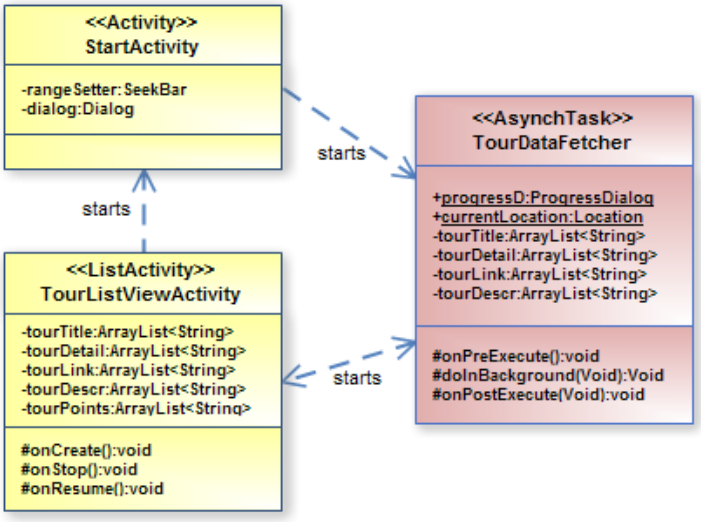


FIGURE 5.18: Classes responsible for loading GPS tracks

`onPostExecute()`. This class comprises an attribute of type `ProgressDialog`. This dialog appears in front of the screen as soon as the system loads tours from OpenStreetMap. There is also an attribute of type `Location` which holds information about the current location of the device. The remaining attributes present array lists. We use these array lists to collect the title of the tours (`tourTitle`), the distance to the starting point of the tours (`tourDetail`), the OpenStreetMap link to the tours (`tourLink`), and a description of theses tours (`tourDescr`). We use these lists to pass GPS track information to `TourListViewActivity` within an intent (further described later within this section).

TourListViewActivity – Attributes/Methods

The class `TourListViewActivity` comprises the same array list attributes as the class `TourDataFetcher`. Additionally, there is an array list (`tourPoints`) that holds all the tracking points (lat/long pair) of a loaded tour. If the user selects a tour of the list, the GPS coordinates of `tourPoints` are used to visualize the tour track on a map view. This class extends a `ListActivity`[13], thus we manage the activity life-cycle with the `onCreate()`, `onResume()`, and `onStop()` methods.

In Figure 5.19, we present the interaction sequence between the `StartActivity`, the `GPSService`, the `TourDataFetcher`, and the `TourListViewActivity` classes. We describe the depicted interactions within Table 5.6.

Within Listing 5.18 we present the `doInBackground()` method of the `TourDataFetcher` which implements the loading of tour data from OpenStreetMap. As OSM offers a huge amount on tracks we have to find a way to look for tracks which exhibit a certain property. The provided GPS tracks at OSM can be specified with tags to display specific tracks which exhibit the certain property the tag stands for (e.g. Austria, Salzburg, running etc.). OSM does not

[13] Activity that displays a list of items.

provide a Web Service to easily be able to fetch the GPS tracks. The tracks are listed on HTML Web pages. Thus, we have to extract the tours out of the HTML page.

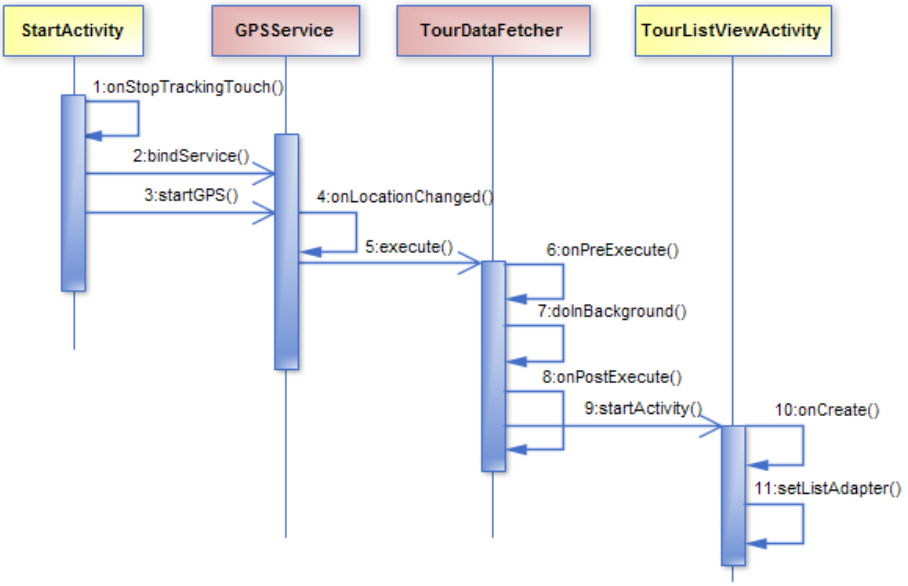


FIGURE 5.19: Tour track loading

ID	DESCRIPTION
1	A user has to select a kilometer value on a seek bar to define a restricted area within which GPS tours are located. An Android seek bar uses a listener which is a callback that notifies a client if the progress level of the bar changes [60]. In this case, the listener uses `onStopTrackingTouch()` to notify the AR system about the completion of the touch gesture from the user.
2, 3	`bindService()` is used to bind `StartActivity` to `GPSService` and `startGPS()` triggers the action of the GPS radio. Then, `StartActivity` receives location updates.
4, 5	The first location update triggers the execution of `TourDataFetcher`.
6	A progress dialog is created and displayed on top of the screen to inform the user that in the background tours are fetched.
7	`doInBackground()` implements methods for fetching GPS tours (cf. Listing 5.18).
8, 9	After the completion of `doInBackground()`, the system executes `onPostExecute()` which is responsible for starting the `TourListViewActivity`. The system uses an intent (cf. Section 2.4.3) to start this activity which is extended with the array lists which comprise the information of the loaded GPS tracks.
10, 11	Within the `onCreate()` method we set a list adapter to be able to display tour tracks within a list. Every list item contains the name of the tour, a short description, and the distance (information of the passed array lists). A graphical UI example of this result list is available in Chapter 6. If a user selects an item of the list the system loads the actual GPX file which contains the way points of the tour track. Then, the system activates `MountainMapActivity` which is responsible for displaying this track onto the map (cf. Section 5.2.7).

TABLE 5.6: Description of interactions from Figure 5.19

In order for this to work, we use the Java HTML parser jsoup[14]. In Listing 5.18, we present the method of extracting GPS tracks of an OSM HTML page. We iterate over the tags OSM offers to list specific GPS tracks. To access the Web pages, we create an instance of the `Connection` class by calling the method `connect()` and by passing a string (the URL of the OSM Web page) as argument. With the help of jsoup's `get()` method we are able to fetch and to parse the HTML file (line 4). We save the content of the downloaded HTML file within a `Document` object. Then, we are able to select and save individual HTML elements with jsoup's `Elements` object.

As all tracks on the GPS track Web sites of OSM are located within different HTML table elements (e.g. `table0`, `table1`), we select for instance every `table0` element and save it within an `Elements` object (line 5). Then, we extract specific data elements out of the selected elements within a for-loop. The OSM GPS track HTML pages hide the GPS starting points within HTML link tags. Thus, we select the elements that exhibit a tag `"a"` and save them within a new element called `links` (line 9). In consequence, we are able to iterate over all the links within the OSM HTML file and to extract data that describe a GPS tour track.

```
1  for (String stag : tags) {
2    String tag = URLEncoder.encode(stag, "UTF-8");
3    Connection connection = Jsoup.connect("http://www.openstreetmap.org/traces/tag/"+tag);
4    Document doc = connection.get();
5    Elements content = doc.select("td.table0");
6    ...
7    for (Element element : content) {
8      String html = element.html();
9      Elements links = element.getElementsByTag("a");
10     for (Element link : links) {
11       Elements geoinfo =link.getElementsByAttributeValueStarting("href","/?lat");
12       for (Element specific : geoinfo) {
13         String astring = specific.toString().replace("<a href='/?lat=", "");
14         astring = astring.replace("&amp;lon=", ";");
15         String[] locationValues = astring.split(";");
16         tourLocation.setLongitude(Double.parseDouble(locationValues[1]));
17         tourLocation.setLatitude(Double.parseDouble(locationValues[0]));
18         float distance=((devicelocation.distanceTo(tourLocation)/1000)*100)/100;
19         ...
20         tourTitle.add(title);
21         tourDetail.add("Distance to Tourstart: "+ distance + " km");
22         tourLink.add(urlfile);
23         tourDescr.add(descr[1]);
24   }}}}
```

Listing 5.18: A code snippet of `TourDataFetcher` – Filtering GPS tour details from HTML page

Whereby, we concentrate on the GPS tour-start coordinates but also on the title, the download link, and a short description of each GPS track. For instance, we extract the tour-start coordinates of a GPS track in Listing 5.18 and we save the extracted tour-start coordinates

[14] jsoup: `http://jsoup.org/`

within a `Location` object named `tourLocation` (lines 16-17). We use the tour-start information to calculate the distance to the tour start. Finally, we add the extracted information for every GPS track to specific array lists (lines 20-23).

5.2.7 Parsing and Plotting GPX Files on a Map

If the user wants to view a GPS recorded tour on a map, all the way points of a GPX file have to be extracted and plotted onto the map. We present the structure of a GPX file in Listing 5.19 where the tag `<trk>` stands for track. `<trk>` includes the name (`<name>`) of the GPS track and a tracking sequence (`<trkseg>`). The tracking sequence contains the tracking points (`<trkpt>`) which present latitude and longitude coordinates of a recorded GPS track. The `<trkpt>` directives enclose the elevation value (`<ele>`) of the current tracking point and the recording time-stamp (`<time>`).

As GPX files exhibit a XML-based structure, we chose to parse the files with the help of the Java XPath API[15]. This API allows the querying of XML-based files from Java programs.

```
1  <gpx xmlns="http://www.topografix.com/GPX/1/1" creator="" version="1.1" xmlns:xsi="http:
       //www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.topografix.com/
       GPX/1/1/gpx.xsd">
2   <trk>
3    <name>Example Track</name>
4    <trkseg>
5     <trkpt lat="47.296852" lon="12.772224">
6      <ele>784.552</ele>
7      <time>2011-10-27T08:00:19Z</time>
8     </trkpt>
9     <trkpt lat="47.296812" lon="12.772171">
10     <ele>779.938</ele>
11     <time>2011-10-27T08:00:29Z</time>
12    </trkpt>
13    ...
14   </trkseg>
15  </trk>
16 </gpx>
```

Listing 5.19: Example of GPX file structure

In Listing 5.20, we present the implementation of parsing GPX files in our system. We create a new `XPath` object by using the `XPathFactory`. Then, `xpath` compiles the XPath expression (`"//trkpt"`) which forces that every tag named `<trkpt>` is selected as soon as the XPath expression is executed. In line 3 we evaluate the compiled expression and we receive a node-set as result. The system casts this node-set to a `NodeList` which we use to iterate over each item of the list. Thereby, we select the latitude and longitude information of each node and retrieve their text content (`getTextContent()`) (lines 6-7). We save the results within a string array and then we add it to an array list (line 9). Later, we are able to iterate over the points within this array list in order to plot them onto a map.

[15]XPath API: `http://xml.apache.org/xalan-j/xpath_apis.html`

```
1  XPath xpath = XPathFactory.newInstance().newXPath();
2  XPathExpression expr = xpath.compile("//trkpt");
3  Object fileObject = expr.evaluate(gpxfile,XPathConstants.NODESET);
4  NodeList nodes = (NodeList)fileObject;
5  for(int j = 0;j<nodes.getLength()-1;j++){
6   String lat =nodes.item(j).getAttributes().getNamedItem("lat").getTextContent();
7   String lng =nodes.item(j).getAttributes().getNamedItem("lon").getTextContent();
8   String point[] = {lng, lat};
9   arrPathPoints.add(point);
10 }
```

LISTING 5.20: Parsing GPX file with XPath API

We present the responsible classes of our AR system for visualizing tracks on a map in Figure 5.20.

