universität
wien

MASTERARBEIT

Titel der Masterarbeit

A Framework for Integrating RDF Data in Mobile Augmented Reality Applications

Verfasserin

Evelyn Hinterramskogler, BSc

angestrebter akademischer Grad

Diplom-Ingenieurin (Dipl.-Ing.)

Wien, 2013

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

Abstract

Augmented Reality (AR) as well as Linked Data represent new research fields that gain major importance. In this thesis, we combine the two approaches of mobile AR, especially location-based AR services, and Linked Data. The use of open data on the Web allows the development of innovative applications and mobile AR opens new possibilities for the representation of geographical information compared to 2D maps. We design an AR framework that bases on open standards and visualizes RDF-based data in an AR interface. The framework displays content such as information about buildings or other resources in the camera's current view area. Basically, it allows the integration of any resource that contains location information and that is queried from a Linked Open Data (LOD) source which provides a SPARQL endpoint. In order to represent the resources in the current view area of the camera, a general approach providing the functionality for retrieving RDF data from LOD sources through SPARQL queries is developed. An important part of the approach represents the mathematical model for transforming the geographic coordinates of the resources to actual screen coordinates that can be displayed in a camera interface. Since our approach bases on location-based technologies, we realize the concepts of 3D viewing that imply the usage of location and orientation sensors of mobile devices. Thus, we present a mathematical model for calculating screen coordinates that rely on the position and orientation of the camera as well as the position of the resources. Although, there already exist some AR applications, none of them uses RDF data from LOD sources for visualizing resources. Primarily, the use of an AR approach improves usability and prevents problems that occur using a 2D map. We will highlight the advantages of these new research fields and present an approach that allows to implement an AR framework for visualizing structured RDF data from LOD sources.

Zusammenfassung

Augmented Reality (Erweiterte Realität) sowie Linked Data stellen neue Forschungsfelder dar, die immer mehr an Bedeutung gewinnen. Der Schwerpunkt dieser Arbeit liegt in der Verbindung von Augmented Reality (AR) Diensten mit Linked Data Konzepten. Dabei liegt der Fokus auf standortbezogenen AR Diensten. Besonders die Verwendung von frei verfügbaren Daten im Web ermöglicht die Entwicklung von innovativen Applikationen. Zudem eröffnet AR neue Möglichkeiten für die Darstellung von geografischen Daten im Vergleich zu der Visualisierung von Daten in einer 2D Karte. Im Rahmen dieser Arbeit wird ein AR Framework entwickelt, welches auf offenen Standards basiert und auf RDF basierende Daten in einem AR Interface visualisiert. Das Framework stellt Informationen über Gebäude oder andere Ressourcen in der Kameraansicht eines mobilen Gerätes dar. Grundsätzlich ist es möglich, jede beliebige Ressource, die über Positionsdaten verfügt und deren Linked Open Data (LOD) Dienst einen SPARQL Zugangspunkt bereitstellt, zu integrieren und darzustellen. Um die Ressourcen in der Kameraansicht darstellen zu können, haben wir einen Ansatz entwickelt, der es ermöglicht, RDF Daten von LOD Diensten mittels SPARQL abzurufen. Einen wichtigen Teil dieser Arbeit bildet das mathematische Modell, welches die Transformation von geographischen Koordinaten zu Bildschirmkoordinaten realisiert. Um eine Transformation der Koordinaten durchführen zu können, werden zusätzlich Daten von Positions- sowie Orientierungssensoren von mobilen Geräten ausgelesen und integriert. Folglich basiert das mathematische Modell auf der Position und Orientierung des mobilen Gerätes sowie der Position der Ressourcen. Die Integration von auf RDF Daten basierenden Ressourcen stellt einen neuen Ansatz für AR Applikationen dar. Die Verwendung dieses Ansatzes soll vor allem die Benutzerfreundlichkeit verbessern sowie Probleme, welche bei der Verwendung von herkömmlichen 2D Karten auftreten, verhindern. Wir werden den Fokus auf die Vorteile, welche sich durch die Integration von diesen neuen Forschungsfeldern ergeben, legen und einen Ansatz entwickeln, der es ermöglicht, strukturierte RDF Daten von LOD Diensten in einem AR Interface darzustellen.

Acknowledgements

First, I would like to thank my supervisor Prof. Dr. Wolfgang Klas and my secondary advisors Dr. Stefan Zander and Dipl.-Ing. Chris Chiu for all their support, guidance and ideas. A special thank goes out to my fellow students, especially my friend Silvia, who always encouraged me during this thesis and the years of study. Among all, I would like to thank my parents for motivating me and giving me the possibility to attend university. Last but not least, my gratitude goes to my fiancé Boris for showing me understanding and for always supporting and motivating me.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Due to the continuous technical sophistication of mobile devices, new research fields like Augmented Reality (AR) [1] gained increasing interest. There already exist a few applications that visualize augmented content such as Wikitude [2], Layar [3], or Mixare [4]. In our work we combine AR and Linked Data [5] which represents another important new research area. The use of open data on the Web allows the development of new and innovative applications. The main objective of this master thesis is to combine these two research areas and define an approach that allows to display information about resources retrieved from LOD sets in the camera view of a mobile device right where the real objects are located. Therefore, nearby locations have to be localized and information about these resources has to be collected. Our approach bases on the integration of data offered by Linked Data sources such as the DBpedia project. Basically, our approach allows to integrate any LOD source that host data with location information. Thus, we do not only limit the approach to points of interest (POIs) but also to any resources that are related to a location information. All the required information such as the geographical positions will be queried from these sources. The usefulness of our approach is demonstrated by means of an AR application prototype.

1.1 Motivation

The main focus of this thesis lies on the visualization of Linked Data content in an AR interface on mobile devices. An application that visualizes Linked Data is DBpedia Mobile [6] which renders a 2D map displaying the nearby POIs and supplying background information about the resources. Moreover, it is possible to navigate along links to related datasets on the Semantic Web. Thus, the user can navigate into related datasets and explore background information about them. Consequently, the approach of DBpedia Mobile enables a data federation. We apply some basics of DBpedia Mobile and enhance the approach by visualizing the information in an AR approach. An AR approach is used primarily for preventing the problems that implicate the visualization of resources on a 2D map such as the manual zooming into maps. For instance, an AR

application enables a more realistic impression of the environment to the user. Moreover, the approach shall bring along an improvement of usability; generally, AR is a technology that combines some advantages compared to the illustration of resources on a 2D map.

Our approach is based on the integration of Linked Data which is built upon an open data format. As opposed to this, popular AR applications such as Wikitude retrieve information from closed databases in proprietary format. Thus, this data is owned by an organization or an individual. Because of that, we will base our approach on the visualization of data using an open and standardized format. Therefore, a very promising way of displaying location-based data in AR interfaces is to request the data from LOD sources such as the DBpedia project [7]. The project is part of the Linking Open Data Community Project [8] which extends the Web by publishing open data sets as RDF and combining such data sets using RDF links. RDF is a Semantic Web language that not only defines metadata but also the relationship among them. It is multifunctional and not bounded to the definition of Web resources in some special domain. Some advantages of using the RDF data model in the context of Linked Data are summarized in [9]. An advantage of using the RDF data model is that the user can look up URIs in an RDF graph for retrieving additional information about the resources. Moreover, the data model allows to set RDF links between resources from different LOD sources. As described in [10], the RDF model can be efficiently stored and processed because it consists of triples. Additionally, Linked Open Data sources provide datasets with dereferenceable URIs that allow to retrieve a description of the resources from the Web. Moreover, the URIs are already linked to other URIs from different LOD sources.

In order to combine the two approaches of AR and Linked Data, we develop a mathematical model that allows to represent resources from LOD sources in an AR interface. In doing so, the focus is on the integration of Linked Data and the exploitation of the benefits of open data sources. Equally, the advantages of visualizing data in an AR view compared to a 2D map is utilized usefully. In the following Section 1.1.1, we focus on the advantages of displaying data in an AR interface. Basically, the mathematical model calculates the position of the resources that are in the device's current view area. We decided to develop a mathematical model that is based on a location-based algorithm. Therefore, we consider the position and orientation of the mobile device as well as the position of the LOD resources. In order to display the resources correctly in the camera's field of view cf. Definition 4.2, we realize a perspective projection. This type of projection allows to create an illusion of depth and considers the distance between the camera and the resources.

Generally, we develop an AR framework that realizes the mathematical model for calculating the position of the resources. Additionally, we demonstrate the usefulness

of our approach by implementing an AR application prototype that extends the AR framework for visualizing resources from the DBpedia project.

1.1.1 Visualizing Data in Augmented Reality compared to 2D Maps

The integration of Linked Data in an AR application implies several advantages compared to the realization on the basis of 2D maps. As already mentioned, there exists an application that visualizes DBpedia content on a 2D map, called DBpedia Mobile [6]. The application is described more detailed in Chapter 3. Basically, this application renders a 2D map that displays information about nearby POIs. The authors in [11] describe the problems that occur by a visualization of POIs on a 2D map. One problem is that most of the maps are aligned with the North direction. That is why, users have to turn the map around every time the direction changes. An AR interface will solve this problem because it considers the orientation of a mobile device. Another problem states the changing of the map scale. It is difficult to manually zoom into a map and receive the desired size and section of the map. For changing the map scale the user has to interact with the device by, e.g., spanning an rectangle or zooming with "+" and "-" signs. Generally, it is very difficult to provide a simple and efficient user interface when displaying a map on a small screen. In contrast, an AR interface allows to hold the device in the desired direction and displays all resources that are in the camera's field of view. The user does not need to zoom manually.

Generally, an AR application provides better orientation facilities because the user is integrated into the real world. Moreover, the user is guided by optical markers. Most people notice special buildings or other objects in the environment. These objects help people to get a better orientation. It is more difficult to use street names for orientation than using pictures of special objects. Therefore, some 2D maps offer landmark photographs that should serve as orientation guide. Indeed this approach does not consider that the landmarks look different for day and night which make the identification of them difficult again.

1.2 Research Questions and Research Objectives

In order to prove the usefulness of our approach, we develop a prototype that visualizes augmented content such as information about buildings in the mobile device's camera view. Resources with geographic coordinates that are located in the environment will be retrieved dynamically and proactively from Linked Open Data sources and represented in the AR user interface. Information about the resources displayed in the AR interface is retrieved from LOD sets such as DBpedia, LinkedGeoData, or GeoNames. The framework allows to integrate resources from different LOD sources and it enables a data federation.

In context of the thesis we elaborate on the following research questions:

- How can resources with geographic coordinates that are in the current view area be detected and represented correctly in an Augmented Reality user interface?
- How can the correct RDF data of the resources in the current view area be requested from Linked Open Data sources and visualized accurately in an Augmented Reality user interface?
- How can data that is received from the sensors of a mobile device such as the camera, the Global Positioning System (GPS), or the accelerometer be used for a computational model of the physical surrounding?

According to the research questions the following research objectives can be deduced:

- Develop a mathematical model for calculating the position of resources that are in the device‘s camera‘s current view area depending on the position of the mobile device, the direction in which the device is held, and the distance of the resources.
- Define SPARQL queries to access information of the resources that should be represented if a resource is in the current view area of the mobile device from LOD sources.
- Develop an Augmented Reality framework that accesses the sensors of a mobile device such as the camera, GPS, or the accelerometer and figure out how these data contribute to the mathematical model for representing the resources.

In order to validate the research questions, we develop an AR prototype that visualizes resources in the camera's field of view.

1.3 Outline of this Work

The contribution of this thesis is to develop a mathematical model that calculates the screen coordinates of the resources located in the surrounding area and represents them in the user interface. Above all, the approach should work with arbitrary data sources that host data with location information. Building such an AR application includes various technologies. First the application has to access the sensors of the mobile device for retrieving GPS information as well as the orientation data of the device. Additionally, the camera interface has to be implemented. Once the sensor data is determined, information that is retrieved from the sensors has to be contributed to the mathematical model for the detection of the resources. Additionally, information about the resources has to be queried from LOD databases. Therefore, SPARQL queries for retrieving basic information such as geographic coordinates, labels, or short summaries have to be defined and executed. Furthermore, the expected results of this master thesis include the evaluation of the possibilities and requirements for building such an AR application. The possibilities of using data from LOD sources in an AR application will

be researched and the advantages of using RDF data will be analyzed. The sensors of a mobile device will be reviewed. Moreover, the AR approach should overcome the problems that occur by visualizing resources in a 2D map and provide better usability and guidance. Generally, data about the resources that are located in a predefined area has to be buffered while the user is executing the application. Thereby, even if the connection is interrupted, data can be displayed and the internet connection must not be available constantly.

This thesis is structured into 8 chapters. The first chapter describes the idea and the motivation of the thesis. The following chapter covers the theoretical background. The background will provide basic information about the main parts of this master thesis such as AR, mobile platforms or Linked Open Data.

Chapter 3 summarizes existing projects that apply AR as well as Linked Open Data approaches. It gives an overview of state of the art methods and allows to bring out the differences and advantages of the approaches used in this master thesis compared to existing projects.

After providing a basic understanding of the concepts, Chapter 4 explains the approach that we propose in this thesis. Thereby, the mathematical model for calculating the position of the resources is explained. Moreover, the approach of visualizing off-screen objects will be discussed. Another important point mentioned in this chapter is the approach for accessing LOD databases.

Chapter 5 aims to provide an overview of the system architecture of the framework that is implemented in this master thesis for verifying the presented approach.

In Chapter 6, we describe the implementation of the framework for data from the LOD source DBpedia. Thereby, we focus on the integration of RDF data from the DBpedia project.

In Chapter 7, the implemented application will be evaluated and problems that occurred during the creation of the master thesis will be discussed. We focus on the evaluation of the times for calculating the screen coordinates and representing the resources in the camera preview as well as the accuracy of location based information from LOD sources.

Chapter 8 consists of conclusions on the work as well as possible improvements for future work.

Chapter 2

Background

This chapter covers basic information about the technologies used in the context of this work. First we give an overview about Augmented Reality in order to define its purpose and to describe the possibilities of this research area. Thereby, our focus is on the different types of AR systems as well as the methods for identifying resources and displaying them in the camera's field of view. Afterwards, we give a summary of current mobile software platforms and compare the well-known platforms, e.g., according to their license type or their operating system. Since, we decided to develop a location based approach, we explain the sensors offered by a mobile device, too. As we use LOD for retrieving all necessary information about the resources, we give an overview about the basic concept of LOD as well as the Geospatial Semantic Web. Moreover, the RDF data model as well as SPARQL are explained in this section. Additionally, we give an overview of specific LOD datasets. Finally, we explain the basic concept of 3D viewing. In order to develop an AR approach, it is important to understand the projection of the coordinates of a 3D object onto a 2D screen.

2.1 Augmented Reality

Augmented Reality (AR) is defined as a real-time direct or indirect view of a physical real-world environment that has been augmented by adding virtual computer-generated information to it [12]. The author in [1] describes AR as a variation of Virtual Environments (VE). The differences between VE technologies and AR are described by the author in the following way: Virtual Environments are computer-generated environments that immerse a user inside a synthetic environment, which means, that the user can only see the artificial environment and cannot see the real world around him. As opposed to this, in AR the user sees the real world combined with virtual objects. Thus, users can see real elements extended with information. The author defines AR with the following characteristics. AR is a real time interaction and registered in 3D as well as a combination of real and virtual objects.

In 1994, Milgram [13] defined a continuum that spans between the real environment and the virtual environment. Figure 2.1 illustrates the Mixed Reality [13] defined by Milgram that ranges from the real environment to the virtual environment. The figure indicates that Augmented Reality is closer to the real world and Augmented Virtuality is closer to the artificial environment.