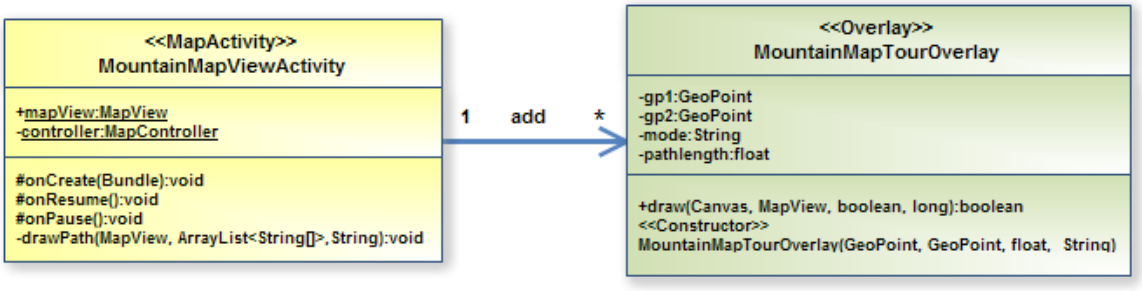


FIGURE 5.20: Classes responsible for plotting GPS tracks

MountainMapActivity – Attributes/Methods

The `MountainMapActivity` class extends `MapActivity` and includes an attribute that is of type `MapView`. `MapView` allows to display a map and it provides programmatic control of the map display (zoom, overlays, display modes e.g. satellite, street) [57]. Furthermore, we define an instance of the class `MapController` to be able to control the map (set center location, set zoom levels). The `onCreate()`, `onPause()`, and `onResume()` methods are necessary for managing the life-cycle of the activity. Moreover, this class comprises a `drawPath()` method that is responsible for generating several instances of `MountainMapTourOverlays`. Thereby, we use a loop to iterate through an array list of geographical points (all points of the GPS track) to generate these overlays.

MountainMapTourOverlay – Attributes/Methods

We use overlays to parse a GPS tour on a map (cf. Definition 5.2). Thus, `MountainMapTour-Overlay` extends the class `Overlay`. It comprises 4 attributes, whereby two of them present instances of the class `GeoPoint`. Between these two points we have to visualize a line. The attribute `pathlength` contains the path length of the GPS track that the system visualizes

on the map. We use this value to display the length of the GPS track in kilometers on top of the map as additional information for the user.

`MountainMapTourOverlay` also comprises a constructor with the geographical points of the GPS tour track, the path length of the track, and the mode value as parameters. We add several instances of the class `MountainMapTourOverlay` to the class `MountainMapActivity` because each `MountainMapTourOverlay` instance presents a line between two geographical points of a GPS track.

The `draw()` method of the class `MountainMapTourOverlay` actually draws a line between two geographical points. We present a part of this method in Listing 5.21. Within this method, we define a projection to convert the passed GPS coordinates (presenting the GPS points of the tour) into pixels (x/y screen coordinates) in lines 1-4. For plotting the path between these points, we use the `drawLine()` method (line 5). To be able to draw special start and end markers, we use a string variable (`mode`) which indicates if certain coordinates present the starting or ending GPS points of the tour (lines 6-11).

```
1  Point point = new Point();
2  projection.toPixels(gp1, point);
3  Point point2 = new Point();
4  projection.toPixels(gp2, point2);
5  canvas.drawLine((float) point.x, (float) point.y, (float) point2.x,(float) point2.y,
       paint);
6  if(mode =="start"){
7   Bitmap start = BitmapFactory.decodeResource(MountainMapViewActivity.context.
        getResources(),R.drawable.marker_start);
8   canvas.drawBitmap(start,point.x-(start.getWidth()/2),point.y-start.getHeight(),null);}
9  if(mode =="end"){
10  Bitmap end = BitmapFactory.decodeResource(MountainMapViewActivity.context.getResources
        (),R.drawable.marker_end);
11  canvas.drawBitmap(end,point.x-(start.getWidth()/2),point.y-start.getHeight(),null);}
```

LISTING 5.21: A code snippet of `MountainMapTourOverlay` – Plotting GPS tour onto map

5.2.8 Recording Tour

Our AR prototype additionally provides the feature of recording GPS tracks and to write recorded tracks into a GPX file on the local storage system of the mobile device. We present the responsible classes of our AR system for recording GPS tracks in Figure 5.21.

TourRecordActivity – Attributes/Methods

The class `TourRecordActivity` comprises four attributes: we use the attribute `mapView` to display a map and `mapController` allows us to control this map. The array list `arrGeoPointList` contains all the GPS points of a recorded GPS tour as instances of the class `GeoPoint`. We use this list for drawing the currently recorded GPS tour on the map with the

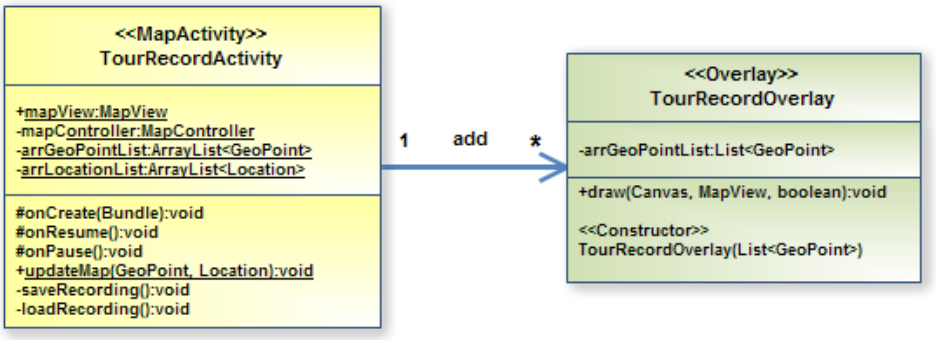


FIGURE 5.21: Classes responsible for recording GPS tracks

help of `TourRecordOverlay`. The other array list `arrLocationList` contains the recorded geographical points as type `Location` (lat/long pair). We use this list for writing each tracking point of the recorded GPS track into a GPX file.

As `TourRecordActiviy` extends the class `MapActivity`, we control the life-cycle of this activity with the three methods `onCreate()`, `onPause()`, and `onResume()`. The method `updateMap()` is called at every location update. We receive the new location point as type of `GeoPoint` and as type of `Location`. In consequence, we are able to add every new location point to `arrGeoPointList` and to `arrLocationList`. `updateMap()` also causes the map view to re-draw. Thus, a new line is visualized between the last recorded geographical point and the new one on the map.

The two remaining methods of `TourRecordActivity` are responsible for saving the recorded GPS track (`saveRecording()`) within a GPX file and for loading a previously recorded GPS track (`loadRecording()` of a GPX file from the local storage system of the device.

TourRecordOverlay – Attributes/Methods

The class `TourRecordOverlay` comprises one attribute which is of type `ArrayList<GeoPoint>`. This list contains the recorded GPS coordinates saved as instances of the `GeoPoint` class. We use this list within the `draw()` method for drawing a line between the last two points of the list. Thus, the user is able to view the already passed path on the map.

5.3 Summary

This chapter comprises the design and implementation of our proof-of-concept AR prototype. The prototype is optimized for running on Android mobile phones that include a GPS radio, digital camera, accelerometer, and magnetometer. Our AR system provides three different modes: AR, map, and tour mode. The AR mode comprises an AR interface and is based on the formal model presented in Chapter 4. The map mode displays resources on a 2D map and the

tour mode offers tour features such as the loading of GPS tours from OpenStreetMap or from the local storage system of the device, and the recording and saving of GPS tracks.

We implemented the receiving of GPS location updates with the help of an Android service which enables the access to the GPS radio of the device. Each mode of the prototype presents a separate Android activity. As soon as an activity of the system is in need of GPS location updates, it binds itself to the GPS service.

The AR interface of our system consists of several Android views that are stacked one above the other (cf. Figure 5.3). Thereby, the base view is the camera view which enables to capture live scenes of the surroundings, whereby the system visualizes virtual content on top of it.

We fetch LOD resources from LinkedGeoData and GeoNames in RDF/XML-based format and we manipulate the resources with the help of the Androjena framework. The two data sets may include duplicate resources. We filter them by accomplishing two similarity measures: the Jaro-Winkler distance and the geographical distance (determining geographical distance between two resources). The system merges resource descriptions of two resources that are identified as identical. During the processing of LOD resources they are stored in a SQLite database on the local storage system of the mobile device. After the data preparation, the system starts with the process of determining screen coordinates for a group of resources that are located within a certain range near the device (based on the formal model introduced in Section 4.4). In the end, the screen coordinates are used to visualize little triangle icons on the display of the mobile device.

We present the AR Mountain Guide through the illustration of several screen shots of the graphical interface of the prototype in the next chapter, where we also evaluate and discuss the data processing and visualization performance of our AR system.

Chapter 6

Results and Discussion

This chapter consists of three main parts: within the first part we present our proof-of-concept prototype by illustrating screen-shots of the AR application. The second part of this chapter comprises the evaluation of our prototype. Thereby, we firstly concentrate on the performance of processing LOD resources in our AR system. This means we measure how long the application needs to parse, filter, and further process LOD resources. Secondly, we evaluate the visualization performance[1] of AR content in our AR system. In the last part of this chapter, we discuss our findings.

6.1 Prototype Presentation

We divide the presentation of the prototype into two parts: the first part contains specifics about the AR mode of the AR application (cf. Section 6.1.1) and the second part is dedicated to the map and tour modes presentation (cf. Section 6.1.2).

6.1.1 AR Mode

We start with the illustration of the home screen of the AR application in Figure 6.1. The home screen appears if the user starts the application and it comprises 3 different buttons on the left side of the screen. The buttons represent the entry points to the various modes of the application.

If the user activates the AR mode of the application, the AR interface presented in Figure 6.2 appears on the screen. The application visualizes recognized mountains as blue triangles with grey boxes beneath them. The grey box contains the name of a mountain. For mountains, which are located further than 5 kilometers from the current location of the device, we use smaller triangles and a transparent blue color for the text boxes, as visible in Figure 6.2. The interface also comprises a compass rose to inform the user about the current heading, and a

[1] The time that the visualized objects need to stabilize in front of the respective real-world object.

refresh button to repeat the data download. We further explain these additional features in Section 6.1.1.1.

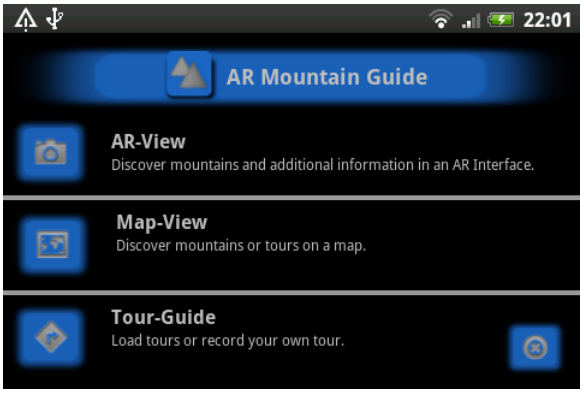


FIGURE 6.1: Home screen - entry point of application

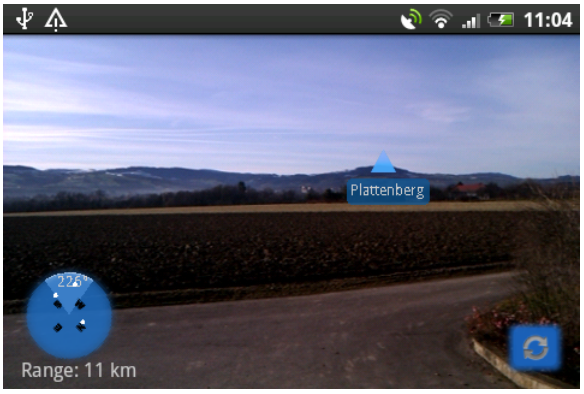
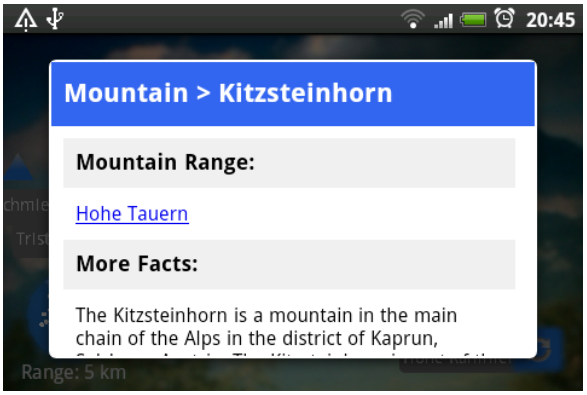
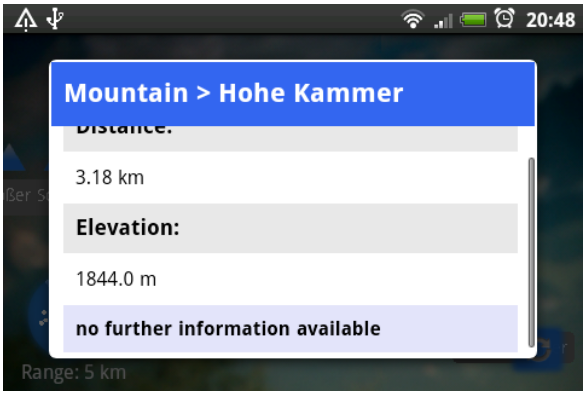


FIGURE 6.2: AR interface with visualized mountain object

If the user selects a visualized object on the screen, a dialog window opens. The dialog contains extracted RDF data fetched from the DBpedia data set (cf. Figure 6.3 (a)). As already explained in Section 5.2.5, a DBpedia resources description does not exist for every mountain resource fetched from GeoNames or LinkedGeoData. Thus, if no further descriptions are available, we simply display the distance value from the mountain to the user and the elevation value of the mountain (cf. Figure 6.3 (b)).



(a) DBpedia description is available



(b) DBpedia description is not available

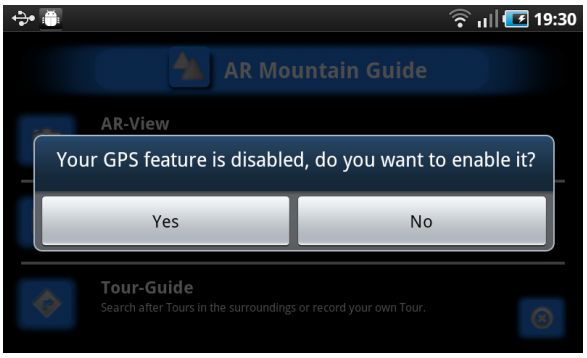
FIGURE 6.3: Displaying further mountain information

6.1.1.1 Additional Features

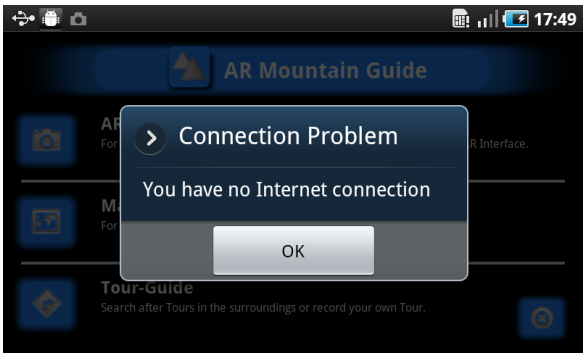
Besides the main functionalities, which we implemented to realize the research questions of Section 1.4, we also created additional features for the prototype. We present them in the following paragraphs:

System Messages

If the GPS radio on the device is not activated, the user will be informed with the help of a dialog visible in Figure 6.4 (a).



(a) GPS radio is inactive

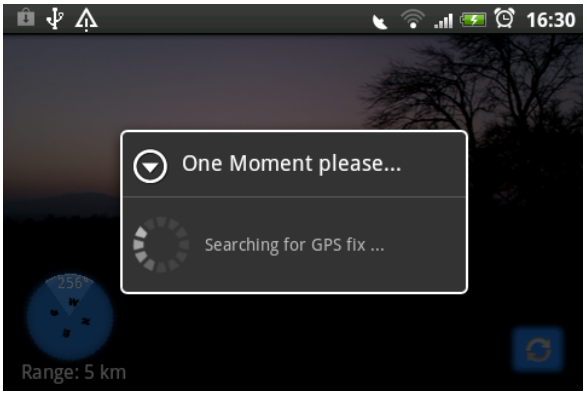


(b) No Internet connection is available

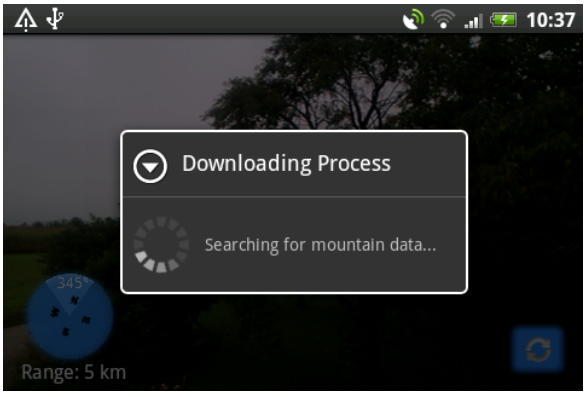
FIGURE 6.4: System messages

If the user taps on a button of the home screen to activate a mode, the application checks if the device has access to the Internet. If there is no connection available, the alert dialog in Figure 6.4 (b) appears on the screen. The different application modes can only be accessed if an Internet connection is available and the GPS radio of the device is activated.

During the time the GPS radio searches for a satellite fix, a progress dialog appears on the screen. At the first GPS fix, another progress dialog appears informing the user that the system currently fetches mountain data from LOD sets. Both of these messages are visible in Figure 6.5. The progress dialogs are necessary to let the users know that their action took effect and to inform them that they have to wait a brief moment for the process to be completed.



(a) GPS fix is searched



(b) Downloading process is active

FIGURE 6.5: Progress dialogs for bypassing waiting time

Range Setter

In Figure 6.6, we present the AR interface of the application with a seek bar on top of it. On this bar the user is able to modify the tracking range[2] in order to restrict the data that the system visualizes on the screen. The bar has a maximum value of 20 km. The higher the range is, the more data we receive from the LOD sets. Thus, we defined a limit of 20 km to avoid an overload of the screen with visualized LOD resources.

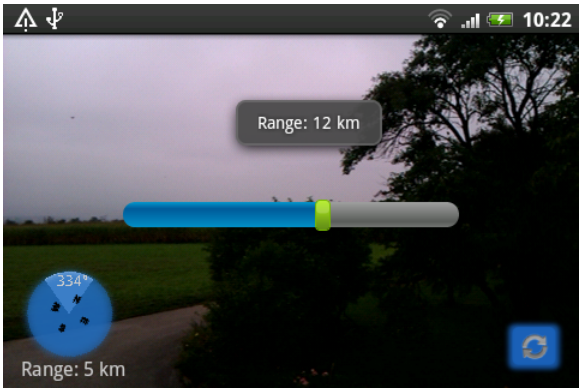


FIGURE 6.6: Range setter bar

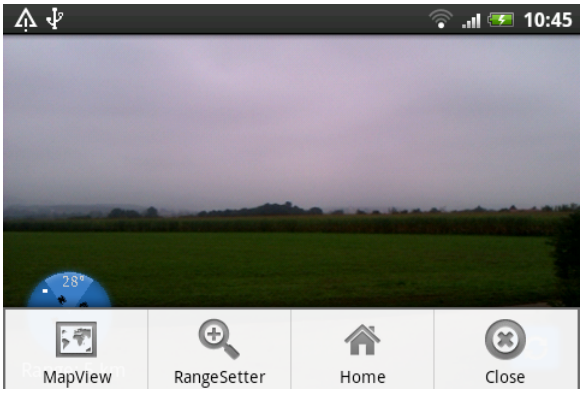


FIGURE 6.7: AR mode - options menu

Compass Rose/Refresh Button

Another feature is the animated compass rose on the left bottom side of the AR interface (cf. Figure 6.6). The rose contains information about the current heading of the device, and the little white dots on the compass rose represent recognized mountains in the near vicinity of the device. Moreover, on the right bottom side of the AR interface in Figure 6.6 there is a refresh button visible which can be used to repeat the loading and visualizing process of LOD resources on the screen.

Option Menu

The AR mode also offers an option menu (cf. Figure 6.7) containing options (starting from the right side of the menu) for:

- leaving the application,
- activating the home screen,
- changing the range,
- activating the map mode to visualize the mountains on a 2D map.

[2]The range in which real-life mountains can be referenced with LOD resources.

6.1.2 Map and Tour-Guide Mode

If the user selects the map view or tour guide button of the home screen, the map mode will be activated. The map mode manages the visualization of mountain resources or GPS tour tracks on a 2D map.

In the map mode, the system loads mountain data from LOD sets and visualizes the resources on the map by placing markers on the respective positions of the map (cf. Figure 6.8).

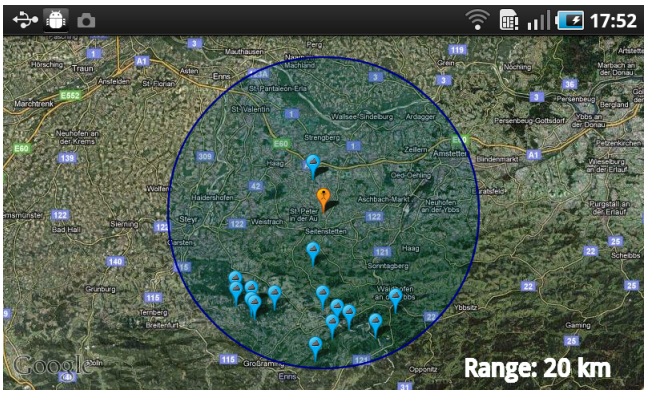
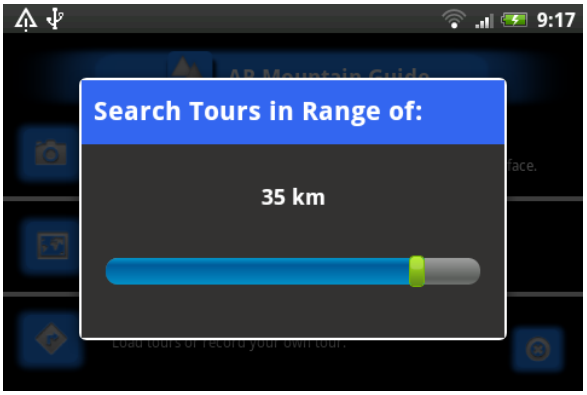


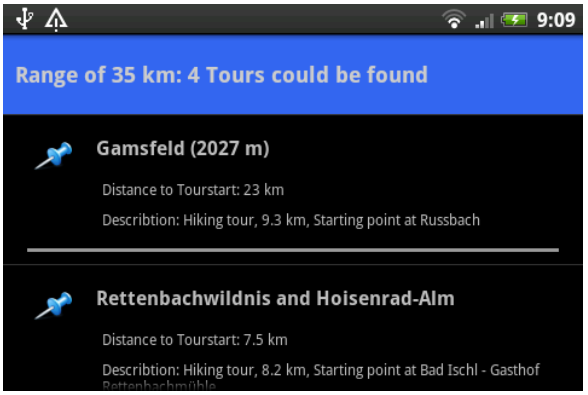
FIGURE 6.8: Map mode of AR application

The orange marker represents the current user position and the blue markers are the mountain resources. In addition, there is a circle visible which indicates the current chosen range. If the user selects a mountain marker the system fetches, as within the AR mode, further information from DBpedia and displays it within a dialog.

If the user selects the last button of the home screen (cf. Figure 6.1), the system asks if the user wants to load tours from the Web or if she/he wants to record GPS tours. In case the user selects the tour loading option, a dialog (cf. Figure 6.9 (a)) appears. The user has to select a range (in kilometers) on the seek bar of Figure 6.9 (a), so that the system is able to load data for a certain restricted range.



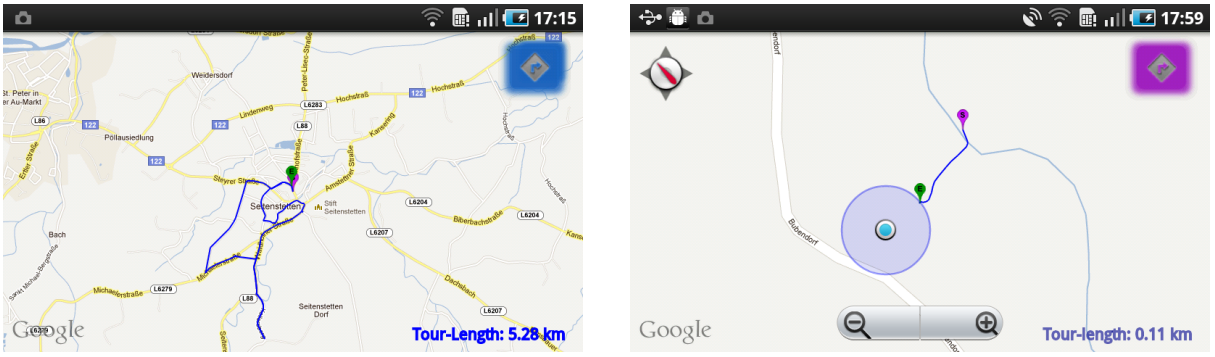
(a) Seek bar for setting tour loading range



(b) Tour tracks presented in a list view

FIGURE 6.9: Loading of tours

After the loading of tour tracks is completed, the system presents the results within a list, as described in Section 5.2.6. An example list view is visible in Figure 6.9 (b), where each item of the list comprises the title of the loaded tour track, followed by the distance to the tour start, and a short description of the track. If the user selects a tour of the list, the system visualizes the GPS tour on the map (cf. Figure 6.10 (a)).