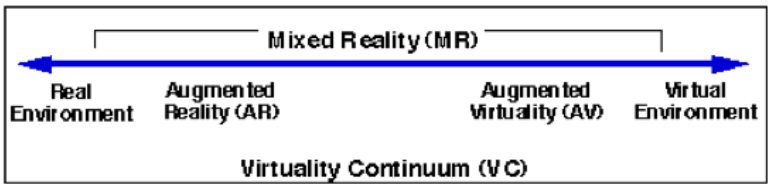

FIGURE 2.1: Simplified representation of a "virtual continuum" [13]

AR brings virtual information to the user's immediate direct surroundings as well as indirect environment such as live-video stream. AR aims to improve the user's perception of the real world and allows an interaction with it. As opposed to this, the Virtual Environment immerse users in a synthetic world.

In[14] the author explains the advantages of an AR System. Because AR is a combination of real world and virtual reality it allows to get a better sense of reality. The environment is not only virtual and therefore, it enables a more realistic impression. Moreover, AR provides a better interaction with the environment than Virtual Reality. AR systems allow to participate in the real world.

There exist different types of AR systems. In order to get an overview about them, we refer to the categories of AR systems defined in [15]. The authors categorize AR systems the following way:

- fixed indoor systems
- fixed outdoor systems
- mobile indoor systems
- mobile outdoor systems
- mobile indoor and outdoor systems

Mobile systems allow the user to move around. In contrast, fixed systems are positioned at a particular point and cannot be moved. In order to illustrate our concepts we develop a mobile outdoor system. Moreover, the systems can be distinguished between AR systems that function either indoors or outdoors, or systems that work indoors as well as outdoors. Talking of mobile AR systems the focus will be on mobile phones' applications. Mobile AR systems especially use tracking systems such as GPS mostly in combination with other techniques like accelerometers, gyroscopes or other rotation sensors.

2.1.1 Augmented Reality Techniques

The development of an AR application implies various different technologies such as image recognition, geopositioning, or virtual reality. This section provides an overview of the present techniques for realizing an AR system. The authors in [16] describe techniques for the identification of a scene that is in the current camera view of a mobile device. We define a scene as a part of the user's environment that has to be recognized and extended with augmented content by the AR system. Moreover the authors explain which hardware and software components are necessary for deploying an AR system. Generally, AR systems can be distinguished between marker-based and non marker-based AR systems. Marker-based systems are using visual tags that are integrated in the real world. The AR systems are programmed to find these tags and identify them. Thus, the AR system uses the markers for identifying a scene. In contrast, there are three different possibilities for identifying a scene that non marker-based systems can use such as image recognition, geopositioning, and a hybrid form that combines image recognition and geopositioning. Image recognition identifies scenes by comparing images or videos. In contrast, geopositioning uses the position and orientation of the device to calculate the position of the points.

Anyhow, optical recognition algorithms have some efforts. They require an extensive database with images as well as enormous calculations. Additionally, image recognition does not apply efficiently to certain situations where objects are covered by other objects. Moreover, images taken at daytime differ from images taken at night which makes the comparison of the images more difficult. That is why in this master thesis a mathematical model is used that reflects the physical real-world environment for detecting the resources' positions by means of geopositioning technologies.

2.2 Mobile Software Platforms

Mobile information systems are information systems in which end-user terminals access information resources and services. The end-user terminals are movable and their functionality is independent of the current location. Typically, they are equipped with a wireless connection. Mobile software platforms are the operating systems that control, e.g., a mobile device. For instance, smartphones, tablet computers, or personal digital assistants run a mobile operating system. In this section, we give an overview of current existing mobile operating systems and compare them. Section 2.2.1, first, summarizes the systems and Table 2.1 compares the basic facts with respect to the license type, the availability of a documentation, or the programming language.

2.2.1 Overview of Different Mobile Platforms

First, we want to give an overview of existing mobile software platforms and compare them in Table 2.1. Above all, we compare the different operating systems and programming languages used by the mobile software platforms. Moreover, we focus on the license type and check if a comprehensive documentation is available.

iPhone OS / iOS[1] The mobile operating system is derived from Mac OS X and consequently, it is a Unix-like operating system. It is programmed in C, C++, Objective-C and it was the first smartphone that offers multi-touch features. Apple provides an iOS SDK that allows in combination with Xcode tools the development of mobile applications.

Android OS[2] Google's Android OS is an open and free software stack that includes an operating system, middleware, and also key applications. It provides an official SDK for downloading as well as a comprehensive documentation for developers.

Symbian[3] The Symbian platform represents an operating system for smartphones and PDAs. Symbian uses standard C++ with QT[4] as SDK. Symbian uses the Nokia Store[5] for distributing applications. The Symbian code is disposable and currently maintained by Accenture[6] on behalf of Nokia.

Windows Phone[7] The mobile operating system is the successor to the Windows Mobile platform. Windows Phone features a new user interface named "Metro Design Language"[8] and it uses multi-touch technology. The operating system is closed source with a proprietary license.

BlackBerry OS[9] The mobile operating system is proprietary and was developed by Research In Motion[10] for its BlackBerry smartphones. Originally, it was designed for business but it was improved and offers various downloadable applications with full multimedia support.

Bada[11] The operating system for mobile devices is developed by Samsung Electronics. The Bada SDK is available for download and a documentation for developers is provided. Moreover, Samsung created an application store only for the Bada platform.

[1] iPhone OS: `http://www.apple.com/ios/`
[2] Android: `www.android.com`
[3] Symbian: `http://symbian.nokia.com/`
[4] Qt: `http://qt.nokia.com/`
[5] Nokia Store: `http://store.ovi.com/`
[6] Accenture: http://www.accenture.com/us-en/pages/index.aspx
[7] Windows Phone: `http://www.microsoft.com/windowsphone/`
[8] Metro: `http://www.microsoft.com/design/toolbox/tutorials/windows-phone-7/metro/`
[9] BlackBerry: `www.blackberry.com`
[10] Research In Motion: `http://www.rim.com/`
[11] Bada: `http://www.bada.com/`

webOS [12] HP webOS bases on a Linux kernel and it was introduced in 2009 by Palm and later acquired by Hewlett-Packard[13]. Moreover, it offers a SDK for developers which includes access to compilers, coding libraries, scripts and other documents. It is developed in C/C++ and applications can be written in HTML5, JavaScript and CSS, too.

[12] HP webOS: `https://developer.palm.com/`
[13] Hewlett-Packard: `http://www.hp.com/`

Mobile Platform	**Company/ developer**	**Underlying OS**	**Programming language**	**Open source**	**License type**	**Open API/SDK**	**Documentation**
iPhone OS / iOS	Apple	Darwin	C, C++, Objective-C	✗	proprietary	✓	iOS Dev Center[14]
Android OS	Google Inc, Open Handset Alliance	Linux	C, Java, C++	✓	Apache	✓	Android Developers[15]
Symbian	Accenture on behalf of Nokia	Symbian OS	C++	✓	Eclipse Public License (EPL)	✓	Symbian C++ - Documentation[16]
Windows Phone	Microsoft Corporation, Windows CE	Win32	C#	✗	proprietary	✓	Windows Phone Development[17]
BlackBerry OS	Research In Motion	Mobile OS	C, C++	✗	proprietary	✓	Developer Doc.[18]
Bada	Samsung Electronics	Linux	C, Java, C++, Flash	✓	proprietary	✓	bada Documentation[19]
webOS	Hewlett-Packard	Linux	C/C++	✓	open except closed source modules	✓	HP webOS Developer Center[20]

TABLE 2.1: Comparison of mobile platforms

[14] iOS DevCenter: `http://developer.apple.com/devcenter/ios/`
[15] Android Developers: `http://developer.android.com`
[16] Symbian C++ - Documentation: `http://www.developer.nokia.com/Develop/Featured_Technologies/Symbian_C++/Documentation/`
[17] Windows Phone Development: `http://msdn.microsoft.com/en-us/library/ff402535(v=vs.92).aspx`
[18] Documentation for Developers: `http://docs.blackberry.com/en/developers/`
[19] bada Documentation: `http://developer.bada.com`
[20] HP webOS Developer Center: `https://developer.palm.com/`

As already mentioned in Chapter 1.1, we decided to proof our approach by implementing a framework on the Android platform. The Android Operating System is an open source platform that offers a comprehensive documentation. The Android SDK provides the tools and APIs necessary to develop AR applications. Moreover, Android devices are equipped with all sensors needed for implementing an AR application such as an accelerometer or a magnetic field sensor and it allows to fetch the GPS signal. The easy access to advanced features such as the camera, sensor data etc. enables a productive development.

2.2.2 Sensors integrated in Mobile Devices

Nowadays, mobile phones provide a set of embedded sensors that offer rich opportunities for developers. Some commonly used sensors provided by mobile platforms are the accelerometer, digital compass, gyroscope, or the proximity sensor. Especially location based AR applications rely on the use of several sensors. In order to compute the screen coordinates of the objects, it is necessary to determine the position and orientation of the mobile device. In [17], the authors describe the sensors of mobile phones. They focus on sensors such as an accelerometer that is used to calculate the orientation in which the user is holding the phone. For instance, this sensor can be used to automatically switch the orientation of the display from horizontal to vertical or backwards. Moreover, sensors like light sensors or proximity sensors are used to adjust the user interface. Furthermore, sensors like the compass or the gyroscope provide additional information about the phone's position and its orientation. In the following, some well-known sensors are briefly described. The author A. Allan describes the purpose of some mobile device's sensors such as the accelerometer or the magnetometer in [18].

Accelerometer The accelerometer measures the acceleration that is applied to the mobile device. The acceleration can be determined in one, two, or three orthogonal axes. There are various types for applying the sensor information. For instance, it can be used as a measurement of velocity or as a sensor that detects vibration. In addition, it is used to determine the inclination or the orientation of a mobile device. Most of the time, smartphones use the accelerometer to switch between a horizontal and a vertical screen. Furthermore, the accelerometer can be used for detecting if the user, e.g., is running or standing. The accelerometer is described detailed in [19].

Gravity The sensor calculates the direction and magnitude of gravity. Thus, the sensor is used to determine the orientation and movement of the mobile device. This allows to automatically switch the screen format from portrait to landscape format and vice versa. Moreover, the sensor can be used for the development of games in order to implement a control according to the device's movement.

Gyroscope The gyroscope [20] is a sensor that measures angular velocity and it is mostly used for navigation purposes. It can be used to measure the orientation of a device. Contrary to the accelerometer, the gyroscope measures the orientation directly. The accelerometer measures the linear acceleration of the mobile device.

Light sensor Light sensors are used to adjust the brightness of the screen depending on the incidence of light. As a result, the screen's brightness can increase if the ambient light increases. Moreover, the brightness will be decreased automatically, if the incidence of light is lower. In this way, the viewing is more comfortable and it saves battery life.

Magnetic field sensor This sensor measures the ambient magnetic field in the x, y, and z-axis and returns the values in micro-Tesla. Thus, the magnetic field sensor measures the strength of the magnetic field surrounding the device. The sensor acts as a digital compass that enables to determine the mobile device's "heading" respectively to the magnetic North Pole.

Rotation vector The Rotation vector determines the orientation of a mobile device, too. The orientation is determined by the combination of an angle and an axis. The angle specifies the degrees that it is rotated around the defined axis.

Orientation sensor The Orientation sensor indicates in which direction the mobile phone is pointing. Android offers a sensor that delivers the data as angles in degrees. Thereby, the following 3 values are returned: azimuth, pitch and roll. The azimuth value represents the angle when the phone is rotated around the z-axis. The returned values are in a range of 0 to 360 where 0 points to the North, 90 to the East, 180 to the South, and 270 to the West. The pitch value illustrates the rotation around the x-axis within a range of -180 to 180 and the roll value shows the angle around the y-axis by values within -90 to 90.

Proximity sensor The Proximity sensor, e.g., detects if the user is holding the phone close to the ear for making a phone call and as a consequence, it disables the touchscreen function. Therefore, the infrared wavelengths are used. The closeness of an object is determined by detecting the infrared reflection strength.

2.2.3 Determining User Location

The implementation of a mobile outdoor AR system implies various different technologies. One of it is the determination of the user's location. Therefore, well-known mobile platforms provide GPS for obtaining the position of the user. Moreover, some platforms allow to determine the user location by using cell-towers and Wi-Fi signals. For instance, Android provides GPS as well as Android's Network Location Provider for calculating the position of the user. In the following, we briefly explain these two methods for determining the user location and outline their advantages and disadvantages.

GPS GPS determines the user's location using satellites. Actually, GPS provides the most accurate data compared to other systems such as the determination of the location through Wi-Fi signals. Indeed it needs much battery power. The GPS signal only works limited in dense urban areas or indoors because the signals are too weak.

Wi-Fi positioning The Wi-Fi positioning uses cell towers and Wi-Fi signals for determining the position. In that way, the user's location can be determined without using GPS. This method works indoors as well as outdoors, it requires less battery power and determines the position faster than the GPS signal. Although the Network Location Provider has several advantages, it is not as accurate as the GPS signal and it only works in case of cell-coverage.

In [21] the authors describe the challenges of obtaining accurate user location. For instance, there could exist multiple signals (GPS, Cell-ID, Wi-Fi) providing different user locations. In addition, the movement of the user can cause problems. It would require very much battery power to determine the user's location at each movement.

2.2.4 Android

Generally, Android [22] is a platform that is built for mobile devices. The software stack includes a Linux-based operating system, a middleware, as well as key applications and the applications are written in the Java programming language. All the tools and APIs that are needed to develop applications are provided by the Android SDK. In [23], the authors give a survey of the Android Computing Platform. The operating system is available for commercial and non-commercial use. One very important fact of the Android platform is that it is open source. For instance, third party developers can provide multimedia codecs or the user can install shells on a device. The open source platform enables that the user does not rely on Google. New functionality can be provided by anyone.

2.2.4.1 Architecture

The Android Development Guide [21] provides a general view of the android architecture. Furthermore, in [24] the author summarizes details about the architecture. The architecture consists of the following major layers and each component of the layer is building on the components of the layer below it.

Applications Generally, Android includes a set of core applications that are written in the Java programming language such as an email client, SMS program, calendar, and others.

Application Framework The application framework provides a rich set of APIs for creating applications and enables huge opportunities to the developer. Especially, the

open development platform as well as the fact that developers have free access to the same framework APIs used by the core application and much more allows the creation of various types of innovative applications. Generally within the framework applications can operate and the framework manages these applications.

Libraries Through the application framework developers have access to a set of C/C++ libraries such as a System C library, several Media Libraries or a Surface Manager that enables the access to the display subsystem. The system libraries are largely responsible for tasks such as graphics rendering, database access, or audio playback.

Android Runtime The Android Runtime is based on the Linux kernel and is responsible for running Android applications. Therefore, each application is equipped with its own Dalvik virtual machine and run in its own process.

Linux Kernel The Kernel provides the basic drivers for the hardware components. It is responsible for services like security, memory management, process management, and more.

2.2.4.2 Coordinate System of an Android Device

The SensorEvent API [25] uses the coordinate system illustrated in Figure 2.2. The x-axis is the horizontal line, the y-axis is the vertical line and the z-axis points towards the user. The z-values are negative if they are behind the screen. The x-axis points to the right and the y-axis points up. When the screen orientation changes from vertical to horizontal or backwards the coordinate system does not change.

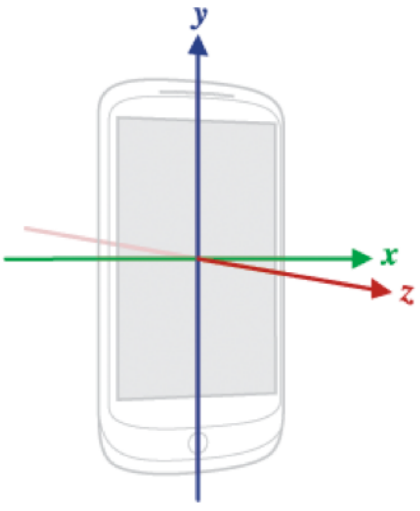

FIGURE 2.2: Coordinate system of an Android device [26]

2.3 Linked Open Data

In recent years, the Web has changed from a global information space that links documents to a space where both documents as well as data are linked. The authors in [27] provide an overview of Linked Data and its concepts. Generally, Linked Data [5] describes a method for publishing and interlinking structured data on the Web. As a result, data from various sources can be connected as well as queried. The main idea of Linked Data is to share structured data on global scale. Linked Data relies on URIs [28] (Uniform Resource Identifiers) as well as HTTP [29] (HyperText Transfer Protocol). Both are two technologies that are essential to the Web. In contrast to URLs (Uniform Resource Locators), that address documents and other entities located on the Web, URIs identify any entity that exists in the world. URIs that use `http://` for identifying entities can be dereferenced over the HTTP protocol. The two technologies are supplemented by RDF which provides a generic, graph-based data model. It links data describing things in the world. The concepts of the RDF data model are described more detailed in Section 2.3.3.

Web of Data Linked Data was adopted by a huge amount of data providers which led to a global data space called the "Web of Data" [30]. The Web of Data consists of a huge amount of RDF statements from various sources. These statements cover all imaginable topics like geographic locations, buildings, films, books, or people. Generally, the Web of Data can be seen as an additional layer because it builds on the general architecture of the Web. As described in [27], the Web of Data can contain any type of data. Moreover, RDF links are used for connecting the entities. The resulting global data graph allows to discover new facts.

Information resources We define an information resource as the authors in [9]: generally, resources are all items of interest and it is distinguished between information resources and non-information resources. Information resources are defined as all resources that can be found on the traditional document Web like documents or images.

Non-information resources In contrast to information resources the authors in [9] describe non-information resources as all objects that exist outside of the Web, so called "real-world objects", such as people or places.

2.3.1 Basic Concept

Tim Berners-Lee introduced the Linked Data principles in his Web architecture note "Linked Data" [5]. They have become known as the best practices for publishing and interlinking structured data on the Web.

Use URIs as names for things. The first principle signifies that not only Web documents but also real objects such as people or places as well as concepts has to be

identified by a URI reference. This means that not only online resources are identified but also any object or concept in the world such as people, places, or a relationship type.

Use HTTP URIs so that people can look up those names. The second principle builds on the fact that HTTP URIs can be dereferenced over the HTTP protocol and a description of an object can be retrieved.

When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL). It is important to support standardized content formats. Therefore, the Resource Description Framework (RDF) is a proper standardized format for publishing structured data on the Web. RDF is explained more detailed in Chapter 2.3.3.

Include links to other URIs, so that they can discover more things. The fourth principle declares that it is important to connect Web documents as well as real world objects using RDF links. For instance, links can be set between a person and a place and the link describes the relationship between them.

2.3.2 Geospatial Semantic Web

The Geospatial Semantic Web [31] represents locations such as POIs, regions, or countries as first-class citizens of the Web that are addressable using URIs. They are represented as Web resources that can represent, e.g., POIs or even areas such as countries. RDF is used for describing the resources. The Geospatial Semantic Web provides geographical relationships among the Web resources. Thus, locations can be related to each other. For instance, administrative hierarchies such as the relationship between a town and its corresponding country can be specified by the relationship type `geoNames:parent-Feature` provided by GeoNames. Another example is the representation of shared borders between two countries by the type `factbook:landboundary`. Moreover, the Geospatial Semantic Web allows to relate Web resources to other types of resources like people, photos or websites. The data can be accessed by dereferencing URIs over the HTTP protocol. The URIs are dereferenced into RDF descriptions and allow client applications to navigate between data sources. Considering how the traditional document Web can be crawled by following hypertext links, the Semantic Web can be similarly crawled by following data links. The Geospatial Semantic Web allows to query for information such as "all museums in Vienna". Moreover, sophisticated query capabilities can be provided.

2.3.3 The RDF Data Model

The Resource Description Framework [32] is a standard model for representing and interchanging information on the Web. In [33], the design goals as well as concepts are listed. RDF is designed to be a simple data model that has formal semantics and

comprehensible inference. Thereby, it uses a URI-based vocabulary that is extensible, too. Basically, RDF is structured as a collection of triples. Each triple consists of a subject, a predicate, and an object. The predicate indicates the relationship between the subject and the object. An RDF graph consists of a set of such triples. Figure 2.3 shows that the subject and object are the nodes in the graph and the predicate, also called property, indicates the relationship between them.

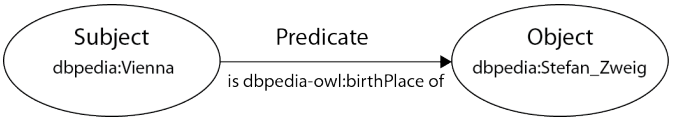

FIGURE 2.3: Example RDF graph

Furthermore, the vocabulary is URI-based which means that a node can be either a URI, a literal or a blank node and a predicate is a URI reference. The URI reference serves as an identification of the nodes or the properties. A blank node is a unique node that has no specific name. Only the object of an RDF graph can be represented as a literal. Literals are a lexical representation of values such as numbers or dates. In addition, datatypes like integers, floats or dates can be represented, too.

2.3.3.1 Serialization Formats

Before RDF data can be published on the Web, it has to be serialized using an RDF syntax. There exist two serialization formats that are standardized by the W3C called RDF/XML and RDFa. The recommended syntax for exchanging RDF information is RDF/XML. Moreover, there are some non-standardized serialization formats. In [30, 34] the authors describe the RDF serialization formats.

RDF/XML [35] This syntax represents an RDF graph as an XML document. Therefore, the subjects, properties, and objects have to be represented in XML terms. Thus, RDF URI references are represented by an XML QName[21] that consists of a URI reference and a short local name. RDF literals are converted to XML elements or XML attributes. The MIME type used within HTTP content negotiation for RDF/XML is `application/rdf+xml` and RDF/XML files have the extension ".rdf" on all platforms.

RDFa [36] RDFa (Resource Description Framework in attributes) is not only a serialization format for RDF, instead it supports the embedding of RDF data into Web documents. RDFa enables to express structured data in any markup language. Generally, RDFa is a specification for attributes. These attributes are used in XHTML to describe the semantics of the displayed information.

[21] QName: `http://www.w3.org/TR/1999/REC-xml-names-19990114`

Notation 3 [37] N3 uses the MIME type `text/n3` and the encoding is always UTF-8. To avoid redundant information, N3 enables to define a URI prefix at the beginning of the document. Thus, entity URIs can be identified by these prefixes. Furthermore, multiple statements about the same subject can be summarized and combined using a semicolon. N-Triples as well as Turtle are subsets of Notation 3.

N-Triples [38] N-Triple statements are reasoned very simple. It is a line-based and plain text format. One statement that represents a triple consisting of subject, predicate and object is expressed in a separate line and is terminated by a dot. Usually, it uses the file extension ".nt" and the Internet media type is `text/plain`. N-Triple is a subset of N3. Thus, libraries that are able to read N3 will read N-Triple, too.

Turtle [39] Turtle enables RDF data to be written in plain text format that is compact. Abbreviations are used for common patterns and datatypes. Generally, Turtle is an extension of N-Triples. In order to express simple triples that consist of subject, predicate, and object it is written as a sequence separated by whitespace and terminated by a dot. The MIME type for Turtle is `text/turtle` and it uses the file extension ".ttl".

2.3.4 The SPARQL Query Language

SPARQL [40, 41] - Simple Protocol And RDF Query Language - is a query language for RDF data. SPARQL is able to query graph patterns. The language is syntactically similar to SQL and it uses pattern matching for querying RDF graphs. SPARQL offers a communication protocol that can be used by clients for querying against SPARQL endpoints[22]. Furthermore, SPARQL supports several XPath operators[23]. For instance, the operators can be used to compare the values of RDF literals. The basic requirements are that the resources are identified by IRIs[24] and that an XML Schema datatype is used for writing literals.

2.3.4.1 Structure of SPARQL Queries

SPARQL queries can be built with four different forms: `SELECT`, `CONSTRUCT`, `ASK`, and `DESCRIBE`.

- `SELECT` as well as `CONSTRUCT` is used to extract data from an endpoint. `SELECT` returns the data in a table format using XML and `CONSTRUCT` returns the data in RDF.
- `ASK` can be used to run a query on an endpoint. It asks if the endpoint could answer the query. It will return a simple `true` or `false` statement.
- The `DESCRIBE` form is used to run queries if the resources' vocabulary is unknown. The returned RDF graph describes the resource.

[22] List of SPARQL Endpoints: `http://www.w3.org/wiki/SparqlEndpoints`
[23] XPATH operators: `http://www.w3.org/TR/xquery-operators`
[24] IRI: `http://www.w3.org/International/O-URL-and-ident.html`

A query is composed of the following parts:

- **Prefix declarations:** A query can include zero or any number of prefix declarations. They are used to abbreviate a long IRI and the defined prefixes are used in the `WHERE` clause.
- **Query result clause:** It defines which of the four different forms (`SELECT`, `CONSTRUCT`, `ASK` or `DESCRIBE`) is used.
- **`FROM` or `FROM NAMED` clause:** The `FROM` or `FROM NAMED` clause defines the used dataset. The statement is optional. The `FROM` clause identifies the contents in the default graph while the `FROM NAMED` clause identifies a named graph.
- **`WHERE` clause:** In the `WHERE` clause the triple patterns are specified.
- **Query modifiers:** Query modifiers such as `ORDER BY`, `LIMIT` or `OFFSET` can be used.

Listing 2.1 shows an example of a SPARQL query. It queries all subjects that have geo-coordinates in a specific area. If the geo-coordinates do not differ more than 0.02 from the point (48.21415, 16.35993), the subject, a label, as well the geo-coordinates will be returned. In the `SELECT` clause it is defined that a subject, latitude, longitude and a label will be returned. The `WHERE` clause selects all triples that contain the required geo-coordinates as well as the label. The option `FILTER` filters all the geo-coordinates that are in the specified area. Moreover, only labels with the language attribute "en" will be returned. Finally, the query modifier `LIMIT` defines that at most 20 results will be returned.

```
1  PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3
4  SELECT ?subject ?lat ?long ?label
5  WHERE
6  { ?subject geo:lat ?lat.
7  ?subject geo:long ?long.
8  ?subject rdfs:label ?label.
9  FILTER (?lat - 48.21415 <= 0.02 && 48.21415 - ?lat <= 0.02 &&
10 ?long - 16.35993 <= 0.02 && 16.35993 - ?long <= 0.02 &&
11 lang(?label) = "en" ).
12 } LIMIT 20
```

Listing 2.1: Example SPARQL query

2.3.5 LOD Datasets

Figure 2.4 shows an extract of the LOD cloud diagram. It summarizes all the datasets that have been published in Linked Data format by contributors to the Linking Open Data community project and others. Currently, the cloud is composed of 295 datasets that include over 31 billion RDF triples. The datasets are connected by about 504 million RDF links (September 2011) [8]. Figure 2.4 represents only a part of the LOD cloud.

Mainly, it contains all the cross-domain datasets marked in grey and all the geographic datasets marked in yellow. In [42] the author gives further information about the LOD diagram. The size of the different circles describing the datasets correspond to the number of triples each dataset contains. The sizes are roughly scaled corresponding to the numbers in Table 2.2. Moreover, the thickness of the arrows indicate the number of links between the datasets. Table 2.3 presents the scaling factor of the arrows.

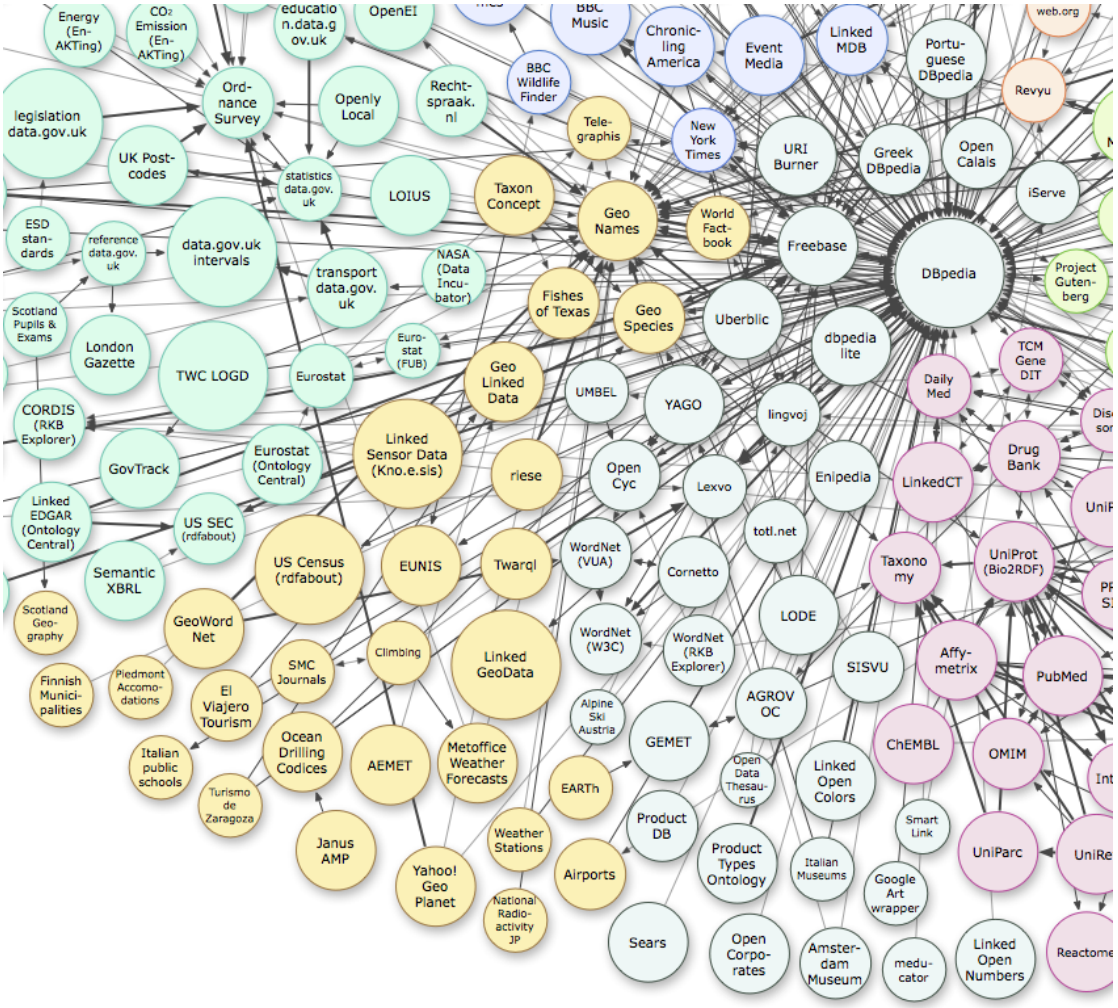

FIGURE 2.4: Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch [25]

2.3.5.1 Interlinking between Open Datasets

The Linking Open Data Community Project [8] is dedicated to extend the Web by setting RDF links between items from distinct data sources. One advantage is that the data returned by the query is in a structured format and can be used within other applications. RDF triples with the predicate `owl:sameAs` indicate that the datasets

[25] LOD cloud: `http://lod-cloud.net`

size circle	triple count
very large	> 1B
large	1B-10M
medium	10M-500k
small	500k-10k
very small	< 10k

TABLE 2.2: Size of datasets in LOD cloud (September 2011)

thickness arrow	triple count
thick	> 100k
medium	100k - 1k
thin	< 1k

TABLE 2.3: Number of links in LOD cloud (September 2011)

refer to the same entity. In [43], the authors describe how DBpedia is interlinked with other Open Datasets. DBpedia is part of the Linking Open Data community project [8]. Because of the fact that DBpedia covers a broad area of topics and intersects with many other datasets, it represents one of the biggest "linking hubs" among the Open Datasets. In [44] a short overview of the datasets that are interlinked with the DBpedia project is provided. Table 2.4 summarizes some of these datasets.