(a) GPS track displayed on the map

(b) Tour guide mode is activated

FIGURE 6.10: Tour mode

Figure 6.10 (a) also illustrates a blue button on the right upper side of the AR interface. This button can be used to activate the tour guide mode which means that the system displays a marker on the map which represents the current user position (cf. Figure 6.10 (b)). Also a compass appears on the screen to additionally support the user with the orientation. As soon as the user walks in reality to another location, the marker moves to the respective position on the map. Thus, the user is able to take a guided tour by following the tour track displayed on the screen.

6.1.2.1 Additional Features

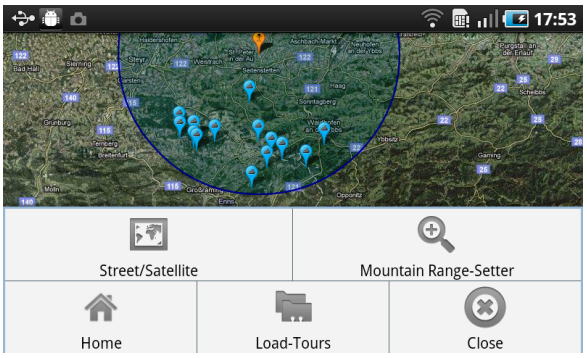
Option Menu

The map mode also offers an option menu (cf. Figure 6.11 (a)). It comprises options (starting from the right bottom side of the menu) for closing the whole application, for loading tours (from OpenStreetMap or from the local storage system), for activating the home screen, for setting a new range in which mountain resources are loaded, and for switching between street or satellite map views.

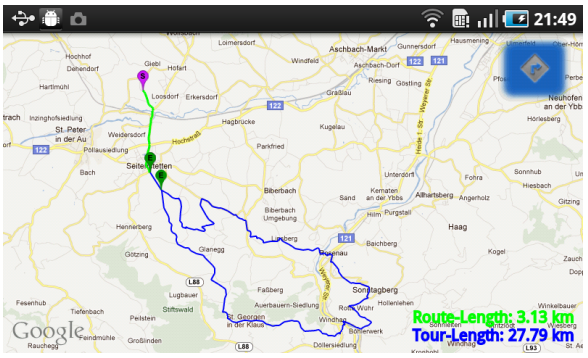
Route Calculation

If a tour is displayed on the map, the user has the additional option to calculate the route to the tour start. We utilize a Google Maps Web service[3] to calculate the route between the current location of the device and the tour start location. In Figure 6.11 (b), we illustrate that we use

[3]Google Maps route calculation service: `https://developers.google.com/maps/documentation/directions/#Routes`



(a) Option menu of map mode



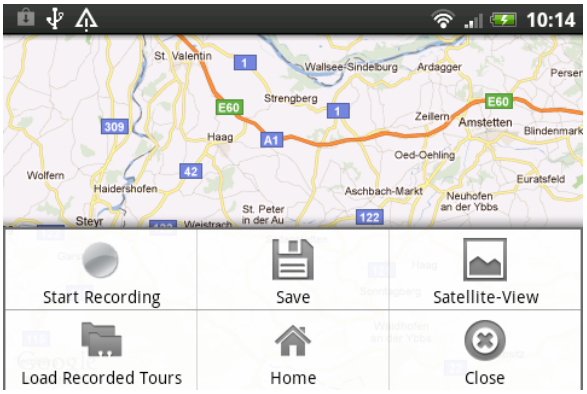
(b) The calculated route to the tour-start visualized in green

FIGURE 6.11: Tour mode features

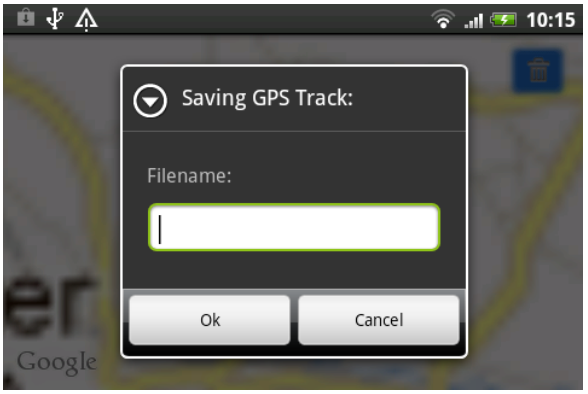
a green color for visualizing the additionally calculated route to the tour-start. In addition, the system displays the path lengths of both paths (tour track and calculated route to tour-start) on the right bottom side of the screen.

Recording Tours

The AR prototype also provides the possibility to record tours, and to save them as GPX files on the local storage system of the device. Through the option menu (cf. Figure 6.12 (a)) the user is able to start the recording of a GPS track. After activating the record mode, a first location fix is searched by the GPS radio of the device. Then, a blue glowing dot appears on the map to display the current position on the map. If the user moves, the blue dot on the map moves to the respective direction on the map. We highlight the already passed path by linking the recorded GPS coordinates with blue lines. As soon as the user stops the recording, she/he is able save the passed path in a GPX file through entering a name for that file, as illustrated in Figure 6.12 (b).



(a) Option menu of tour recording mode



(b) Dialog to enter GPX file name

FIGURE 6.12: Tour mode - recording tours

6.2 Evaluation

During the evaluation, we focused on two main criteria: first, the duration for processing the downloaded resources from LinkedGeoData and GeoNames, and secondly, the performance of the sensor-based AR tracking method which means the measuring of the duration until virtual content stabilizes on the correct position within the AR interface of our AR prototype. Additionally, we performed the evaluation on two different devices: Samsung Galaxy SL I9003[4] and HTC Wildfire S[5]. We used two devices in order to find out whether our AR system exhibits similar data processing and visualization performance behaviors on devices where each comprises a processor that runs at a different clock speed.

The evaluation section of this work is structured as follows: in Section 6.2.1, we describe our test environment of the evaluation. We dedicate Section 6.2.2 to the evaluation of the LOD resource processing performances of our prototype, where we firstly explain the test setup (cf. Section 6.2.2.1) and then we describe details about the test data that we used to simulate the data processing live on the test devices (cf. Section 6.2.1). At last, we present the results of the data processing evaluation in Section 6.2.2.2. In Section 6.2.3, we discuss the evaluation of sensor-based tracking performances of the prototype. We begin with the outlining of the test setup (cf. Section 6.2.3.1) and we conclude this section with the presentation of the results of the visualization performance tests (cf. Section 6.2.3.2).

6.2.1 Test Environment

In the following, we present the two devices we included in the evaluation.

The Samsung Galaxy SL I9003, released in February 2011, is powered by a 1 GHz ARM Cortex-A8 processor with PowerVR SGX530 GPU. This device provides 512 MB of RAM (478 MB of RAM to the user) and it is shipped with the Android operating system Froyo (Android 2.2). It exhibits a 4 inch 16M-color super clear LCD capacitive touchscreen with a resolution of 480x800 pixel. The Samsung Galaxy I9003 is additionally equipped with a 5 MP camera allowing 2592x1944 pixels and a GPS radio with A-GPS support. Internet access is provided through 2G GPRS/EDGE (Class 12) or 3G HSDPA (7.2 MB/s), and Wi-Fi (802.11 b/g/n) [103].

We also used the HTC Wildfire S for the evaluation of our AR prototype. This HTC smartphone was released in February 2011 and is powered by a 600 MHz Qualcomm MSM7227 processor and provides 512 MB RAM (150 MB of storage available to the user). This device uses the Andreno 200 GPU for graphics. It exhibits a 3.2-inch TFT capacitive touchscreen (supports 256 K colors) with a resolution of 480x320 pixels and it comes with the Android

[4]Technical specification of the Samsung SL I9003: `http://www.samsung.com/at/consumer/mobile-phone/mobile-phone/smartphones/BGT-I9003-spec?subsubtype=galaxy`

[5]Technical specification of the HTC Wildfire S: `http://www.htc.com/www/smartphones/htc-wildfire-s/#specs`

operating system Gingerbread (Android 2.3). The Wildfire S is also equipped with a 5 MP camera (allowing 2592x 944 pixels) and a GPS radio. Connection to the Internet is possible via Wi-Fi (802.11 b/g/n) and mobile broadband with a 2G GPRS/EDGE (Class 10) connection or a 3G HSDPA (7.2 MB/s) connection [104].

The two devices feature the regular sensors such as accelerometer, proximity sensor, digital compass, and ambient light sensor.

6.2.2 Data Processing Performance

In this section, we present the evaluation of the data processing performance of our AR system. This evaluation exhibits if our prototype is capable of processing LOD resources in reasonable time on the two different mobile phones presented in Section 6.2.1. With processing performance we mean the time the prototype needs for parsing LOD resources from a file into an Androjena model, for transforming the resources of the Androjena model into Java objects, for the identification and merging of identical resources, and for storing the processed resources locally on the mobile phone.

We do not evaluate the downloading performance of resources from the Web because this depends for instance on the Internet connection speed and the available bandwidth at the time the evaluation is taken which does not lie in the focus of our evaluation.

6.2.2.1 Test Setup and Test Data Preparation

This evaluation includes the analysis of the execution time of different LOD resource processing operations:

- Parsing data from a local RDF file into an RDF model.
- Filtering identical resources and merging identical resource descriptions.
- Processing (saving as Java object) the resources.
- Storing the resources in a SQLite database on the device.

In particular, we execute these operations with different amounts of resources:

20, 100, 200, 500, 1,000, 1,500, 2,000

For instance, during a test round where 20 resources are used, we process 10 resources originally downloaded from LinkedGeoData and 10 resources fetched from GeoNames. Thereby, a single resource is described by several triples. Thus, in total each test round includes the processing of 170, 850, 1,700, 4,250, 8,500, 12,750, and 15,000 triples.

We fetched the LOD resources, used for this evaluation, from the LinkedGeoData and GeoNames platforms in RDF/XML-serialized format. An example, RDF graph of data from LinkedGeoData is visible in Listing 5.6 and from GeoNames in Listing 2.1. In total, we have

14 different RDF/XML-based files containing test data in the amounts of 10, 50, 100, 250, 500, 750, and 1,000 resources per data set. Furthermore, each LinkedGeoData resource is described by 5 triples and each GeoNames resource by 12 triples. For instance, a test file that contains 10 different resources from LinkedGeoData contains 50 triples and a test file of 10 resources from GeoNames 120 triples. The test data dumps also contain identical resources (two resources that describe the same real-world object but exhibit different URIs) to be able to evaluate also the processing duration of the filtering of identical resources during a test round.

The data processing evaluation consists of 10 test rounds for each test data group and in each test round the following operations are performed: first of all, the AR system parses the LinkedGeoData and GeoNames test resources from 2 files (stored on the mobile device) into two separate Androjena models (cf. Section 5.2.2). The next task is to process the LinkedGeoData resources of the model which means that the model is iterated and the label and the geographical coordinates of each resource are saved as Java object of type `MountainObject` (cf. Section 5.2.4). Then, these objects are saved within a Java array list (cf. Section 5.2.2). The following processing step is the identifying of identical resources from the two different test data sources. Thereby, the method presented in the previous chapter in Listing 5.9 is performed to filter identical resources and to merge resource descriptions if a GeoNames resource is identical to an already processed LinkedGeoData resource. Then, the system transforms the remaining GeoNames resources into Java objects (of type `MountainObject`) in order to add the newly processed GeoNames objects also to the Java array list that contains the already processed LinkedGeoData resources.

Every data processing duration is written into a log file on the local storage system of the device. Thereby, we generated log files for every test data dump size. In the end, we obtained 14 different log files (7 per device) containing data processing evaluation results.

For our data processing evaluation, we defined several evaluation symbols that we present and explain in Table 6.1.

Symbol	Definition / Unit
$\overline{t}_{parsing}$	Arithmetic mean for parsing of resources / in ms
$\overline{t}_{processing}$	Arithmetic mean for processing of resources / in ms
$\overline{t}_{identical}$	Arithmetic mean for detecting identical resources / in ms
$\overline{t}_{storing}$	Arithmetic mean for storing resources on the device / in ms
$\overline{t}_{total}$	Total processing duration of resources / in ms (cf. Definition 6.1)
RES_{psec}	Set of all resources processed per second / in quantity

TABLE 6.1: Evaluation Parameters

The evaluation symbols for describing the processing durations in milliseconds present the arithmetic mean of processing durations obtained from 10 conducted test rounds. We determine the total processing duration $\overline{t}_{total}$ by summing up all data operation durations (cf. Definition 6.1).