Dataset	Description	Number of Links
GeoNames	GeoNames contains information about places and geographic features.	85,000
flickr wrapper	Photos that are related to DBpedia resources are extracted from flickr and provided as RDF.	1,950,000
Freebase	Freebase contains information about things from various domains. It is an open-license database.	3,300,000
DBLP Bibliography	It contains information about scientific publications.	200

TABLE 2.4: Datasets interlinked with DBpedia

2.3.5.2 DBpedia

The purpose of the DBpedia project [7] is to extract structured information from Wikipedia and to publish it on the Web. As described in [43], the main task of the project is the conversion of Wikipedia content to RDF. Each resource in the DBpedia dataset is identified by a URI reference. The URI reference has the form `http://www.dbpedia.org/resource/NAME`. The name corresponds to the name in the URL of the

Wikipedia article. Generally, each resource is related to the English Wikipedia article. DBpedia converts structured Wikipedia content, such as the infobox templates, images, geo-coordinates, links to external Web pages, or links between the different language editions of Wikipedia to RDF.

An overview of the current size of the project is given in [45]. Currently, the dataset includes 3.77 million (December 2012) resources that are identified by a URI reference. Table 2.5 shows which resources are included in the dataset. Therefore, the resources are classified in a consistent ontology. These resources do contain labels as well as abstracts

Category	Triples
persons	416 000
places	526 000
includes populated places	360 000
music albums	106 000
films	60 000
video games	17 500
organizations	169 000
includes companies	40 000
includes educational institutions	38 000
species	183 000
diseases	5 400

TABLE 2.5: DBpedia dataset: Categories supported by the DBpedia project (July 2011)

in up to 97 different languages. Moreover, the dataset contains about 2.7 million links to images and it is linked to other external web pages with about 6.3 million links. In addition, it supports 6.2 million external links to other datasets that are using RDF content.

The DBpedia dataset contains about 697 000 resources that contain geographic coordinates. The W3C Basic Geo Vocabulary[26] is used for expressing the coordinates. The code sample 2.2 shows a point with geo-coordinates expressed in RDF. The geo-coordinates enable to write queries, such as "find all POIs next to the specified geo-coordinates".

```
1 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
2          xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
3          <geo:Point>
4                  <geo:lat> 48.205456 </geo:lat>
5                  <geo:long> 16.373062 </geo:long>
6          </geo:Point>
7 </rdf:RDF>
```

LISTING 2.2: Expressing Geo-Coordinates in RDF

[26]W3C Basic Geo Vocabulary: `http://www.w3.org/2003/01/geo/`

In order to access these resources, DBpedia offers three mechanisms: Linked Data[27], the SPARQL protocol, and downloadable RDF dumps[28].

Linked Data Linked Data relies on URIs as resource identifiers and the HTTP protocol to retrieve descriptions of the resources. It is a method of publishing RDF data on the Web. Resource identifiers are used to return RDF descriptions and the information can be presented in an HTML view in a traditional web browser.

SPARQL endpoint DBpedia provides a SPARQL endpoint that is used for querying the dataset. The endpoint is at `http://dbpedia.org/sparql` and is hosted by Virtuoso Universal Server[29]. The SPARQL query language is described more detailed in Chapter 2.3.4.

RDF dumps Furthermore, the DBpedia project offers RDF dumps for accessing the dataset. Datasets in the N-Triples serialization format can be directly downloaded.

2.3.5.3 LinkedGeoData

LinkedGeoData [46] extracts structured information from the OpenStreetMap project[30] and makes it accessible on the Web as an RDF knowledge base. Additionally, the data is interlinked to other knowledge bases such as DBpedia. Currently, the database is composed of about 350 million nodes and 30 million ways (September 2011). LinkedGeoData includes about 2 billion triples. The dataset consists of individual nodes that represent points and ways interlinking between nodes and forming collections of nodes like streets or rivers. Just as DBpedia and GeoNames, the latitude and longitude values are in WGS84[31] coordinates.

LinkedGeoData offers several possibilities for accessing the dataset. Generally, all files are available for download[32]. Besides that, LinkedGeoData offers the following two interfaces for accessing the data online:

REST API LinkedGeoData provides a REST API that enables access to the database. It allows to access all nodes and ways and additionally, content negotiation is supported. RDF data for individual nodes and ways can be obtained with the following URLs:

```
http://linkedgeodata.org/triplify/node206121402
http://linkedgeodata.org/triplify/way23289876
```

[27] Linked Data Tutorial: `http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/`
[28] RDF dumps: `http://wiki.dbpedia.org/Downloads36`
[29] Virtuoso Universal Server: `http://virtuoso.openlinksw.com`
[30] OpenStreetMap: `http://www.openstreetmap.org`
[31] WGS84 coordinates: `http://dev.w3.org/geo/api/spec-source.html`
[32] LinkedGeoData download: `http://downloads.linkedgeodata.org`

Additionally, the REST API allows to query data within a circular and rectangular area using the following URLs:

```
http://linkedgeodata.org/triplify/near/48.213006,16.360531/1000/
http://linkedgeodata.org/page/near/48.20877-48.21300,16.36053-16.36962/
```

However, the REST API shows some disadvantages. Currently, it is not able to query relations between nodes. Thus, the query capabilities are limited. Besides, at the moment the database used by the REST API is not synchronized with the OpenStreetMap project.

SPARQL endpoints The SPARQL endpoint offers full query capabilities. More than 220 million triples can be queried. Nevertheless, the entities and tags are filtered and that is why it is not possible to retrieve all available data.

2.4 3D Viewing

The use of geopositioning technologies for creating an AR application implies the transformation of three-dimensional coordinates to two-dimensional screen coordinates. Therefore, the concepts of 3D viewing have to be realized. The authors of [47] present the general processing steps for converting a three dimensional point to two dimensional device coordinates. Figure 2.5 illustrates the general 3D transformation pipeline for converting modeling coordinates to the device internal coordinate system. Modeling coordinates are the individual coordinates of each object.

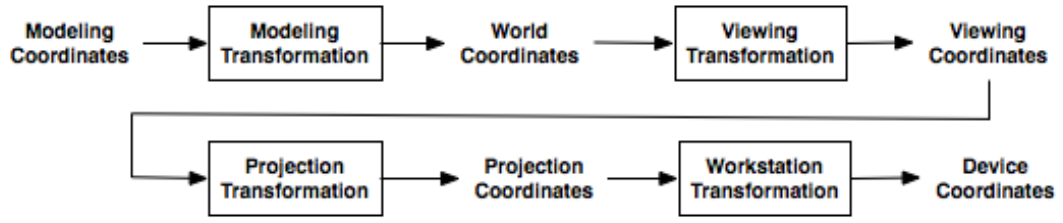

FIGURE 2.5: Three-dimensional transformation pipeline [47]

Since each object has its own modeling coordinates the coordinates do not correlate with coordinates of other objects within the world space. Therefore, the coordinates have to be transformed to world coordinates. Consequently, all the objects are located in the world space. Because of the fact that the world space is viewed by a camera, the world coordinates have to be transformed to viewing coordinates. The viewing coordinate system is used for specifying the viewing position of the observer and the position of the projection plane on which the objects will finally be projected. The projection plane is

illustrated in Figure 2.9. The plane is positioned between the object and the camera and the object is projected onto this plane. Therefore, the viewing coordinates are transformed to coordinates on the projection plane. The projection transformation is described more detailed in Section 2.4.1. Once the coordinates on the projection plane are calculated, they have to be transformed to device coordinates. In doing so, the 3D coordinates are converted to the pixel coordinates of the screen.

2.4.1 3D Projection

A 3D projection is a method for mapping a three-dimensional point onto a two-dimensional plane. Generally, a projection is a mathematical operation that converts an object from n dimensions into n-1 or fewer dimensions. There exist various types of projections [48]. Figure 2.6 gives a summary of the three main classes of projections: parallel, perspective, and nonlinear.

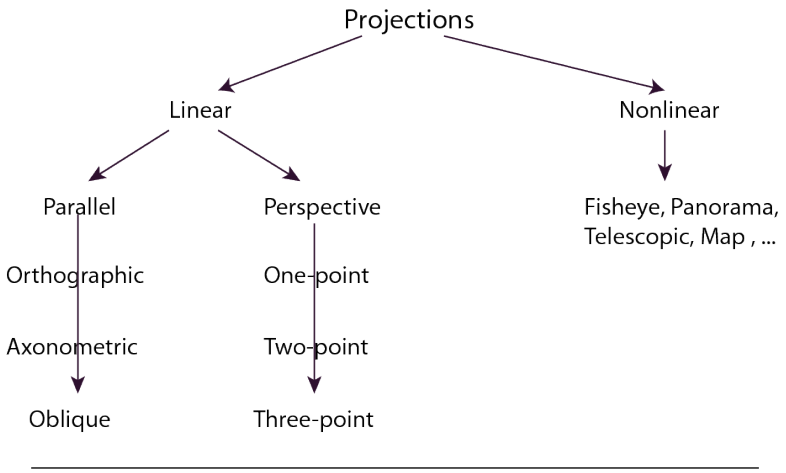

FIGURE 2.6: Three-dimensional projection types [48]

In [48], the author explains the different projection types. Each of the mentioned main classes can be divided into different subclasses. For instance, the perspective projection is distinguished into one-point, two-point, or three-point perspective projections. For visualizing the resources in this master thesis, the linear projection, more precisely the perspective projection, is used because this projection type considers the distance of the objects. Contrary to the parallel projection where the center of projection is at infinity, the center of the perspective projection is at the observer. Visualizing such an AR application implies that the center of projection is at the user's eye, thus at a finite distance. Figure 2.7 illustrates the principle of a parallel projection. It transforms the coordinates along parallel lines onto the projection plane. In contrast, Figure 2.8 shows a perspective projection that transforms the coordinates along lines that run together at the center of projection also called the projection reference point as defined in [47]. The perspective projection allows to create the illusion of depth in a two-dimensional drawing.

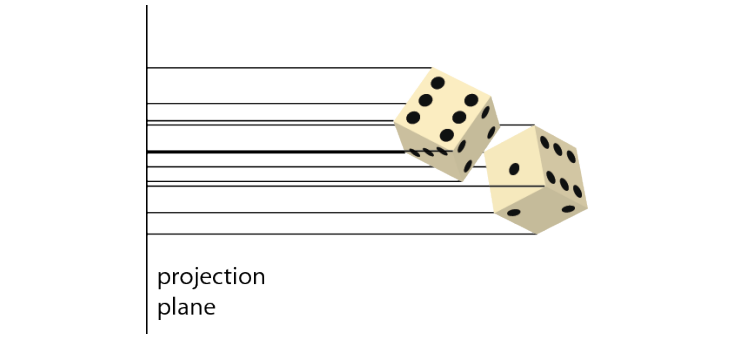

FIGURE 2.7: Parallel projection

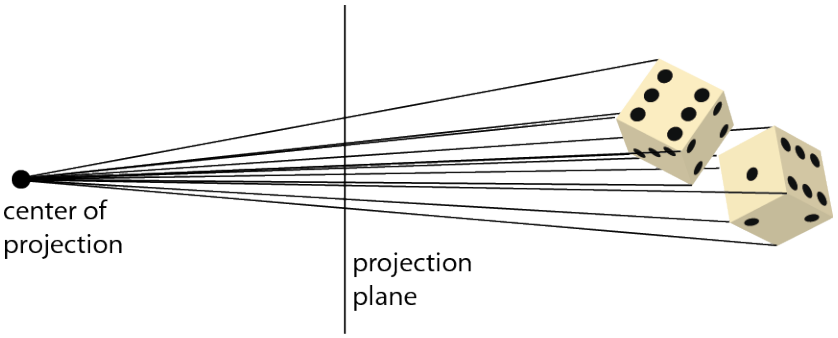

FIGURE 2.8: Perspective projection

2.4.1.1 Perspective Projection

In this work we use the perspective projection because it considers the distance of the points that have to be transformed. This is important because if the camera moves, objects that are far away stay longer in the camera's field of view than objects that are nearby. The authors in [47] provide a detailed explanation of the perspective projection. As already mentioned, the perspective projection considers the fact that objects in the distance appear smaller than objects near by. Thus, the points have to be transformed along lines that meet at the projection reference point. Generally, the mathematical model implies an object to be projected, a projection plane, and a viewer that is watching the projection on this plane. The plane is positioned between the 3D object and the viewer. The challenge is now to define what the viewer will see on the projection plane. The calculation of the perspective projection is explained in [48]. There exist a point $P = (x, y, z)$ and this point should be transformed onto the plane by computing the two-dimensional coordinates (x', y') of its projection P'. Therefore, a transformation T that transforms the point ($P' = PT$) has to be determined. Basically, the mathematical model for the perspective projection depends on the position and orientation of the viewer and the projection plane as well as the position of the object. Figure 2.9 shows the transformation of a point P with the coordinates (x, y, z) to a position P' on the projection plane with the coordinates (x', y'). In order to compute the coordinates of $P'(x', y')$ the following simple relations can be derived:

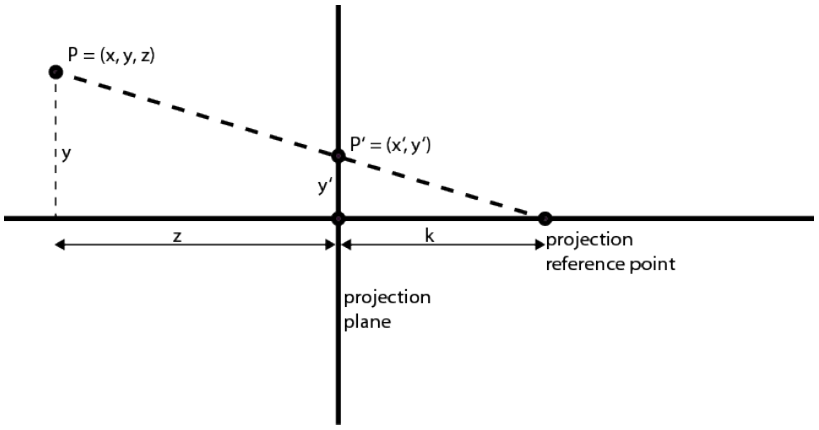

FIGURE 2.9: Projection plane: converting a point $P(x, y, z)$ to screen coordinates $P'(x', y')$

$$\frac{x'}{k} = \frac{x}{z+k} \text{ and } \frac{y'}{k} = \frac{y}{z+k}$$
$$x' = \frac{x}{(z/k)+1} \text{ and } y' = \frac{y}{(z/k)+1} \tag{2.1}$$

where k is the distance between the viewer and the projection plane. The denominator can be zero only if $z/k = -1$. Because of the fact that k is positive and z is not negative, it will not be zero. The result represents the projected point $P'(x', y')$.

2.4.2 3D Coordinate Systems - Left-handed and Right-handed System

Generally, transformations in three dimensions are more complex than in two dimensions because there exist more directions for rotations. Three dimensional coordinate systems are distinguished into two different systems called left-handed and right-handed. The systems are illustrated in Figure 2.10.

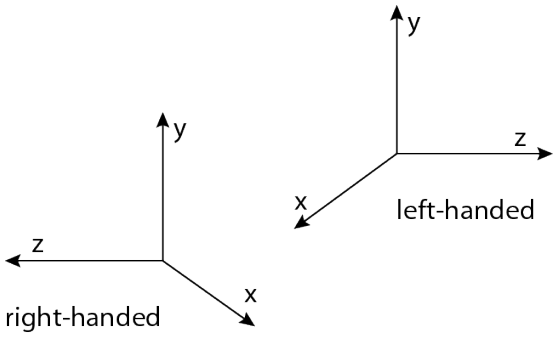

FIGURE 2.10: Left-handed and right-handed coordinate system

Especially, the transformation of 3D coordinates to 2D screen coordinates requires the distinction of these systems. In [48], the difference between the two coordinate systems

is explained. A left-handed coordinate system is constructed in the following way. It can be remembered by holding up the left and right hand. The thumb is aligned with the positive x-axis and the index finger points up illustrating the y-axis. Now the middle finger points in the direction of the z-axis.

Android devices make use of a right-handed coordinate system. The Android coordinate system is explained more detailed in Section 2.2.4.2. In [49], the author explains the conversion from left-handed to right-handed coordinate systems. A point in the left handed coordinate system can be easily converted to a point in the right-handed coordinate system by flipping the direction of one of the axes. Basically, it does not matter which axis is inverted. For instance, the point $P(x, y, z)$ in the left-handed system can be converted to the point $P(x, y, -z)$ in the right-handed system.

2.5 Summary

In order to define an approach that combines the technologies of AR as well as Linked Data, it is important to understand the basics of AR, mobile software platforms, Linked Open Data as well as 3D viewing. In this chapter, we summarized the most important facts about these topics.

AR simplifies the user's life by bringing virtual information to her environment. Thus, her perception of the real world is enhanced. As opposed to virtual reality, where the user is located in a synthetic environment, AR combines the real world with virtual objects. AR allows the user to get a better sense of reality. Since the user can see the real world through her device, the user gets a better impression of the surrounding area. AR systems can be either fixed or mobile. Mobile systems allow the user to move around while fixed systems are placed at a predefined position. Moreover, it can be distinguished between systems that work either indoors, outdoors, or both indoors and outdoors. Our approach bases on a mobile outdoor system. Furthermore, there exist some techniques for identifying the objects and resources situated in the user's environment. An image recognition algorithm can be used for comparing images of the surrounding area with images in a database. A location based algorithm is another technique for identifying objects. For that, sensors of a mobile device are used to determine the position of the camera and the objects as well as the orientation of the camera. By means of a 3D projection, points can be represented in the camera's field of view. Besides, these two algorithms can be combined for getting more accurate results.

In order to use geopositioning technologies for identifying resources located in the environment, it is necessary to understand the concepts of 3D viewing. The 3D transformation pipeline illustrated in Figure 2.5 represents the essential transformations for the conversion of 3D modeling coordinates to 2D screen coordinates. In Chapter 2.4.1, an overview of the different types of projections is provided. The linear projections

are distinguished between parallel projections and perspective projections. At a parallel projection, the coordinates are transformed along parallel lines onto the projection plane; in a perspective projection, the coordinates are transformed along lines that run together at the center of projection. The projection types are illustrated in Figure 2.7 and 2.8. We decide to use the perspective projection in our approach because it allows to create an illusion of depth in a 2D drawing. Generally, a perspective projection consists of an object, a projection plane and a viewer watching the projection. Thus, the mathematical model depends on the position and orientation of the viewer and the projection plane as well as the position of the object. Figure 2.9 illustrates the conversion of a 3D point to 2D screen coordinates.

The presentation of points in the camera view of a mobile device implies that data about the resources are available. Therefore, we integrate RDF data from Linked Data sources such as DBpedia or LinkedGeoData. Chapter 2.3 gives an overview of the concept of LOD. As already mentioned, Linked Data describes a method for publishing and interlinking structured data on the Web. RDF is a standard model for representing and interchanging information on the Web. The concepts and serialization formats of RDF are described in more detail in Section 2.3.3. The DBpedia project extracts structured information from Wikipedia and makes it accessible on the Web. As described in Chapter 2.3.5.2, the current dataset includes 3.77 million (December 2012) resources identified by a URI reference. The project allows to access data via the SPARQL query language. SPARQL is a query language for RDF data that is able to query graph patterns. Moreover, the DBpedia dataset is linked to other open datasets such as LinkedGeoData or GeoNames. LinkedGeoData extracts information from the OpenStreetMap project and makes it accessible on the Web as an RDF knowledge base. Just as DBpedia, LinkedGeoData offers a SPARQL endpoint for querying RDF data.

Chapter 3

Related Work

In this chapter, a general overview of works that realize an AR interface or process Linked Open Data is provided. We provide a structured and consolidated view on relevant research. Further, we discuss the differences of these systems compared to our approach. The first part of this chapter deals with the different features we selected for comparing the projects. The second part gives an overview of the chosen projects and provides a table that compares them according to the presented features. Finally, we discuss the differences in relation to our concept and summarize the most important facts.

3.1 Feature Description

In this section, we will describe different criteria for comparing related works. Particularly, we select features that focus on the used approach.

Main algorithm The main algorithm describes the basic approach of the applications for detecting objects in the camera's field of view. The techniques are explained more detailed in Section 2.1.1. We categorized the feature into the following 4 types:

- Image recognition algorithm
  The algorithm bases on the comparison of images for identifying the objects located in the environment.
- Location-based algorithm
  A location-based AR system uses GPS positions as well as the orientation of the mobile devices for calculating the position of the points in the user interface.
- Orientation sensors
  The orientation sensors of a mobile device are often used in addition to another approach such as an image recognition or a location-based algorithm for achieving a better result.
- Visual tracking of markers
  Systems that base on the tracking of markers are using visual tags that are integrated in the real world for orientation.

Data source The data source gives an overview about the used data. An AR application that visualizes resources in the camera view requires a database. If the main algorithm bases on the comparison of images, a database with numerous pictures will be required. Similarly, a location-based algorithm requires an extensive database containing geographic information about the resources. For instance, some AR applications use a database with user-generated content. Moreover, LOD sources or data accessed directly from Wikipedia are used.

Data format The data format determines the format in which the data has to be in order to be processed and interpreted correctly by the system. It is important to provide data in a workable format for expanding the system. In the majority of cases, data is processed in XML, JSON, or RDF. XML as well as JSON are open standards that are suitable for data interchange. RDF is a general method for describing or modeling information and it uses several serialization formats. RDF and its serialization formats are described more detailed in Section 2.3.3.

Mobile platform The mobile platform describes the operating system that controls the device. The most known mobile software platforms are summarized and described in Chapter 2.2. The mobile platforms differ in some points. For instance, contrary to iOS or Windows Phone, Android OS is an open source platform. Moreover, the programming language as well as the license type differ from each other. Though, all most known platforms offer an open API/SDK for developers.

Indoor/outdoor The feature indoor/outdoor describes if the AR application can be used either indoors or outdoors. Some applications can be used indoors as well as outdoors. Most often, it depends on the used main algorithm. Location-based applications mostly depend on the GPS signal, which only can be used outdoors. Contrary to that, image recognition algorithms can recognize objects indoors, too. The algorithm depends on the comprehension of the database. If the database contains images of objects situated indoors as well as outdoors the application will be used both ways.

Visualization technique The visualization technique describes the basic technique for visualizing data in the user interface. For instance, data can be illustrated in an AR interface that displays POIs in the mobile device's camera view. Another method is to visualize data in a 2D map. Thereby, the POIs are presented on a map and the user can zoom manually. Moreover, some applications ask the user to take a photo and afterwards the application provides information about specific parts of the picture. Consequently, information about the POIs is provided on a static photo and not in the camera view of a mobile device.

Application infrastructure The application infrastructure describes the basic construction of the system. We compare AR systems that base on the following architectural styles:

- stand alone - stand alone architecture
  The user interface and the data access is combined into a single program. It is independent from other computing applications.
- C/S - client/server architecture
  Each computer or process in the network is either a client or a server. One or more servers offer services to one or more clients. The user only interacts with the client. The clients rely on servers for resources such as files.
- three-tier - three-tier architecture
  It is a client-server architecture in which, e.g., the user interface, data access, or the data storage are developed as independent modules. The presentation, the application processing, and the data management are logically separate processes.

User-generated content The feature "user-generated content" shows whether the user is allowed to upload his own content to the application. Some AR applications are based on content that is created from users. Additionally, there exist applications that permit users to upload information about specific points and share them with other users. This provides an opportunity to extend the database with additional information.

3.2 Comparison of Works

Based on the criteria described in Section 3.1, we will compare several works. In the following, we give a short overview of the selected projects. The projects illustrate the different techniques for calculating the resource's position in the camera's field of view. Additionally, projects that visualize data in a 2D map are compared to projects that realize AR. Moreover, we picked out projects that use LOD sources in a similar context.

Wikitude Wikitude is the first available mobile AR browser worldwide. The application scans the environment using the sensors and the camera of the mobile device. Therefore, it requires a compass, an accelerometer, GPS, as well as a camera. After analyzing the geobased data the information of the objects is displayed in the camera at the position of the real objects. Wikitude allows the user to define her own information or points of interest in so called 'Worlds'. Contrary to LOD sources, the displayed information is collected from closed databases in proprietary format. But yet, the dataset shows similarity to Linked Data because it is not constrained to one source. Users can create their own Worlds.

Layar Layar is an Android application that visualizes digital information on real images similar to the application "Wikitude". The Layar platform covers a wide area of AR features. The application allows users to play games within their environment or to

visit a virtual shop. One part of the application builds the location-based layers that are similar to Wikitude and allow users to get information about nearby locations, shops, monuments, or other POIs.

Mixare [4] Mixare is an open source AR engine that is currently available for the Android platform and for iOS. It is freely expandable and can be modified individually. The application uses a webservice offered by GeoNames for querying cities and placenames in a specific area. Moreover, users can generate additional content.

DBpedia Mobile [6] DBpedia Mobile is an application that visualizes DBpedia content. The client application for mobile devices allows users to access information about the resources located in their surrounding area. DBpedia Mobile renders a 2D map displaying the nearby POIs and supplies background information about the locations. Moreover, it is possible to navigate along links to related datasets on the Semantic Web. Thus, the user can navigate into related datasets and explore background information about them. DBpedia Mobile is implemented as a client-server application. The application determines the current position applying the GPS-coordinates of a mobile device. SPARQL queries are used to obtain the required information from the DBpedia project for visualizing the POIs on a 2D map.

Belimpasakis et al. [50] Belimpasakis et al. describe an AR system that queries user-generated content from Internet services. Therefore, a sensor based tracking solution is applied and metadata is used for visualizing the POIs at the correct screen position. The authors use EXIF metadata of the images for determining the GPS position of the POIs as well as the orientation data of the image. Moreover, a so called 'Content Aggregation Service' that serves as intermediate service is implemented. The service communicates with Internet services such as Flickr[1] and handles additional tasks such as filtering mechanisms. The authors proved their approach by implementing an AR prototype that displays images about the POIs queried from Internet services.

Gray et al. [51] Gray et al. developed an application that takes advantage of community-driven websites with user-generated content. The authors designed a system that allows to query for information about photos simply by snapping a photo. The implemented system crawls Wikipedia to find photos and information about them. On the one side, geopositioning is used for determining the area where the photo has been taken. On the other side, image matching algorithms are implemented for detecting matching photos. The application bases on a Client-Server architecture. The server is responsible for crawling community driven websites and the client provides a user-interface that allows to snap photos and retrieve related photos and information about them.

[1] Flickr photo sharing site: `http://www.flickr.com/`

Gammeter et al. [52] Gammeter et al. present a system for visualizing augmented content based on visual recognition. The infrastructure of the application bases on a Client-Server architecture and they proof their approach by implementing an application on the Android platform. Since the approach bases on a visual recognition algorithm, stationary POIs as well as non-stationary objects can be recognized. Consequently, the application can be used indoors as well as outdoors. The used database consists of a huge amount of landmarks as well as media covers.

Choubassi et al. [53] and Takacs et al. [54] Choubassi et al. and Takacs et al. present mobile AR systems that base on the usage of an image matching algorithm as well as location-tagged images. However, both papers focus on the implementation of an image matching algorithm. Therefore, a huge database with location-tagged images is used and camera images are matched against this database. Either the authors used a database that consists of geotagged English Wikipedia pages or a database with content from users as well as geotagged images.

Reitmayr & Schmalstieg [55] Reitmayr & Schmalstieg present an indoor AR system. Contrary to the other systems mentioned in this chapter, the system bases on the visual tracking of markers and it can be used indoors. The AR system can only be used in an area that contains predefined markers. In order to realize this system, they use an HMD as well as a notebook connected through WLAN.

Omercevic & Leonardis [56] Omercevic & Leonardis developed an AR system that displays hyperlinks about objects located in the environment. The user just snaps a photo of her surroundings and hyperlinks that refer to information about the objects are displayed. They implemented a three-tier architecture. The client captures a picture and sends it to the application server. The application server modifies the image by adding information such as the focal length and sends it to the computer vision server. This server visualizes the results as a web page. The authors implement an image matching algorithm for identifying the objects in the picture.

van Aart et al. [57] van Aart et al. developed a mobile cultural heritage guide that allows users to explore their surroundings. The authors combine two different kinds of knowledge: general knowledge from LOD databases such as DBpedia, LinkedGeoData or GeoNames and specific knowledge about the cultural heritage domain. The calculations entirely base on a geoposition approach. The guide is visualized in the form of a map that visualizes nearby POIs as well as an AR interface.

WikEye [58] WikEye is an AR project that links objects on a map to their corresponding Wikipedia entries developed by Hecht et al. Therefore, the user has to point the camera of a mobile device at a printed map. The realization of a projective mapping from the map coordinate system to the image coordinate system allows the display to

act as a viewfinder. The user interface displays information about the recognized objects on the map. However, the project depends on predefined markers that has to be defined in advance. Consequently, the application has to be aware of the map. The application receives all information from Wikipedia.

Table 3.1 provides an overview of the applied criteria and the projects.