Definition 6.1. Let $\bar{t}_{parsing}$ be the arithmetic mean of 10 rounds of parsing LinkedGeoData and GeoNames resources in an RDF model. Let $\bar{t}_{identical}$ be the arithmetic mean of 10 rounds of filtering identical resources, and let $\bar{t}_{processing}$ be the arithmetic mean of 10 rounds of processing the resources. We obtain $\bar{t}_{total}$ by Equation 6.1:

$$\bar{t}_{total} = \bar{t}_{parsing} + \bar{t}_{identical} + \bar{t}_{processing} \tag{6.1}$$

Parallel to these operation steps, the system stores data into a SQLite database (cf. Section 5.2.3). The system performs the storing in a background thread and thus do not has impact on the processing duration of parsing, processing, and identifying of identical resources. This is why we do not add this operation duration to the total amount of processing durations in Equation 6.1.

6.2.2.2 Data Processing Results

All tests were performed on two different devices presented in Section 6.2.1. In Table 6.2, we list the results of the data processing tests. As expected, all operation steps took more time on the less powerful HTC device than on the Samsung device. Furthermore, on both devices, results for $\bar{t}_{parsing}$ are always higher than results for $\bar{t}_{processing}$ and $\bar{t}_{identical}$. As explained in Section 6.2.2.1, resources are described by several triples. Therefore, parsing the resources into the model depends on the amount of triples. As soon as the triples are in the Androjena model, we do not access all the triples but specific ones like the ones containing the label and the geo-coordinates of a resource (cf. Listing 5.7). The system also makes use of label and geo-coordinates of a resource during the process of detecting identical resources. Thus, during the processing and filtering operation steps, fewer data are processed which leads to lower processing durations.

Resources		**20**	**100**	**200**	**500**	**1,000**	**1,500**	**2,000**
Samsung	$\bar{t}_{parsing}$	472	2,342	4,995	11,519	24,520	38,518	51,215
	$\bar{t}_{processing}$	112	569	1,037	2,658	6,465	9,564	13,073
	$\bar{t}_{identical}$	76	369	735	4,499	22,259	38,096	65,776
HTC	$\bar{t}_{parsing}$	4,354	20,482	39,786	102,832	211,150	320,625	446,888
	$\bar{t}_{processing}$	628	2,854	5,684	17,811	28,179	41,033	57,902
	$\bar{t}_{identical}$	342	3,599	10,080	59,114	203,134	468,091	816,177

TABLE 6.2: Data processing durations

The average time for processing of resources $\bar{t}_{processing}$ (saving the URI, label, and geo-coordinates of a resource as Java object) exhibits the fastest processing durations in all test data groups on both devices. Thereby, $\bar{t}_{processing}$ reveals reasonable processing durations for test data dump sizes up to 1,000 resources on the Samsung device and up to 200 resources on the HTC device.

Table 6.2 exhibits that the average processing time $\bar{t}_{identical}$ is acceptable for identifying identical resources when processing 20, 100, 200, and 500 resources on the Samsung device and 20, 100, and 200 resources on the HTC device. But $\bar{t}_{identical}$ increases to an unacceptable height when filtering identical resources in higher amounts of resources. The higher the amount of resources the more labels have to be compared. Furthermore, our method for identifying identical resources (cf. Listing 5.9) includes the calculation of the geometrical distance between two resources that exhibit a low Jaro-Winkler distance (e.g. a Jaro-Winkler distance of 0 equates to similarity). So, this additional calculation step within the identical search process also contributes to the higher processing durations, if there are several possible equal resources in the two data dumps.

In Table 6.3, we present the total processing time $\bar{t}_{total}$ of resources from all operation steps (cf. Definition 6.1) but also how much resources per second RES_{psec} our AR system was able to process within each test data group.

Resources		**20**	**100**	**200**	**500**	**1,000**	**1,500**	**2,000**
Samsung	$\bar{t}_{total}$	660	3,280	6,766	18,675	53,243	86,178	130,064
	RES_{psec}	30.30	30.49	29.56	26.77	18.78	17.41	15.38
HTC	$\bar{t}_{total}$	5,325	26,934	55,550	179,756	442,463	829,749	1,320,967
	RES_{psec}	3.76	3.71	3.60	2.78	2.26	1.81	1.51

TABLE 6.3: Comparison of total data processing time

Generally, the total data processing duration $\bar{t}_{total}$ on the Samsung device was acceptable when loading up to 200 resources. On the HTC device, the total processing duration $\bar{t}_{total}$ was only acceptable when using 20 test resources. As we restricted our prototype into loading not more than 50 resources, we additionally determined $\bar{t}_{total}$ for 50 resources on the HTC device which resulted in about 13 seconds. This time is acceptable but nevertheless a bit high. In Figure 6.13, we illustrate the different processing durations of $\bar{t}_{total}$ on both devices where we marked the size of the test data dumps on the x-axis and we displayed the processing durations in milliseconds along the y-axis. On the HTC device the processing durations clearly took more time for all data dump sizes than on the Samsung device because the HTC exhibits a weaker CPU performance.

Additionally, we present in Figure 6.14, the number of processed resources per second on both devices where the size of the test data dumps is marked on the x-axis and the processed resources per second ratios are displayed along the y-axis. The figure illustrates that on the Samsung device the system was able to process much more resources per second than on the HTC device. Moreover, the figures show that on both devices there is a linear dependency between the number of resources and their processing time recognizable. This means both devices become slower in processing data the more test resources we load.

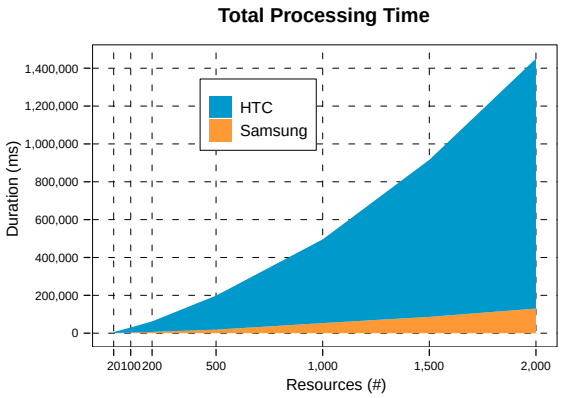


FIGURE 6.13: Total resource processing durations

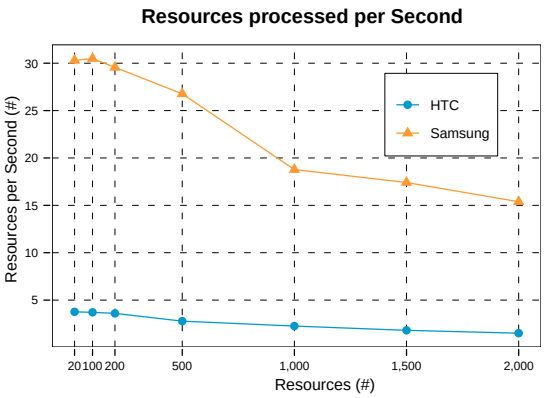


FIGURE 6.14: Number of resources processed per second

In Table 6.4, we present the processing durations for storing resources in the database on the local storage system of the device. As explained in the previous section, the system performs the storing of data in a separate background process. Thus, we present the results for the storing processes in a separate table. The amounts of test resources, presented in the table, are lower than the original test data dump sizes because we store data after the identifying of identical resources. If the system detects identical resources, it adds the additional descriptions to the respective resource within the database. In consequence, resource descriptions are merged.

As expected, the HTC device needs more time for storing data than the Samsung device. With an amount of 467 resources the HTC device already needs about 2 minutes and the Samsung device only about 14 seconds to store the data. Thus, the HTC device is in this case about eight times slower than the Samsung device. Storing higher amounts on the HTC device ends in unacceptable storing durations. $\bar{t}_{storing}$ is acceptable for test data dump sizes up to 467 resources on the Samsung device and on the HTC device up to 93 resources.

Resources		**18**	**93**	**197**	**467**	**929**	**1,368**	**1,842**
Samsung	$\bar{t}_{storing}$	496	2,655	5,590	13,997	29,256	45,293	68,044
HTC	$\bar{t}_{storing}$	1,649	7,302	27,724	115,570	166,385	301,654	343,567

TABLE 6.4: Data storing durations

6.2.3 AR Visualization Performance

A basic execution process of our AR prototype is the visualization of AR content on top of real-world objects. Thus, the visualization performance concentrates on the execution time of our AR system until the AR content stabilizes on the proper position within the AR interface. This test procedure relies on human impact because it is based on pointing which means during the test the user hast to point the mobile device at pre-defined physical targets. We present these defined targets in the following section.

We performed this evaluation in order to prove that our AR prototype is capable of stabilizing AR content in reasonable time on top of the respective real-world objects.

6.2.3.1 Test Preparation

We defined five different test targets that we used to point the device at. In Table 6.5, we present all the test targets that we used during the visualization tests. These targets are all physical objects located in Austria and we classify them according to the three different terms *peak*, *place*, and *monument*. In Table 6.5, we present the name, the location, the elevation, and the type of classification of each test target.

Target	Location (lat/long)	Elevation	Classification
Sonntagberg	47.996183°,14.761759°	580 m	place
St. Michael	48.015033°,14.614883°	545 m	place
Stift Seitenstetten	48.035612°,14.654921°	334 m	monument
Wetterkogel	47.882967°,14.783077°	1,111 m	peak
Plattenberg	48.016454°,14.553695°	600 m	peak

Table 6.5: Test target definition

In Listing 6.1, we present how we defined the test targets programmatically. We directly stored these test targets as Java objects and then we added them to the `mountains` array list which contains all the pre-processed resources for a certain restricted range near the device (cf. Section 5.2.4). The AR system iterates the `mountains` array list and visualizes every object within this list on the screen, as soon as the real-world object appears in the field of view of the device's camera.

```
1  Location location1 = new Location("Sonntagberg");
2  location1.setLatitude(47.996183);
3  location1.setLongitude(14.761759);
4  location1.setAltitude(580);
5  MountainObject object1 = new MountainObject(null, location1, "Sonntagberg");
6  ARView.mountains.add(object1);
```

Listing 6.1: Defining of a test object in our AR system

During the evaluation, the user has to pan the device because the panning causes the sensor values for heading and orientation to change. Thus, the icon is moving on the screen as new screen coordinates are calculated at every sensor value change. Now, in this evaluation, we measure the time until the icon stabilizes upon the respective physical target object.

Moreover, this test procedure requires user input: at the beginning the user has to determine an admissible target region on the device screen. Touching the screen causes the visualization of a box (30x30 pixels) around the physical target region that the user currently captures with the device's camera. During the test execution, the test target is presented as graphical icon

on the screen. The test includes that the user pans the device. Thereby, we distinguish two different pan directions: left and right. A left or a right pan of 180° from the current test target causes the test to start. As soon as the test starts, the user has to pan back to the test target. The moment the user reaches the starting position and the icon is visualized on top of the test target within the previously defined box, the visualization succeeds which causes the test to stop. We repeated the tests for every test target and every pan direction 10 times.

The user has to initialize every test round manually, whereby the total amount of time until the icon is visualized on the proper position on the screen of each test round is written into a log file and stored on the local storage system of the device. For every test target and every pan direction, a separate log file was generated.

To evaluate the AR visualization performance of our approach, we define 4 mathematical symbols listed in Table 6.6. We obtain these values for the evaluation parameters directly through our implemented AR prototype, as in the previous evaluation presented in Section 6.2.2.

Symbol	Definition / Unit
$\bar{t}_{HTC}$	Arithmetic mean of visualization performances on HTC / in ms
$\bar{t}_{Samsung}$	Arithmetic mean of visualization performances on Samsung / in ms
σ_{HTC}	Standard deviation of visualization performances on HTC / in ms
$\sigma_{Samsung}$	Standard deviation of visualization performances on Samsung / in ms
le	Left pan of device
ri	Right pan of device

TABLE 6.6: Evaluation parameters for visualization performance

6.2.3.2 Visualization Performance Results

We summarize the results of the visualization tests in Table 6.7.

Test Target	Stift Seitenstetten		St. Michael		Sonntagberg		Plattenberg		Wetterkogel	
Pan Direction	le	ri	le	ri	le	ri	le	ri	le	ri
$\bar{t}_{Samsung}$	5,831	6,583	5,867	6,043	5,454	5,920	6,029	6,229	5,136	6,432
$\sigma_{Samsung}$	907	576	947	874	1,072	857	966	1,610	947	1,488
$\bar{t}_{HTC}$	6,289	5,178	5,714	5,201	6,388	5,407	6,586	6,033	5,535	5,447
σ_{HTC}	752	759	878	588	594	712	900	649	1,224	870

TABLE 6.7: Visualization performances in ms

Table 6.7 exhibits that all average performance durations $\bar{t}_{HTC}$ lie between 5.2 and 6.6 seconds and all average performance durations $\bar{t}_{Samsung}$ lie between 5.1 and 6.6 seconds. Thus, our system exhibits a constant AR content visualization performance for a proper visualization

of icons on top of the real-world objects on both test devices. In Figure 6.15 and Figure 6.16, we illustrate the arithmetic mean of left and right movements for each target object. Figure 6.15 exhibits that $\bar{t}_{Samsung}$ for the pan direction re is always a bit higher than for the pan direction le. On the HTC device, all the tests when performing le took longer than for the tests with pan direction re. Although, the difference of the average performance durations of the test rounds when performing left or right pans is minimal, we ascribe this trend to the sequence of conducting the two different tests. Because on the Samsung device we firstly performed the tests for the right panning of the device and on the HTC we started with the left panning of the device. When performing 10 times left movements, the user gets used to it and thus she/he pans the device a bit faster during the second evaluation test rounds with the different pan direction. The visualization performance results on the HTC device are nearly equal to the results, when performing the tests on the Samsung device. Thus, this time the less powerful HTC does not exhibit any kind of weakness which means that our AR system provides a proper and equal AR content visualization performance for both devices.

We illustrate the standard deviations $\sigma_{Samsung}$ and σ_{HTC} of Table 6.7 together with the arithmetic mean results in Figure 6.15 and Figure 6.16. Within these figures, the test targets are displayed along the x-axis where each test target comprises a bar for left and right pan directions and the visualization durations in milliseconds are marked on the y-axis. Both figures illustrate the standard deviation of the individual sample means. The values for the standard deviation σ_{HTC} lie between 0.6 and 1.2 seconds and for $\sigma_{Samsung}$ between 0.6 and 1.6 seconds. Thus, the variability of the individual performance durations is acceptable.

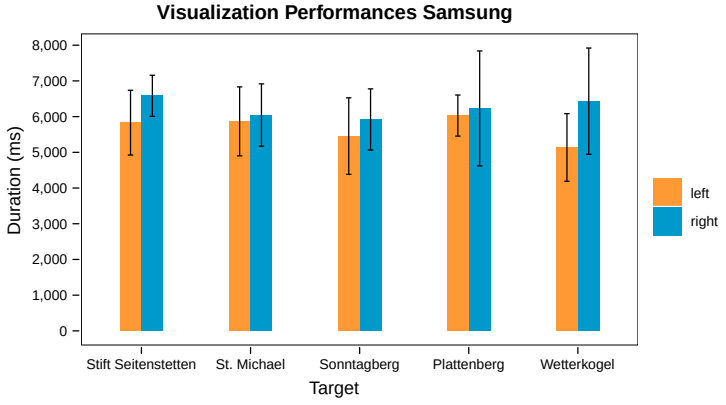


FIGURE 6.15: Visualization Performances on Samsung SL I9003

The small time variations that we nonetheless obtained in the different test rounds can be ascribed to the panning of the devices by the user that each test requires. Not in every test round, the user is able to pan back to the starting point in the same amount of time. Moreover, each slightly movement of the device causes a re-calculation of screen coordinates of the AR content. The user has to make smooth and calm movements so that the AR content stops moving around on the screen and stabilizes itself on top of the target object.

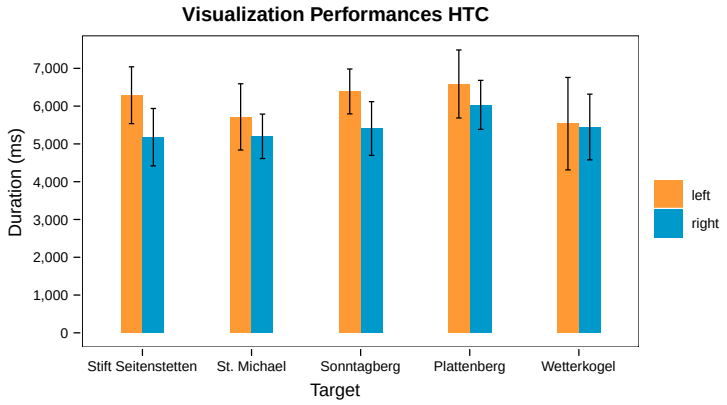


FIGURE 6.16: Visualization Performances on HTC Wildfire S

Sometimes, the alignment of icons and real-world objects can be inaccurate. Reasons for inaccurate AR content visualizations are for instance imprecise recorded direction and orientation sensor values or inaccurate location and GPS information. We discuss these issues in the following paragraphs.

Inaccurate Orientation Values

As we already mentioned in Section 5.2.1.2, we use the magnetometer of the device in combination with the accelerometer to provide magnetic north orientation. Magnetometers pick up every possible magnetic field in the near environment. This leads to interferences with the sensor values and inaccurate orientation estimations. Magnetic fields interfere with the earth's magnetic field all the time, whereby interferences can be caused by cars, metal structure of buildings, the built-in Bluetooth chip or speakers of a mobile device itself.

To reduce the sensor noise, we use a low pass filter (cf. Section 5.2.1.2 in Listing 5.4). The low pass filter eliminates fast changing sensor values and allows only slow changes of values. To be able to show the effects of the low pass filter, we recorded raw and filtered azimuth and pitch values and put them together in two diagrams (Figure 6.17 and Figure 6.18). Within both figures, we illustrated the amount of recorded sample sets along the x-axis and we marked the measured degrees of azimuth or pitch on the y-axis. As visible, the filter method filters sudden changes in the sensor reading out. We obtain a more accurate orientation estimate through the filter than from the raw sensor values alone. Additionally, the system is more robust in case of sudden fast motions.

Inaccurate GPS Location Information

Our AR tracking approach contains the distance and bearing calculation to the real-world objects. This requires accurate location information of the device and the resource. In some situations, the location information of the resources, provided by the LOD sets, is not exact.

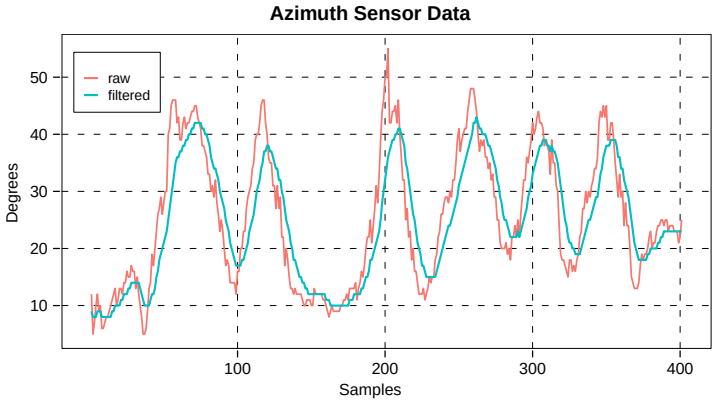


FIGURE 6.17: Raw and filtered Azimuth sensor values

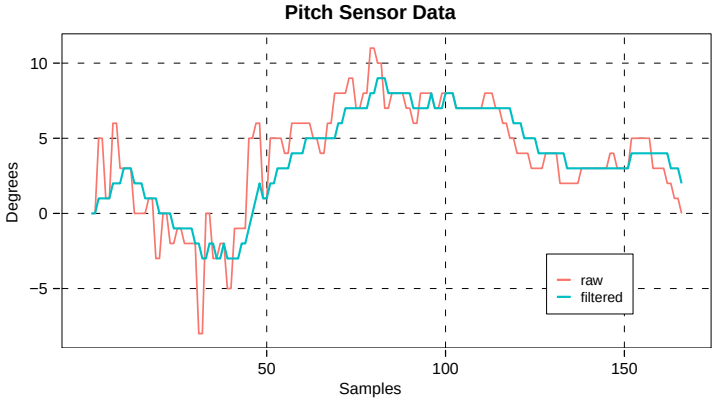


FIGURE 6.18: Raw and filtered Pitch sensor values

Sometimes different data sets provide different location information for the same resource and it is not certain which is more accurate. We are not able to measure the quality of the geographical information of location resources that are provided by the LOD sets. Hence, to make our system more accurate, we merge resource descriptions of identical resources. As a result, some resources exhibit two different geo-coordinates (one pair from LinkedGeoData and the other from GeoNames). During the visualization process the system takes the geo-coordinates with the longest digits (the highest number of decimal degrees) for further calculation steps because the higher the number of the decimal degrees is the more accurate the geo-coordinates are (cf. Section 5.2.2.3).

In addition, the recorded GPS location of the mobile device can also be inaccurate. As explained in Section 2.3.2, the GPS radio of the device needs to lock with more than 3 satellites in order to determine an accurate location. Clear view of sky enables better lock possibility with enough GPS satellites. The more satellites the GPS receiver locks with the better it is. Buildings and trees can interfere with the GPS connection to the satellites and can lead to inaccurate or non-existing location information. However, the quality of the positioning service not only depends on external conditions but also on the hardware of the used mobile device

(quality of the phone's GPS radio). Thus, we do not evaluate the accuracy of recorded GPS location information.

6.3 Summary

Our evaluation does not contain any processing operations that requires a network connection. The use of third party Web services generally may end up in slow, unstable or totally broken connections because of server errors on the provider side that would influence our evaluation results.

The evaluation of our proof-of-concept prototype includes LOD processing operations on two different devices (cf. Section 6.2.1). The data processing operations took an acceptable amount of time when processing up to 200 resources on the Samsung device. The less powerful HTC device already exhibited weakness when processing about 100 resources. In reality, 200 visualized resources overload the screen thus we limited the amount of resources that actually can be visualized to 50 and we additionally provide a range limit of 20 km.

We also evaluated the performance of AR content visualizations on both devices. This evaluation test relies on human impact. The user has to pan the device 180° until she/he captures the test target and the AR content stabilizes on top of the test object. The average duration for each test target took about 5 to 6.5 seconds until the icon is visualized on top of the respective real-world object on both devices. Thus, our AR prototype exhibits a constant visualization performance for the higher and less powerful test devices used in this evaluation.

However, these tests are affected by the user's movements of the device. Thus, the variations of the results between the test rounds depend on the time the user needs until she/he pans the device back to the target resource and on how smoothly she/he pans the device. The variations between the test results can also be caused by noisy orientation and direction sensor values. To prevent sensor value inaccuracies we use a low pass filter (cf. Section 5.2.1.2) which eliminates sudden sensor value changes. As a result, we obtain more accurate orientation estimates.

Incorrect icon visualizations can also be caused by inaccurate GPS location information. As LinkedGeoData and GeoNames are community driven projects, the provided geographical coordinates of some resources might not be accurate enough. Fortunately, users are able to contribute and to improve the quality of data by correcting inaccurate resource descriptions. In addition, the accuracy of the recorded GPS location information of the device's GPS radio depends on the quality of the provided hardware of the device and on the external conditions (is there free view of sky?) during the execution of the application.

In conclusion, we could observe that our prototype is capable of processing LOD resources on the device and of visualizing them on proper positions in reasonable time on mobile phones. Thus, we were able to proof the feasibility of our approach of integrating Linked Data in a mobile sensor-based AR system in order to identify and describe real-world objects (based on the LOD resource descriptions) in the near physical environment.

Chapter 7

Final Remarks

7.1 Conclusions

In this work, we investigated the integration of Linked Data into a mobile AR system. The Web of Linked Data offers a huge amount of interlinked data and the LOD project is growing year by year. Therefore, LOD presents an attractive data source for AR systems. Based on Linked Data resources, it is possible to identify and describe real-world objects within AR interfaces.

In this thesis, we provided an overview on the different kinds of AR systems that exist and discussed problems and limitations of current approaches (cf. Chapter 3). Vision-based AR systems rely on complex image pattern algorithms and mostly depend on an external server infrastructure. Such systems can be very resource consuming for the mobile phone's hardware. As a consequence, we put the focus on sensor-based AR systems which rely solely on the built-in sensors of a mobile device. In our work, we gave an overview on existing sensor-based AR systems for mobile phones and compared them with our approach (cf. Chapter 3). Most commercial mobile AR browsers use proprietary standards for describing AR content and they fetch data from isolated data repositories. Moreover, they mostly rely on client-server architectures and require permanent network connections. On the other hand, exploiting LOD to use it as AR content in AR systems such as our approach aims at is rarely investigated.