Author	Main algorithm	Data source	Data format	Mobile platform	indoor/ outdoor	Visualization technique	Application infrastructure	User-created content
Wikitude [2]	location-based	N/A	ARML	Android, iOS, Bada, Blackberry, Windows Phone 7	O	AR	stand alone	✓
Layar [3]	location-based	N/A	JSON	Android, iOS, Symbian, BlackBerry	O	AR	C/S	✓
Mixare [4]	location-based	Wikipedia, GeoNames	JSON	Android, iOS	O	AR	stand alone	✓
Becker & Bizer [6]	location-based	LOD - DBpedia	RDF	mobile devices (Opera Mobile 8) & desktop	I+O	map	C/S	✓
Belimpasakis et al. [50]	location-based	internet services e.g. Flickr	XML, JSON	Symbian OS - S60 platform	O	map + AR	C/S	✓
Gray et al. [51]	image recognition + location-based	Community driven websites - Wikipedia, Flickr	N/A	Intel ATOM powered MID	I+O	photos	C/S	N/A
Gammeter et al. [52]	image recognition + orientation sensors	N/A	XML	Android	I+O	AR	C/S	N/A
Choubassi et al. [53]	image recognition + orientation sensors	Wikipedia - photos with geotagged features	N/A	MID with Intel Atom TM processor	N/A	AR	C/S	N/A
Takacs et al. [54]	image recognition	database with content from users - geo-tagged images	N/A	Symbian OS - Nokia N95	O	AR	C/S	✓
Reitmayr & Schmal-stieg [55]	location-based + visual tracking of markers	N/A	XML	notebook, Windows XP, WLAN, HMD	I	AR	N/A	N/A
Omercevic & Leonardis [56]	image recognition	Open Data - GUIS107[2]	XML	MID	O	photos	three-tier architecture	N/A
van Aart et al. [57]	location-based	LOD - DBpedia	RDF	iOS	O	map + AR	C/S	N/A
Hecht et al. [58]	image recognition + visual tracking of markers	Wikipedia	N/A	Symbian OS - Nokia N80	I+O	AR	N/A	N/A

Table 3.1: Related work

3.3 Discussion and Summary

In this section, we discussed the similarities and differences of related approaches compared to our work. Generally, our approach shows some similarities to the basic approach from DBpedia Mobile. Principally, we want to use some basic concepts of the application such as the integration of the LOD source project DBpedia and expand the visualization of the data by integrating an AR user interface. Similar to DBpedia Mobile, information is requested from the DBpedia project through SPARQL queries. Contrary to DBpedia Mobile, additional information is queried, e.g. from LinkedGeoData through SPARQL queries. Moreover, in contrast to DBpedia Mobile, the application is not implemented as a client-server application. All calculations are executed directly on the mobile device. Since we develop an AR framework, it offers a more realistic impression of the environment compared to a 2D map as implemented by DBpedia Mobile. Moreover, DBpedia Mobile only queries data from the DBpedia project. Our work allows to query data from different LOD sources.

An application that visualizes a map view as well as an AR view was implemented by Belimpasakis et al. Contrary to our thesis in which we use RDF data for determining the position of the resources, the authors use EXIF metadata of the images that contain the GPS position as well as orientation data. Another difference to our system is that they use an intermediate service that communicates with Internet services such as Flickr[3] and handles additional tasks like filtering mechanisms. We decided to communicate directly with the LOD databases because we can address all necessary data directly with SPARQL queries and filter the data during this process simultaneously. Moreover, LOD databases provide data in RDF format.

DBpedia Mobile as well as the application implemented by Belimpasakis et al. are both location-based applications. Contrary to that, Gray et al. use geopositioning technologies only for improving the image matching results. Moreover, the application described in [51] bases on a client-server architecture. The server is responsible for crawling community driven websites and the client provides a user-interface that allows to snap photos and retrieve related photos and information about them. In contrast, our approach bases on the presentation of information about the resources in the camera's live view. It is not necessary to snap photos for obtaining information about points that appear in the user-interface. Similar to our approach, they use user-generated content for determining the resources' positions and getting information about them.

Gammeter et al. present an approach for visualizing content that bases on an image recognition algorithm. Contrary to our approach, this application is able to recognize stationary as well as non-stationary objects. Equally to our thesis, the implemented

[3]Flickr photo sharing site: `http://www.flickr.com/`

application relies on the Android platform but they decided to use a client-server architecture.

The summarized projects in Table 3.1 show some similarities. We particularly compare the used algorithms as well as the databases. Generally, the projects can be divided according to their main algorithm. For instance, the projects [2], [4], [6], or [50] only use a location-based algorithm. Whereas, the projects [54] or [56] only implement a visual recognition algorithm. Some projects use the orientation sensors in addition to an image recognition algorithm for improving the accuracy such as [52] or [53]. Although, there exist some similar AR applications none of them use data in RDF format from LOD databases in the process of locating certain resources. Contrary to already existing AR applications, we will combine the approaches of using LOD sources in an AR user interface. Additionally, our approach will implement a location-based algorithm.

Chapter 4

Approach

In order to build a framework visualizing resources that are in the current view area of an AR user interface, we define an approach that combines AR technologies as well as LOD. In Figure 4.1, we summarize the approach of visualizing LOD resources in the camera's current field of view cf. Definition 4.2.

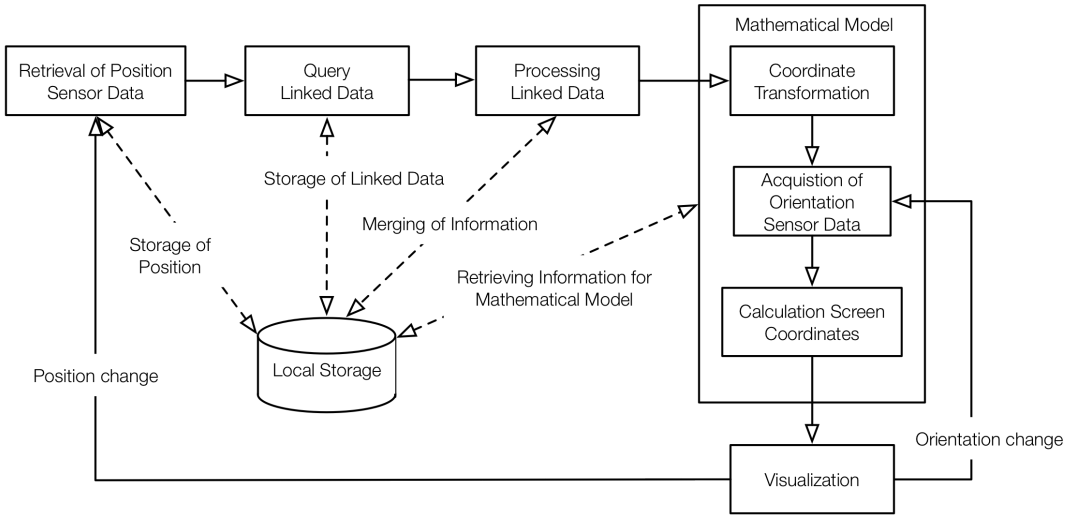

FIGURE 4.1: Overview of the conceptional model

Our approach consists of the following parts: first, sensor data such as the position of the device is determined. After storing the geographic position of the camera, Linked Data is queried for discovering resources that contain geographic information and that are located within a predefined radius cf. Definition 4.1. Next, we consider a mapping framework to filter identical resources retrieved from different LOD sources. Afterwards, the mathematical model for calculating the screen coordinates of the resources is realized. Within the mathematical model orientation sensor data of the mobile device is queried. After determining the screen coordinates of each resource, the resources are displayed in the camera interface. Once the orientation of the mobile device changes,

new orientation data is queried and the screen coordinates are recalculated. If the position of the mobile device changes, a new position will be queried and the process starts from the beginning with new position data.

Definition 4.1. [Resource] *Let R be a set of resources. We define a resource r as* $\forall r \in R : \exists! p \in P \leftrightarrow r \in [radius]$. *A point $p \in P$ is a 3-tuple* $p = \mathbb{LAT} \times \mathbb{LNG} \times \mathbb{ALT}$ *where* $\mathbb{LAT} := \{ lat \,|\, lat$ is a physical addressable latitude coordinate$\}$, $\mathbb{LNG} := \{ lng \,|\, lng$ is a physical addressable longitude coordinate$\}$, *and* $\mathbb{ALT} := \{ alt \,|\, alt$ is a height indication$\}$

Definition 4.2. [Field of view] *The camera's field of view describes which the part of the world is currently displayed in the camera's interface. The field of view is depicted by the horizontal field of view as well as the vertical field of view in degrees. The field of view is illustrated in Figure 4.2.*

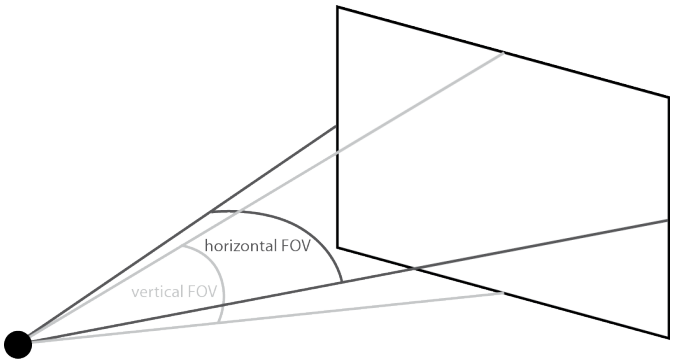

FIGURE 4.2: Camera's field of view

This chapter is structured as follows: first, we summarize the symbols and constants defined in our approach in Table 4.1 and 4.2. In Section 4.1, we describe the use of data from GPS sensors for determining the camera's position. In the following Section 4.2, we describe the process of retrieving GPS data from LOD sources by means of SPARQL queries. After that, we present an approach for the consolidation of LOD sources in Section 4.3. In Section 4.4, we define an approach for converting actual GPS coordinates of the resources to screen coordinates in the current camera's field of view. Besides the mathematical model for displaying the resources, we present an approach for visualizing off-screen objects cf. Definition 4.3 in mobile AR for improving usability in Section 4.5.

Definition 4.3. [Off-screen objects] *Off-screen objects are all resources that are currently not in the field of view cf. Definition 4.2. Off-screen objects that are within the specified radius but not in the camera's field of view are still represented in the minimap.*

Symbol	Description
R	set of resources cf. Definition 4.1
lat	latitude in decimal degrees
lng	longitude in decimal degrees
alt	altitude in meters
β	bearing in degrees
ppm	pixels per meter
w	screen width in pixels
h	screen height in pixels
X_C, Y_C, Z_C	coordinates of the camera's position in decimal degrees
X_P, Y_P, Z_P	coordinates of resource's position in decimal degrees
$X_P^{trans}, Y_P^{trans}, Z_P^{trans}$	position of point after transforming coordinates relative to camera's position in meters
$X_P^{rot}, Y_P^{rot}, Z_P^{rot}$	position of point after rotating coordinates according to the camera's orientation
X_P^{proj}, Y_P^{proj}	position of point after applying the perspective projection
$X_P^{screen}, Y_P^{screen}$	position of point after converting to screen coordinates
$X_\Theta, Y_\Theta, Z_\Theta$	orientation of the camera (azimuth, pitch, roll) in degrees
$\Delta(X_1, X_2)$	difference between two latitude or two longitude values
μ	maximum difference between two latitude values
λ	maximum difference between two longitude values
$f_{distance}(P_1, P_2)$	distance between two geographic coordinates in meters
$hFOV$	horizontal field of view of the camera interface in degrees
$vFOV$	vertical field of view of the camera interface in degrees
X_M, Y_M	screen coordinates of the center of the minimap in pixels
X_N, Y_N	screen coordinates of the most northern point within the minimap in pixels
α	rotation angle of points displayed in the minimap in degrees
X_R, Y_R	screen coordinates of the reference point in the minimap in pixels
X_P^{map}, X_P^{map}	screen coordinates of the points displayed in the minimap in pixels
r	predefined radius for querying and displaying resources in meters
d	distance between the camera's position and a resource's position in meters

Table 4.1: Symbols

Name	Symbol		Value
Earth radius	$\mathcal{R}$	=	6371 km

Table 4.2: Constants

4.1 Sensor Data - Detection of the Camera's Position

As illustrated in Figure 4.1, the visualization of resources in the camera's interface depends on the camera's position. Basically, our approach relies on the usage of a mobile device that deploys orientation as well as position sensors. In order to use the GPS coordinates in our approach, some requirements have to be fulfilled:

Accurate signal Our approach relies on an accurate and stable GPS signal. The closer the resources are located to the camera, the more accurate the position of the camera has to be. In order to minimize such inaccuracies, we propose to use additional approaches such as the determination of the position by using cell-towers and Wi-Fi signals as described in Chapter 2.2.3.

Completeness Moreover, our approach relies on the completeness of the geographic coordinates which means that it is important to get a significant value for each coordinate (latitude, longitude, and altitude). If the GPS signal does not return a value for the height above the sea level, we propose to use an additional method for retrieving the necessary information. For instance, the missing altitude value can be determined by using the Google Elevation API[1] or by the use of a web service offered by Earthtools[2].

Format of coordinates In order to process the coordinates in the mathematical model, the latitude and longitude values of the camera's GPS position have to be converted to decimal degrees and the altitude has to be in meters.

Once the latitude, longitude, and altitude values of the camera are determined and fulfill the requirements, resources that are located in the surroundings can be queried and the information can be processed in the mathematical model for calculating the screen coordinates of the resources.

4.2 Query Linked Open Data Sources

After acquiring all sensor information, Linked Open Data sources are used to determine resources that are situated nearby the camera's position. Our approach bases on SPARQL cf. Section 2.3.4 which is a query language for RDF cf. Section 2.3.3.

Listing 4.1 shows an example SPARQL query for retrieving geographic information about resources situated nearby the camera's position. The example query is constructed as follows. It selects the subject, that functions as identifier as well as the latitude and longitude information. For selecting only the resources in the surrounding area we define a filter. The variables `LAT` and `LNG` describe the camera's position X_C, Y_C. In the filter we select all resources that are located within a predefined area. This area is restricted by

[1] Google Elevation API: `https://developers.google.com/maps/documentation/elevation`
[2] Earthtools: `http://www.earthtools.org`

the constants `MAX_DIFF_LAT` and `MAX_DIFF_LNG`. These constants describe the maximum distance between the latitude of the camera's position X_C and the resource's position X_P as well as the maximum difference between the longitude of the camera's position Y_C and the resource's position Y_P.

```
1 SELECT ?subject ?latitude ?longitude
2 WHERE {
3     ?subject geo:lat ?latitude.
4     ?subject geo:long ?longitude.
5     FILTER(
6         ?latitude - LAT <= MAX_DIFF_LAT && LAT - ?latitude <= MAX_DIFF_LAT
7         && ?longitude - LNG <= MAX_DIFF_LNG && LNG - ?longitude <= MAX_DIFF_LNG
8     ).
9 }
```

LISTING 4.1: Example SPARQL query for retrieving resources within a predefined area

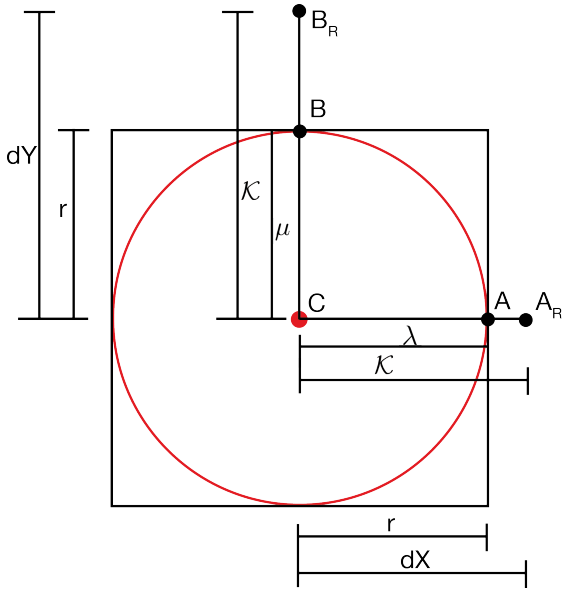

FIGURE 4.3: Calculation of the range of latitude and longitude for processing SPARQL queries according to a predefined radius r

Before querying resources from LOD sources, the area has to be predefined by determining the maximum distances. In order to calculate the maximum differences μ, λ, we assume that the camera's position X_C, Y_C in decimal degrees is already determined cf. Section 4.1 and a radius r in meters is predefined. Generally, our approach of using SPARQL queries allows to query all resources within a square. This square is illustrated in Figure 4.3. In order to determine a range within which the resources have to be located, we calculate the maximum and minimum latitude and longitude of a resource located in the area. Therefore, we define two reference points A_R and B_R with a constant value $\mathcal{K}$ cf. Equation 4.1.

$$\begin{aligned} A_R &= X_C, Y_C + \mathcal{K} \\ B_R &= X_C + \mathcal{K}, Y_C \end{aligned} \tag{4.1}$$

Moreover, we define two points A and B in Equation 4.2 that are located at the edge of the area.

$$\begin{aligned} A &= X_C, Y_C + \lambda \\ B &= X_C + \mu, Y_C \end{aligned} \tag{4.2}$$

In order to calculate the differences μ and λ, we determine the distance between the camera's position and the position of the reference points A_R, B_R as described in Equation 4.3.

$$\begin{aligned} dX &= f_{distance}(C, A_R) \\ dY &= f_{distance}(C, B_R) \end{aligned} \tag{4.3}$$

As depicted in Figure 4.3, the distance between the camera's position X_C, Y_C and the reference point X_B+$\mathcal{K}, Y_B$ as well as the radius r allow the determination of the difference μ between the camera's latitude X_C and the resource's latitude X_B as described in Equation 4.4. Moreover, the calculation of the difference λ is described in Equation 4.6.

$$\begin{aligned} \frac{r}{dY} &= \frac{\mathcal{K}}{\mu} \\ \mu &= \frac{\mathcal{K} * dY}{r} \end{aligned} \tag{4.4}$$

$$\begin{aligned} \frac{r}{dX} &= \frac{\mathcal{K}}{\lambda} \\ \lambda &= \frac{\mathcal{K} * dX}{r} \end{aligned} \tag{4.5}$$

Once we determined the maximum differences μ, λ, we can compose a SPARQL query for retrieving all resources within a square as described in Listing 4.1. In order to retrieve all resources within the predefined circle, the distances between the camera's position and all queried resources X_P, Y_P have to be calculated as illustrated in Section 4.4.2. If

$$f_{distance}(C, P) \leq r \tag{4.6}$$

the resources are used in the mathematical model for calculating the screen coordinates cf. Section 4.4.

4.3 Consolidation of Linked Open Data Sources

Once, RDF resources from different LOD sources are queried, the resources have to be consolidated, as described in Chapter 4, Figure 4.1. During this step, identical resources are detected, merged and identical information is removed. Therefore, links between resources have to be discovered. For instance, Listings 4.2 and 4.3 depict an example of a resource from two different LOD sources. Because both examples describe the same resource, we will merge the resources and remove identical information.

```
<rdf:RDF>
   <rdf:Description rdf:about="http://dbpedia.org/resource/Austrian_National_Library">
      <rdfs:label xml:lang="en">Austrian National Library</rdfs:label>
      <geo:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#float"> 48.2061
      </geo:lat>
      <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#float"> 16.3664
      </geo:long>
   </rdf:Description>
</rdf:RDF>
```

Listing 4.2: Example of an RDF resource from DBpedia

```
<rdf:RDF>
   <rdf:Description
   rdf:about="http://linkedgeodata.org/data/triplify/node76519314?output=xml">
      <rdfs:label>RDF description of Österreichische Nationalbibliothek
      </rdfs:label>
      <foaf:primaryTopic>
         <spatial:Feature rdf:about="http://linkedgeodata.org/triplify/node76519314">
            <geo:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
            48.2055458e0</geo:lat>
            <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">
            16.3652966e0</geo:long>
            <rdfs:label>Österreichische Nationalbibliothek</rdfs:label>
         </spatial:Feature>
      </foaf:primaryTopic>
   </rdf:Description>
</rdf:RDF>
```

Listing 4.3: Example of an RDF resource from LinkedGeoData

In our approach, we propose the integration of a link discovery framework such as Silk [59]. Silk is a framework for finding relationships between entities within different data sources and it supports the generation of RDF links among different sources such as `owl:sameAs` links. Therefore, the Silk - Link Specification Language (Silk-LSL) is used for specifying the type of RDF links to query for as well as different conditions which entities must fulfill. Moreover, Silk uses the SPARQL protocol for accessing data sources. In order to discover links between data sources, the framework integrates a number of similarity metrics such as the Jaro distance metric for string similarity.

Once, the resources are consolidated, they can be processed in the mathematical model for calculating screen coordinates.

4.4 Mathematical Model for Calculating Screen Coordinates

In this section, the mathematical model for determining the screen coordinates is described. First, we give an overview about the processing steps for converting the world coordinates of the resources to screen coordinates and define the different coordinates resulting from each step. Moreover, a detailed description of each processing step is provided in the subsections.

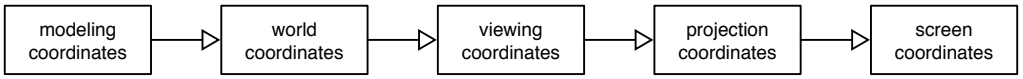

FIGURE 4.4: Processing steps for converting modeling coordinates to screen coordinates

Generally, the mathematical model realizes the processing steps for converting the world coordinates of the resources to final screen coordinates, as illustrated in Figure 4.4. The authors in [47] define the different types of coordinates. First, the individual modeling coordinates cf. Definition 4.4 of the resources have to be transformed to world coordinates cf. Definition 4.5. Since the geographical positions of the resources are already defined in world coordinates we do not transform the coordinates of the resources.

Definition 4.4. [Modeling coordinates] *Each individual resource has its own separate coordinate reference frame. Thus, modeling coordinates describe the individual coordinates of an object. The coordinates of an object do not correlate with coordinates of other objects.*

Definition 4.5. [World coordinates] *In order to construct a scene, the separate modeling coordinates are converted to world coordinates. The world coordinates allow to place the resources within a scene. Consequently, the different resources are correlated to each other.*

Next, the world coordinates are transformed to viewing coordinates cf. Definition 4.6. Thereby, it is considered that the world space including all resources is viewed by a camera. In order to get the viewing coordinates, we need to set the coordinates of the resources in relation to the camera's coordinates which will be described in Section 4.4.3. Afterwards, the points have to be rotated considering the orientation of the camera. The determination of the camera's orientation is described in Section 4.4.4 and the rotation around each axis will be described in Section 4.4.5.

Definition 4.6. [Viewing coordinates] *The viewing coordinates consider the position of the camera. In order to convert the world coordinates to viewing coordinates, the coordinates have to be transformed and rotated according to the camera's orientation.*

After that, a projection transformation for getting the projection coordinates cf. Definition 4.7 is performed. In Section 4.4.6, the realization of a perspective projection is discussed.

Definition 4.7. [Projection coordinates] *By means of the viewing coordinates, the coordinates can be converted to projection coordinates. Therefore, the coordinates are projected onto a projection plane, e.g., by a perspective projection.*

Finally, these coordinates have to be transformed to device coordinates cf. Definition 4.8. This transformation is described detailed in Section 4.4.7.

Definition 4.8. [Screen coordinates] *The final screen coordinates are the actual pixel coordinates of the screen. In order to represent a resource in the camera interface, the projection coordinates are transformed to screen coordinates.*

In order to realize the transformation of the coordinates, the distance as well as the bearing between the camera's position and the position of each resource has to be calculated.

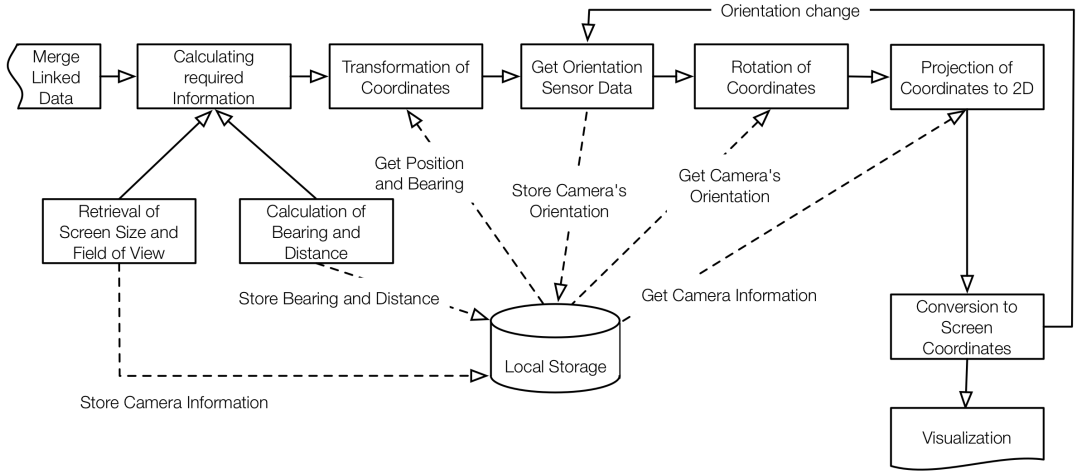

FIGURE 4.5: Conceptual overview of the mathematical model

In Figure 4.5, we provide a conceptual overview of the mathematical model: first, necessary information such as bearing and distance between the camera's position and each resource are calculated. Therefore, we describe the determination of the bearing in degrees and the distance in meters in Section 4.4.1 and 4.4.2. Moreover, parameters of the camera like the screen size and the horizontal and vertical field of view are determined. After storing these parameters, the resources are transformed according to the camera's

position. Once the coordinates are transformed, orientation data (azimuth, pitch, and roll) of the mobile device's sensors are retrieved as described in Section 4.4.4. According to this orientation data, the coordinates are rotated around each axis. Afterwards, the coordinates are projected to 2D coordinates. Finally, the conversion to screen coordinates allow to display the coordinates in a camera's interface. Once the orientation of the device changes, orientation sensor data of the mobile device is requested and the coordinates are converted again for getting accurate screen coordinates.

4.4.1 Calculation of Bearing between two Geographic Coordinates

Definition 4.9. [Bearing] *Bearing is defined as the angle between two points with reference to the North direction. As illustrated in Figure 4.6, the arms of the angle are an imaginary line from one point to the North and a line to the other point. Bearing is the compass direction from one position to another position.*

FIGURE 4.6: Bearing between two points

In order to calculate the bearing between the user's location and a specific resource, Equation 4.7 [60] is applied. The formula considers the coordinates of the camera's position X_C, Y_C as well as the coordinates of the resource's position X_P, Y_P in decimal degrees.

$$\begin{aligned}\beta =& \text{atan2}(\sin(\Delta(Y_C, Y_P)) * \cos(X_P), \cos(X_C) * \sin(X_P) \\ & - \sin(X_C) * \cos(X_P) * \cos(\Delta(Y_C, Y_P)))\end{aligned} \tag{4.7}$$

The function atan2 returns a value in the range of -180° and +180°. In order to get a value in the compass range of 0° to 360°, the result is normalized as described in Equation 4.8.

$$\beta_{norm} = (b + 360) \bmod 360 \tag{4.8}$$

Basically, the calculation of the bearing is required in the mathematical model for converting the coordinates of the resources relative to the coordinates of the user's position as described in Section 4.4.3. In order to perform a perspective projection and calculate the screen coordinates of each resource, we set the resources in relation to the camera's position. Moreover, the visualization of the off-screen objects in the minimap requires the calculation of the bearing. The visualization of the resources in a minimap is discussed in Section 4.5. Besides the determination of the bearing, the calculation of the distance between two points is required in the mathematical model. Therefore, we discuss the determination of the distance in Chapter 4.4.2.

4.4.2 Calculation of the Distance between two Geographic Coordinates

We use the "haversine" formula [61], depicted in Equation 4.9, for calculating the distance between two geographic points. Each geographic point consists of a latitude, longitude, and altitude value. The objects that are displayed in the camera's field of view as well as the camera position are represented by geographic points. The Formula 4.9 requires the latitude and longitude values of two geographical points and it contains two variables a and c for caching the intermediate result. Since the distance is calculated in kilometers, it has to be converted into meters scale.

$$\begin{aligned} a &= \sin^2(\frac{\Delta(X_C, X_P)}{2}) + \cos(X_C) * \cos(X_P) * \sin^2(\frac{\Delta(Y_C, Y_P)}{2}) \\ c &= 2 * \text{atan2}(\sqrt{a}, \sqrt{1-a}) \\ d &= \mathcal{R} * c \end{aligned} \tag{4.9}$$

The calculation of the distance between two geographic coordinates is particularly required for the transformation of the resources' coordinates relative to the camera's position cf. Section 4.4.3. Moreover, the representation of the resources in the minimap requires the calculation of the distance between the camera's position and the position of the resources.

4.4.3 Transformation of Coordinates relative to the Camera's Position

In order to calculate the viewing coordinates, the world coordinates have to be transformed and subsequently, rotated which is described in Section 4.4.5. We consider the camera as the central point of a coordinate system in which

$$X_C = 0 \quad Y_C = 0 \quad Z_C = 0$$

and recalculate the coordinates $P^{trans}(x, y, z)$ for each resource. The coordinates are determined relative to the position of the camera.

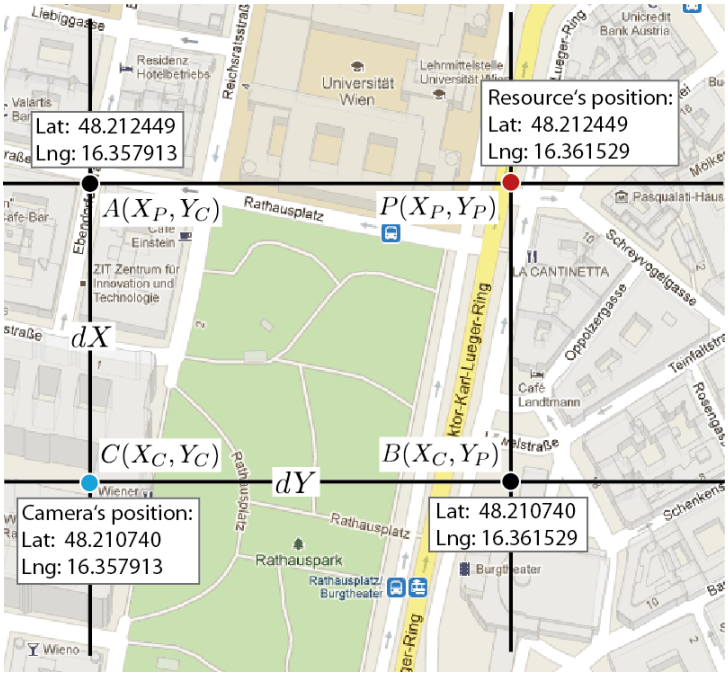

FIGURE 4.7: Converting the resources' coordinates

In Figure 4.7, we illustrate the calculation of X_P^{trans} and Y_P^{trans}. The figure depicts four points with geographic coordinates: The camera's position $C(X_C, Y_C)$, the resource's position $P(X_P, Y_P)$, and two points placed where the degree of latitude crosses the degree of longitude $A(X_P, Y_C)$ and $B(X_C, Y_P)$. In Equation 4.10, the coordinates of A and B are defined.

$$\begin{aligned} X_A &= X_P \quad Y_A = Y_C \\ X_B &= X_C \quad Y_B = Y_P \end{aligned} \tag{4.10}$$

In order to transform the coordinates of a resource relative to the camera's position, we define new coordinates for a resource as depicted in Equation 4.11.

$$\begin{aligned} X_P^{trans} &= (+/-)dX \\ Y_P^{trans} &= (+/-)dY \\ Z_P^{trans} &= dZ \end{aligned} \tag{4.11}$$

The symbols dX and dY are variables for defining an intermedia result and can be determined by calculating the distance between the camera C and the defined points A and B as described in Equation 4.12. Moreover, dZ describes the height difference between

the camera's position and the resource's position. The function $f_{distance}$ calculates the distance between two points as described in Equation 4.9.

$$\begin{aligned} dX &= f_{distance}(C, A) \\ dY &= f_{distance}(C, D) \\ dZ &= Z_P - Z_C \end{aligned} \tag{4.12}$$

Finally, we determine in which direction the resources are located by calculating the bearing. Afterwards, the values dX, dY, and dZ have to be inverted according to the determined bearing. Figure 4.8 shows the modification of these values.