In consequence, we discussed how we are able to exploit LOD resources and presented background information on the structures, ontologies, and technologies of LOD (cf. Chapter 4). Federating LOD resources from different sources and merging them leads to the availability of duplicate resources. Hence, we introduced an existing framework (Silk-framework cf. [90]) that provides tools for identifying identical resources in different LOD sources in order to be able to merge resource descriptions. In our work, we identify identical resources by comparing their labels using the Jaro-Winkler distance and we additionally determine the geographical distance between the two resources if the Jaro-Winkler distance metric does not result in a perfect match (a Jaro-Winkler distance value of 0) (cf. Chapter 5). If the system identifies two resources as identical, it merges the resource descriptions of these two resources.

On the basis of the assumption that we focus on sensor-based AR systems, our work includes the presentation of a conceptual model. Thereby, we outline that the AR system comprises a mathematical model which is capable of determining the current field of view of the device's digital camera based on the orientation and heading information of the device (cf. Chapter 4). Under the awareness of the current field of view, the model allows the calculation of screen coordinates for projecting LOD resources into the AR interface of the system. We described this mathematical model on a formal basis where we introduced and defined all including calculations through formal notation.

As a proof of concept, we implemented the proposed conceptual and formal model as a AR application for Android mobile phones (cf. Chapter 5). The prototype presents an AR mountain guide and is capable of identifying mountains in the near surroundings. The system accesses three different LOD platforms whereby it uses LinkedGeoData and GeoNames for fetching geo-coordinates to identify real-world objects, and it uses DBpedia to deliver further resource descriptions about the visualized objects. Our AR system includes methods for federating, consolidating (identifying and merging identical resources), and storing LOD resources directly on the mobile device. It is possible to execute the prototype without permanent network connection. Moreover, the application is able to display GPS tracks on a 2D map. The GPS tracks are fetched from OpenStreetMap. Additionally, it is possible to record GPS tracks and store it on the mobile phone's local storage system.

Finally, we conducted an evaluation in order to give insights on the performance of the AR prototype (cf. Chapter 6). We tested the data processing execution which focuses on the parsing, consolidation, and storing of LOD resources on the mobile device. The results of this evaluation revealed that the prototype is able to process a sufficient amount of LOD resources in acceptable time on two different mobile devices (Samsung SL i9003, HTC Wildfire S). The second evaluation concentrates on the AR content visualization performance. Thereby, we determined the time the AR systems needs to stabilize visualized LOD resources on top of proper screen positions (on top of the respective real-world objects). This evaluation showed that our prototype allows visualizations in reasonable time live on top of the captured view of a more powerful (Samsung SL i9003) but also a less powerful (HTC Wildfire S) smartphone.

Consequently, we are able to affirm that the development of the AR prototype proofs the feasibility of our approach. We were able realize the research objectives of this work which we defined in Section 1.4 and we implemented the functional requirements presented in Section 4.1 in order to enable the realization of an AR system for mobile phones. Hence, the main objective which was the integration of LOD resources into an AR interface of a sensor-based mobile AR system has been transposed. Therefore, our work contributes to the domain of Linked Data because we developed a new form of LOD exploitation and application but also to the domain of AR because we introduced a new possible way of standardizing AR content. Our work can be used as basis for ongoing research in the domains of LOD and AR. Therefore, we present several possible future works in the following section.

7.2 Outlook

Exploiting LOD in sensor-based mobile AR systems was successful and approved itself as a contribution to the domain of mobile AR. Nevertheless, we present several suggestions of possible future works in the following paragraphs:

In our work, we only use three different LOD sets for proving the feasibility of our approach. In future, the federation of LOD resources could be extended to much more LOD sets than three and therefore the process of consolidating the resources has to be improved.

Another suggestion is the integration of a feature that allows users to make annotations directly in physical space in order to contribute to the LOD project. Thus, the user could generate new resources, extend descriptions of existing resources or link existing resources with related ones directly through the AR interface of the AR system.

There is a need for open standards and APIs for AR applications. Wikitude's ARML (cf. Chapter 3) is a notable example for an AR content specification to describe POIs. RDF the graph-based data model used by the LOD project could also be deployed as AR standard for AR applications. RDF includes properties which allow to express geo-coordinates for a POI. Thus, by using RDF as AR content standard for mobile AR browsers it is possible to uniquely identify real-world objects by referencing them with LOD resources.

Concerning the feature of downloading GPS tours which our prototype provides, it would be useful if there exists a GPS track platform that offers an API for fetching specific tours. Or even a complete new approach could be the saving of GPS tracks within RDF/XML-based files and to store them in further consequence within a new Linked Data platform (e.g. like DBpedia) which then only provides interlinked GPS tours. A Web page (or RDF/XML-based file) that describes a single GPS tour could contain additional information about the tour like duration, distance, time, links to pictures of the tour, special features of the tour, points of interests that are located along the tour or the degree of difficulty. In addition, these tours could be interlinked with other similar tours, e.g. tours that have been recorded in the same geographical area.

As the prototype of this work enables the recording of GPS tours, it would be possible to integrate an interface that allows to upload self-recorded tracks to the proposed Linked Data platform that contains interlinked GPS tracks to increase the Web of Linked Data and to share tracks with everyone.

Furthermore, our prototype only allows the displaying of GPS tracks on a 2D map. Therefore, another possible future work could be the developing of an AR interface that integrates GPS tracks by augmenting the live view with visualized arrows and lines presenting the path and visualized signs on the side of the road. Concerning the domain of Linked Data, there could not only be displayed mountain data and navigation icons but also RDF-based data informing the user about nearby hostels.

Appendix A

CD Attachment

We have attached a CD-ROM to this work that contains our AR prototype as Android project. Executing this Android project through Eclipse requires the installation of the Android SDK and the ADT plugin for Eclipse. The attachment comprises all the necessary files and libraries to deploy the AR prototype on an Android (version 2.2 or higher) mobile phone. Detailed installation guidelines are summarized in a "README.TXT" file which is located in the root directory of the attached CD-ROM.

Appendix B

Curriculum Vitae

SILVIA MARIA FÜRST

Student at

University of Vienna, Faculty of Computer Science

Email: silvia.fuerst@gmx.at

PERSONAL INFORMATION

Born: September 30, 1986 in Steyr, Austria

Nationality: Austria

Single, no children

EDUCATION

2010 - 2013

Master studies at the University of Vienna in Media Informatics

2006 - 2010

Bachelor studies at the University of Vienna in Informatics

2001 - 2006

HLW Stadt Haag, Lower Austria – Key course element Spanish

Bibliography

[1] Harry E. Pence. Smartphones, Smart Objects, and Augmented Reality. *The Reference Librarian*, 53(1, 2):136–145, 2011.

[2] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space.* Morgan & Claypool, 1st edition, 2011.

[3] Ronald Azuma, Mark Billinghurst, and Gudrun Klinker. Editorial: Special section on mobile augmented reality. *Comput. Graph.*, 35(4):vii–viii, 2011.

[4] Freek Uijtdewilligen. *A framework for context-aware applications using augmented reality: A train station navigation proof-of-concept on Google Android.* PhD thesis, University of Twente, 2010.

[5] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, and Blair MacIntyre. Recent Advances in Augmented Reality. *IEEE Computer Graphics and Applications*, 21:34–47, 2001.

[6] Riku Suomela and Juha Lehikoinen. Taxonomy for visualizing location-based information. *Virtual Real.*, 8:71–82, September 2004.

[7] Gerhard Reitmayr and Tom W. Drummond. Going out: Robust model-based tracking for outdoor augmented reality. In *Proceedings of 5th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 109–118, 2006.

[8] Nishkam Ravi, Pravin Shankar, Andrew Frankel, Ahmed Elgammal, and Liviu Iftode. Indoor Localization Using Camera Phones. In *Proceedings of the Seventh IEEE Workshop on Mobile Computing Systems & Applications*, pages 19–, 2006.

[9] D. W. F. (Rick) van Krevelen and Ronald Poelman. A Survey of Augmented Reality Technologies, Applications and Limitations. *The International Journal of Virtual Reality*, 9(2):1–20, 2010.

[10] Martin Raubal und Ilija Panov. A Formal Model for Mobile Map Adaptation. *Location Based Services and TeleCartography II*, pages 11–34, 2009.

[11] Will Seager and Danae Stanton Fraser. Comparing physical, automatic and manual map rotation for pedestrian navigation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 767–776, 2007.

[12] Anthony J. Aretz and Christopher D. Wickens. The mental rotation of map displays. *Human Performance*, 5(4):303–328, 1992.

[13] Marvin Levinew, Iris Marchon, and Gerard Hanley. The placement and misplacement of you-are-here maps. *Environment and Behavior*, 16(2):139–157, 1984.

[14] Matt J. Rossano, David H. Warren, and Allison Kenan. Orientation Specificity – How general is it? *American Journal of Psychology*, 108(3):359–380, 1995.

[15] Juha Lehikoinen. An evaluation of augmented reality navigational maps in head-worn displays. *Proceedings of IFIP INTERACT01: Human-Computer Interaction*, 2001.

[16] Oulasvirta Antti, Estlander Sara, and Nurminen Antti. Embodied interaction with a 3D versus 2D mobile map. *Personal and Ubiquitous Computing*, 13:303–320, 2009.

[17] Daniel Wagner. *Handheld Augmented Reality*. PhD thesis, Graz University of Technology, 2007.

[18] Ben Butchart. Augmented Reality for Smartphones. In *Institutional Web Managers Workshop*, 2011.

[19] Alex Hill, Blair MacIntyre, Maribeth Gandy, Brian Davidson, and Hafez Rouzati. KHARMA - KML/HTML Augmented Reality Mobile Architecture. [Online]. Available: `https://research.cc.gatech.edu/kharma/content/kharma-framework`, [Accessed January 2012].

[20] Gerard J. Kim, Youngsun Kim, and Christine Perey. A Proposal for Reference AR System Architecture Based on Standardized Content Model. International AR Standards Meeting, June 2011.

[21] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst*, 5(3):1–22, 2009.

[22] World Wide Web Consortium (W3C). Linked Open Data W3C SWEO Community Project. [Online]. Available: `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`, [Accessed February 2012].

[23] The EDUCAUSE Learning Initiative. 7 things you should know about Augmented Reality. [Online]. Available: `http://www.educause.edu/ir/library/pdf/ELI7007.pdf`, [Accessed August 2011]., 2005.

[24] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum. In *Proceedings of the SPIE Conference on Telemanipulator and Telepresence Technologies*, volume 2351, pages 282–292, 1994.

[25] STL Partners Telco 2.0, Perey Research, and Consulting. Augmented Reality: Is there a valuable role for telcos? 2011. [Online]. Available: `http://www.perey.com/Augmented_Reality_Analyst_Note_Jan_31_2011.pdf`, [Accessed September 2011].

[26] Pucky2012. Wikitude World Browser @Salzburg Old Town. 2011. [Picture on the Web]. Available: `http://commons.wikimedia.org/wiki/File:Wikitude_World_Browser_@Salzburg_Old_Town.jpg`, [Accessed October 2012].

[27] Feng Zhou, Duh Henry Been-Lirn, and Billinghurst Mark. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '08, pages 193–202, 2008.

[28] Tobias Sielhorst, Marco Feuerstein, and Nassir Navab. Advanced Medical Displays: A Literature Review of Augmented Reality. *J. Display Technol.*, 4(4):451–467, Dec 2008.

[29] Ronald T. Azuma. A Survey of Augmented Reality. *Presence*, 6:355–385, 1997.

[30] George Percivall. Increasing market opportunities for Augmented Reality through collaborative development of open standards. Position Paper - for the International AR Standards Meeting, 2011.

[31] Tim Berners-Lee. Linked data, 2006. [Online]. Available: `http://www.w3.org/DesignIssues/LinkedData.html`, [Accessed April 2012].

[32] Richard Cyganiak and Anja Jentzsch. Linking Open Data cloud diagram, 2011. [Picture on the Web]. Available: `http://richard.cyganiak.de/2007/10/lod/lod-datasets_2011-09-19_colored.png`, [Accessed May 2012].

[33] Richard Cyganiak. The Linking Open Data cloud diagram. [Online]. Available: `http://richard.cyganiak.de/2007/10/lod/`, [Accessed October 2011].

[34] Christian Becker and Christian Bizer. DBpedia Mobile-A Location-Aware Semantic Web Client. In *Proceedings of the Semantic Web Challenge*, ISWC '08, 2008.

[35] GeoNames. About GeoNames. [Online]. Available: `http://www.geonames.org`, [Accessed August 2011].

[36] LinkedGeoData. LinkedGeoData.org - The Linked Geo Data Knowledge Base. [Online]. Available: `http://linkedgeodata.org/About`, [Accessed August 2011].

[37] Sören Auer, Jens Lehmann, and Sebastian Hellmann. Linkedgeodata: Adding a spatial dimension to the web of data. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 731–746. Springer-Verlag, 2009.

[38] OpenStreetMap Wiki. Beginners' guide. [Online]. Available: `http://wiki.openstreetmap.org/wiki/Beginners%27_guide`, [Accessed September 2011].

[39] Sören Auer, Chris Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference*, ISWC '07, pages 722–735. Springer, 2007.

[40] Chris Bizer. DBpedia 3.7 released including 15 localized Editions. 2011. [Online]. Available: `http://blog.dbpedia.org/2011/09/11/dbpedia-37-released-including-15-localized-editions/`, [Accessed September 2011].

[41] Patrick Hayes. RDF Semantics (W3C Recommendation 10 February 2004). World Wide Web Consortium, 2004.

[42] W3C. Rdf vocabulary description language 1.0 rdf schema. [Online]. Available: `http://www.w3.org/TR/rdf-schema/`, [Accessed March 2012].

[43] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. In *International Semantic Web Conference*, pages 30–43, 2006.

[44] Jeffrey T. Pollock. *The Semantic Web for Dummies.* Wiley Publishing, Inc., 1st edition, 2009.

[45] James Q. Jacobs. Geodesy Page - Definitions. 2006. [Online]. Available: `http://www.jqjacobs.net/astro/geodesy.html`, [Accessed October 2011].

[46] Djexplo. Illustration of geographic latitude and longitude of the earth. 2011. [Picture on the Web]. Available `http://commons.wikimedia.org/wiki/File:Latitude_and_Longitude_of_the_Earth.svg`, [Accessed October 2012].

[47] IBM. Geodtische Lngen- und Breitengrade. [Online]. Available: `http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/opt/csbgeo06.htm`, [Accessed October 2011].

[48] Mohinder S. Grewal, Lawrence R. Weill, and Angus P. Andrews. *Global Positioning Systems, Inertial Navigation, and Integration.* Wiley-Interscience, 2007.

[49] Ahmed El-Rabbany. *Introduction to Gps: The Global Positioning System.* Artech House, second edition, 2002.

[50] Department of Defense (DoD) and the Department of Transportation (DoT). NAVSTAR GPS Operations. [Online]. Available: `http://tycho.usno.navy.mil/gpsinfo.html`, [Accessed November 2011].

[51] Enemy. GPS signal modulation scheme. 2008. [Picture on the Web]. Available: `http://commons.wikimedia.org/wiki/File:GPS_signal_modulation_scheme2.svg`, [Accessed October 2012].

[52] Federal Aviation Administration. Navigation programs - global positioning system - control segment. [Online]. Available: `http://www.faa.gov/`, [Accessed October 2012].

[53] Corvallis Microtechnology. Introduction to the Global Positioning System for GIS and TRAVERSE. [Online]. Available: `http://cmtinc.com/gpsbook/`, [Accessed January 2012].

[54] Android Developers. App Framework. [Online]. Available: `http://developer.android.com/about/versions/index.html`, [Accessed August 2011].

[55] Alvaro Fuentes Vasquez (Kronox). Diagram system architecture android. 2009. [Picture on the Web]. Available: `http://commons.wikimedia.org/wiki/File:Diagram_android.png`, [Accessed October 2012].

[56] Satya Komatineni, Dave MacLean, and Sayed Y. Hashimi. *Pro Android 3.* Apress, 1st edition, 2011.

[57] Reto Meier. *Professional Android 2 Application Development.* Wiley Publishing, Inc., 2nd edition, 2010.

[58] Android Developers. API Guides. [Online]. Available: `http://developer.android.com/guide/`, [Accessed August 2011].

[59] Donn Felker and Joshua Dobbs. *AndroidTM Application Development For Dummies.* Wiley Publishing, Inc., 1st edition, 2011.

[60] Android Developers. References. [Online]. Available: `http://developer.android.com/reference/`, [Accessed August 2011].

[61] Ulrich Neumann and Suya You. Natural Feature Tracking for Augmented Reality. *IEEE Transactions on Multimedia*, 1(1):53–64, 1999.

[62] Amir H. Behzadan and Vineet R. Kamat. Visualization of construction graphics in outdoor augmented reality. In *Proceedings of the 37th conference on Winter simulation*, WSC '05, pages 1914–1920. Winter Simulation Conference, 2005.

[63] Bonnie Danette Allen, Gary Bishop, and Greg Welch. Course 11 – Tracking: Beyond 15 Minutes of Thought. In *SIGGRAPH 2001 Courses*, 2001.

[64] Tobias Langlotz, Claus Degendorfer, Alessandro Mulloni, Gerhard Schall, Gerhard Reitmayr, and Dieter Schmalstieg. Robust detection and tracking of annotations for outdoor augmented reality browsing. *Computers & Graphics*, 35(4):831 – 840, 2011.

[65] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. Real-time detection and tracking for augmented reality on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 16:355–368, 2010.

[66] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *Proc. Eigth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'09)*, Orlando, October 2009.

[67] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle Adjustment - A Modern Synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ICCV '99, pages 298–372. Springer-Verlag, 2000.

[68] Wagner Daniel, Mulloni Alessandro, Langlotz Tobias, and Schmalstieg Dieter. Real-time panoramic mapping and tracking on mobile phones. In *2010 IEEE Virtual Reality Conference (VR)*, pages 211–218, 2010.

[69] Markus Kähäri and David J. Murphy. MARA Sensor Based Augmented Reality System for Mobile Imaging Device. In *Proceedings of the 5th Annual IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '06, pages 180–180, 2006.

[70] Takacs Gabriel, Chandrasekhar Vijay, Gelfand Natasha, Xiong Yingen, Chen Wei-Chao, Bismpigiannis Thanos, Grzeszczuk Radek, Pulli Kari, and Girod Bernd. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, MIR '08, pages 427–434, 2008.

[71] Tobias Langlotz, Daniel Wagner, Alessandro Mulloni, and Dieter Schmalstieg. Online creation of panoramic augmented reality annotations on mobile phones. *Pervasive Computing, IEEE*, 11(2):56 –63, 2012 (submitted 2010).

[72] David Marimon, Tomasz Adamek, Arturo Bonnin, and Tomasz Trzcinski. Enhancing global positioning by image recognition. Technical report, Telefonica Research and Development - Barcelona, Spain, 2011.

[73] Greg Welch and Gary Bishop. An introduction to the kalman filter, 1995.

[74] Rémi Paucher and Matthew Turk. Location-based Augmented Reality on mobile phones. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 9 – 16, 2010.

[75] Vinny Reynolds, Michael Hausenblas, Axel Polleres, Manfred Hauswirth, and Vinod Hegde. Exploiting Linked Open Data for Mobile Augmented Reality. In *W3C Workshop: Augmented Reality on the Web*, Barcelona, 2010.

[76] Martin Lechner, CTO, and Wikitude-GmbH. ARML Specification for Wikitude. [Online]. Available: `http://www.openarml.org/wikitude4.html`, [Accessed October 2011].

[77] Wikitude GmbH. Wikitude Webservice Documentation. [Online]. Available: `http://www.wikitude.me/w4/wme/assets/WikitudeWebservice.pdf`, [Accessed January 2012]., 2011.

[78] Layar. Layar Developer Documentation. [Online]. Available: `http://www.layar.com/documentation/browser/`, [Accessed January 2012].

[79] Sekai Camera Support Center. Sekai camera openair api for publishers. [Online]. Available: `http://support.sekaicamera.com/en/archives/category/business`, [Accessed March 2012].

[80] Pedro de-las Heras-Quiros, Raul Roman, Roberto Calvo, Jose Gato-Luis, and Juan F. Gato-Luis. Mobile augmented reality browsers should allow labeling objects. In *W3C Workshop: Augmented Reality on the Web*, Barcelona, 2010.

[81] Morfeo Competence Center. Morfeo LibreGeoSocial. [Online]. Available: `http://libregeosocial.morfeo-project.org/`, [Accessed March 2012].

[82] Stephan Karpischek, Claudio Marforio, Mike Godenzi, Florian Michahelles, and Stephan Heuel. Mobile augmented reality to identify mountains. In *Adjunct Proceedings of the 3rd European Conference on Ambient Intelligence*, AmI '09, 2009.

[83] Salzburg Research. Peak.AR - FAQ. [Online]. Available: `http://peakar.salzburgresearch.at/faq/`, [Accessed January 2012].

[84] Lyndon Nixon, Jens Grubert, and Gerhard Reitmayr. SmartReality: Augmented Reality + Services + Semantics. In *Proceedings of the 2nd International Augmented Reality Standards Workshop*, Barcelona, 2011.

[85] Thomas Visser. A survey of XML languages for augmented reality content. In *Proc. of AR Standardization Forum*, Barcelona, 2011.

[86] Ssolbergj. Compass Rose. 2010. [Picture on the Web]. Available: `http://commons.wikimedia.org/w/index.php?title=File:Blue_compass_rose.svg`, [Accessed October 2012].

[87] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5:199–220, June 1993.

[88] Chris Van Aart, Bob Wielinga, and Willem RobertVan Hage. Mobile cultural heritage guide: location-aware semantic search. In *Proceedings of the 17th international conference on Knowledge engineering and management by the masses*, EKAW'10, pages 257–271, 2010.

[89] Ontotext AD. FactForge Inference. [Online]. Available: `http://www.ontotext.com/factforge/inference`, [Accessed July 2012].

[90] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk - A Link Discovery Framework for the Web of Data. In *Proc. of LDOW '09*, 2009.

[91] Anja Jentzsch. Link Specification Language. 2012. [Online]. Available: `http://www.assembla.com/spaces/silk/wiki/Link_Specification_Language`, [Accessed July 2012].

[92] Peter Ward. *Digital Video Camerawork*. Focal Press, 2000.

[93] Thaddeus Vincenty. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, 22(176):88–93, 1975.

[94] Chris Veness. Vincenty formula for distance between two Latitude Longitude points. [Online]. Available: `http://www.movable-type.co.uk/scripts/latlong-vincenty.html`, [Accessed September 2011].

[95] Jim Riesterer, Scott Hughes, Dan Narsavage, and Diana Boyack. Introduction to Topographic Maps - Get A Bearing. [Online]. Available: `http://geology.isu.edu/geostac/Field_Exercise/topomaps/bearing.htm`, [Accessed November 2011].

[96] Bruce Hoff and Ronald Azuma. Autocalibration of an electronic compass in an outdoor augmented reality system. In *Proceedings of the International Symposium on Augmented Reality*, ISAR 00, pages 159–164, Munich, 2000.

[97] Thom Nichols. Smoothing Sensor Data with a Low-Pass Filter. 2011. [Online]. Available: `http://blog.thomnichols.org/2011/08/smoothing-sensor-data-with-a-low-pass-filter`, [Accessed August 2011].

[98] Claus Stadler, Jens Lehmann, Konrad Hffner, and Sören Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web Journal*, 2011.

[99] LinkedGeoData. Virtuoso Functions Guide - st_intersects. [Online]. Available: `http://docs.openlinksw.com/virtuoso/fn_st_intersects.html`, [Accessed July 2011].

[100] Alias-i. Lingpipe 4.1.0. 2008. [Online]. Available: `http://alias-i.com/lingpipe/`, [Accessed September 2012].

[101] Alias-i. Class JaroWinklerDistance (LingPipe API). 2011. [Online]. Available: http://alias-i.com/lingpipe/docs/api/com/aliasi/spell/JaroWinklerDistance.html, [Accessed September 2012].

[102] Sterling Udell. *Beginning Google Maps Mashups with Mapplets, KML, and GeoRSS: From Novice to Professional*. Apress, 1st edition, 2008.

[103] GSM Arena. Samsung I9003 Galaxy SL. [Online]. Available: http://www.gsmarena.com/samsung_i9003_galaxy_sl-3761.php, [Accessed August 2012].

[104] Flipkart.com. HTC Wildfire S. [Online]. Available: http://www.flipkart.com/htc-wildfire-s-mobile-phone/p/itmcz2pz4zq7hmzq, [Accessed August 2012].