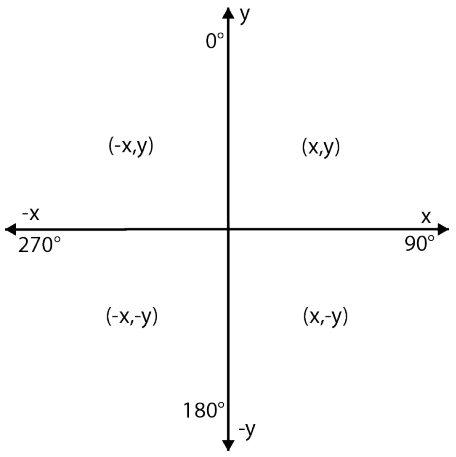

FIGURE 4.8: Converting coordinates of the resources relative to bearing

We change the signs according to the following algorithm:

Data: bearing β, distances dX,dY,dZ
Result: transformed world coordinates $X_P^{trans}, Y_P^{trans}, Z_P^{trans}$

if $0 \leq \beta \leq 90$ **then**
  $X_P^{trans} \leftarrow dX$;
  $Y_P^{trans} \leftarrow dY$;
  $Z_P^{trans} \leftarrow dZ$;
else if $90 \leq \beta \leq 180$ **then**
  $X_P^{trans} \leftarrow -dX$;
  $Y_P^{trans} \leftarrow dY$;
  $Z_P^{trans} \leftarrow dZ$;
else if $180 \leq \beta \leq 270$ **then**
  $X_P^{trans} \leftarrow -dX$;
  $Y_P^{trans} \leftarrow -dY$;
  $Z_P^{trans} \leftarrow dZ$;
else if $270 \leq \beta \leq 360$ **then**
  $X_P^{trans} \leftarrow dX$;
  $Y_P^{trans} \leftarrow -dY$;
  $Z_P^{trans} \leftarrow dZ$;
end

Algorithm 4.1: Changing signs of coordinates

After changing the signs of the coordinates we obtain the point $P^{trans}(x, y, z)$ that can be used in the mathematical model for calculating the screen coordinates. The transformation of the coordinates according to the camera's position is the first part of transforming world coordinates into viewing coordinates. In order to get viewing coordinates, the coordinates have to be rotated around each axis according to the orientation of the camera which is described in Section 4.4.5. Before applying the rotation around each axis, the orientation of the camera has to be determined, as described in Section 4.4.4.

4.4.4 Sensor Data - Determination of the Camera's Orientation

Applying orientation data in the mathematical model for calculating screen coordinates requires the orientation of the camera in azimuth, pitch, and roll cf. Definition 4.10, 4.11, and 4.12. We suppose that the coordinate system is right handed as illustrated in Figure 2.10 from Section 2.4.2. Moreover, we use the definition of the coordinate system as depicted in Figure 4.9.

Definition 4.10. [Azimuth] *Azimuth depicts the rotation around the z-axis. The z-axis refers to the vertical axis of the screen and it points towards the top. The value is within*

a range of 0° to 360° where 0° points to the North, 90° to the East, 180° to the South and 270° to the West.

Definition 4.11. [Pitch] *Pitch indicates the rotation around the x-axis. The x-axis refers to the horizontal axis of the screen and it points to the right. The value ranges between -180° and 180°.*

Definition 4.12. [Roll] *Roll indicates the rotation around the y-axis. The y-axis points towards the user when she looks at the top of the screen. The value ranges between -90° and 90°.*

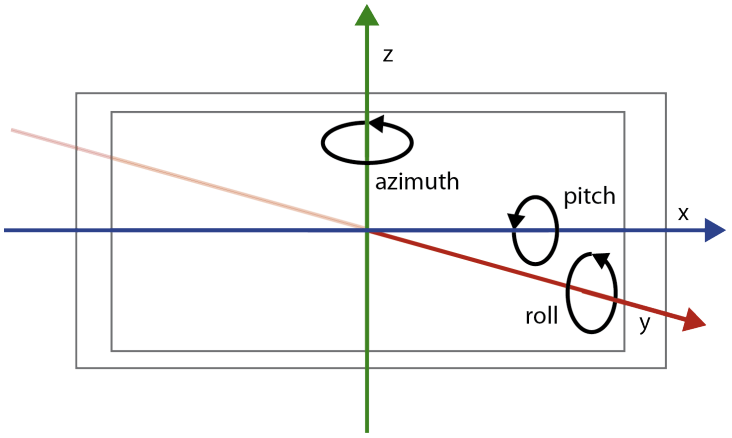

FIGURE 4.9: Definition of a device's coordinate system and orientation

In order to calculate azimuth, pitch, and roll we suppose that the mobile device provides appropriate orientation sensors. As described in Chapter 2.2.2, mobile devices integrate multiple sensors that allow the determination of the device's orientation. In order to use the data received from mobile devices' sensors in our mathematical model the following requirements have to be considered:

Acquisition of sensor data Our approach relies on accurate sensor data. Therefore, an appropriate sensor that returns accurate and stable information about the camera's orientation is required.

Remapping of coordinate system If the screen is used in landscape mode, the values returned by the sensors can eventually still be in portrait mode. In this case, the axes of the coordinate system have to be remapped.

Angle conversion If the angles are in degrees, they have to be converted to radians in order to process the data correctly in the mathematical model.

Value range Before processing the values, the angles have to be adapted to fit in the range specified in the Definitions 4.10, 4.11, and 4.12.

Once the values are in the specified range and in degrees, they can be processed by the mathematical model for calculating the 2D screen coordinates. The orientation of the camera is used for rotating the coordinates around each axis by utilizing Formula 4.13 as described in Section 4.4.5. This rotation is part of the transformation from world coordinates to viewing coordinates.

4.4.5 Rotation of Coordinates relative to the Camera's Orientation

After transforming the resources' coordinates relative to the camera's position as described in the previous Section 4.4.3, the coordinates of the resources are rotated relative to the camera's orientation. In this section, we describe the rotation of the transformed world coordinates in order to obtain the viewing coordinates of the resources. Basically, the rotation depends on the angles of the camera's orientation X_Θ, Y_Θ, and Z_Θ. The calculation of the camera's orientation (azimuth, pitch, and roll) is described detailed in Section 4.4.4. Moreover, a definition of azimuth, pitch, and roll is provided in Section 4.4.4. In order to calculate the viewing coordinates X_P^{rot}, Y_P^{rot}, and Z_P^{rot}, the coordinates are rotated around each axis as shown in Equation 4.13.

$$\begin{bmatrix} X_P^{rot} \\ Y_P^{rot} \\ Z_P^{rot} \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(X_\Theta) & -\sin(X_\Theta) \\ 0 & \sin(X_\Theta) & \cos(X_\Theta) \end{pmatrix} * \begin{pmatrix} \cos(Y_\Theta) & 0 & \sin(Y_\Theta) \\ 0 & 1 & 0 \\ -\sin(Y_\Theta) & 0 & \cos(Y_\Theta) \end{pmatrix} \tag{4.13}$$
$$* \begin{pmatrix} \cos(Z_\Theta) & -\sin(Z_\Theta) & 0 \\ \sin(Z_\Theta) & \cos(Z_\Theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} * \left(\begin{bmatrix} X_P^{trans} \\ Y_P^{trans} \\ Z_P^{trans} \end{bmatrix} - \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} \right)$$

After determining the viewing coordinates of the resources, the projection coordinates are calculated by means of a perspective projection as described in the next Section 4.4.6.

4.4.6 Perspective Projection - Transforming Viewing Coordinates to Projection Coordinates

After rotating the coordinates, the viewing coordinates of the resources are obtained. Next, the coordinates are transformed to projection coordinates. Therefore, we project the transformed point on a 2D screen. In Figure 4.10, we illustrate the computation of the screen coordinates. First, we check if the resource is located in front of the camera or behind the camera by considering Z_P^{rot}. If

$$Z_P^{rot} < 0$$

the resource is in the field of view. Otherwise, if

$$Z_P^{rot} > 0$$

the resource is not situated in the field of view.

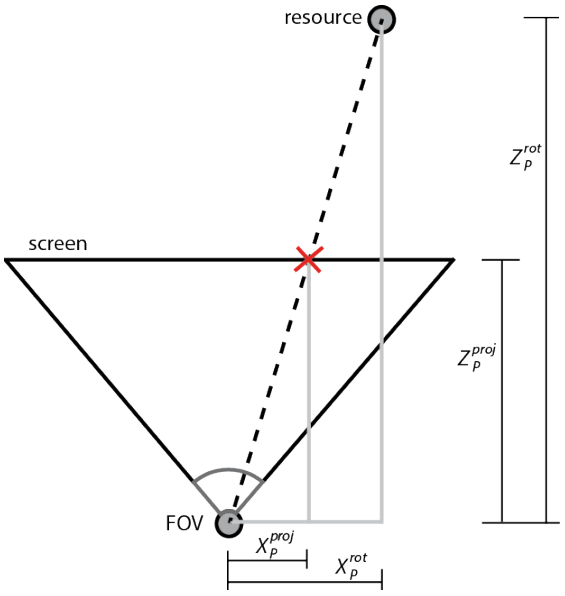

FIGURE 4.10: Perspective projection - Transformation to screen coordinates

After that, we calculate the projection coordinates (X_P^{proj}, Y_P^{proj}) by applying the similar triangles rule. In Figure 4.10, we only depicted the computation of the x-coordinate X_P^{proj}. The y-coordinate Y_P^{proj} can be calculated in a similar way. In order to compute X_P^{proj}, we derive a similar triangles rule as described in Equation 4.14. The computation of Y_P^{proj} is depicted in Equation 4.15.

$$\begin{aligned} \frac{X_P^{proj}}{Z_P^{proj}} &= \frac{X_P^{rot}}{Z_P^{rot}} \\ X_P^{proj} &= \frac{X_P^{rot} * Z_P^{proj}}{Z_P^{rot}} \end{aligned} \tag{4.14}$$

$$\begin{aligned} \frac{Y_P^{proj}}{Z_P^{proj}} &= \frac{Y_P^{rot}}{Z_P^{rot}} \\ Y_P^{proj} &= \frac{Y_P^{rot} * Z_P^{proj}}{Z_P^{rot}} \end{aligned} \tag{4.15}$$

In order to calculate X_P^{proj} and X_P^{proj}, we have to determine a value for Z_P^{proj}. As shown in Figure 4.11, the value Z_P^{proj} is directly related to the field of view. In order to calculate the distance between the camera center and the image plane, we consider a triangle that is highlighted in Figure 4.11 and apply a trigonometric function. By considering the trigonometric function

$$\tan(\alpha) = \frac{opposite}{adjacent} \tag{4.16}$$

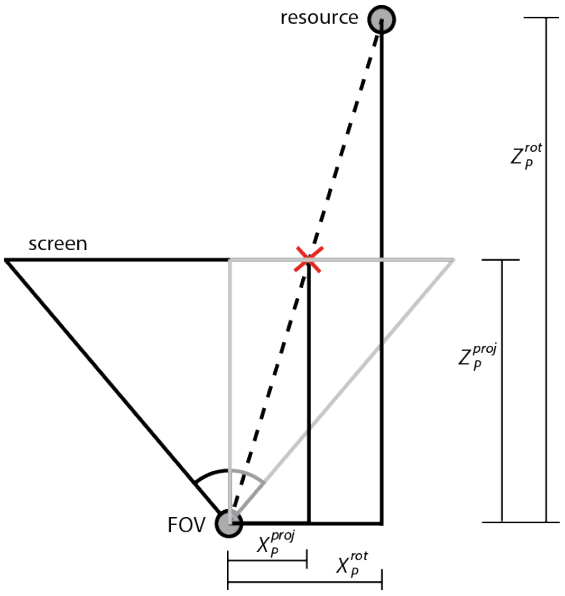

FIGURE 4.11: Calculation of the distance between camera center and image plane

we only need to calculate the half of the screen width as well as the half of the horizontal field of view for retrieving the distance. For determining the adjacent side, we convert the equation to

$$adjacent = \frac{opposite}{\tan(\alpha)} \tag{4.17}$$

Next, the value of Z_P^{proj} can be calculated by setting the horizontal field of view and the screen width.

$$Z_P^{proj} = \frac{w}{2 * \tan(\frac{hFOV}{2})} \tag{4.18}$$

After determining the value for Z_P^{proj}, the projection coordinates are computed by using Equation 4.15. In the next Section 4.4.7, the process of converting projection coordinates of a point into screen coordinates is described detailed.

4.4.7 Converting Projection Coordinates to Screen Coordinates

In this section, the transformation of projection coordinates to screen coordinates is described. In Figure 4.12, we illustrate the difference between projection and screen coordinates; the projection coordinates are positioned in the middle of the camera's field of view. However, screen coordinates are defined by setting the center to the upper left corner and they have to be specified in pixels. The screen coordinates can be calculated by applying the Equations 4.19 and 4.20.

$$X_P^{screen} = X_P^{proj} * ppm + \frac{w}{2} \tag{4.19}$$

$$Y_P^{screen} = Y_P^{proj} * ppm - \frac{h}{2} \tag{4.20}$$

First, the projection coordinates X_P^{proj} and Y_P^{proj} are multiplied by pixels per meter (ppm). Afterwards, half of the screen width has to be added to X_P^{proj} and half of the screen height has to be subtracted from Y_P^{proj}. Finally, the screen coordinates $P^{screen}(x, y)$ can be displayed in the camera's interface.

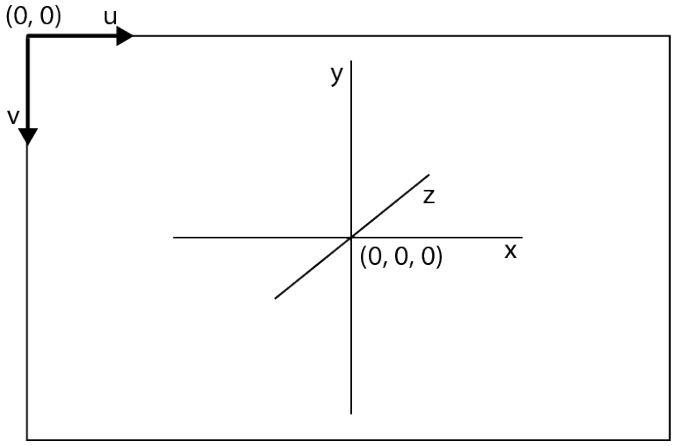

FIGURE 4.12: Transformation from projection coordinates to device coordinates

4.5 Visualization of Off-Screen Objects in Mobile Augmented Reality

Mobile AR applications such as Wikitude or Layar include a minimap for providing information about the nearby resources. By means of this map, the user gets an overview about the positions of the nearby resources. In [62] the authors analyze the visualization techniques of off-screen objects in mobile AR applications and develop an alternative visualization technique based on arrows. In this Section, we present an approach for realizing a minimap in an AR application. The minimap bases on the principle illustrated in Figure 4.13.

Generally, the minimap consists of two circles; one illustrates the center of the minimap M and the other depicts the circle representing the radius of the visible area. Additionally, a triangle that realizes the horizontal field of view is visualized. In order to illustrate the triangle, two lines are depicted; one ranges from the center of the minimap to a point that exists on the circle with an angle of

$$\beta = -\frac{hFOV}{2} \tag{4.21}$$

and the other ranges from the center to a point with the angle of

$$\beta = \frac{hFOV}{2} \tag{4.22}$$

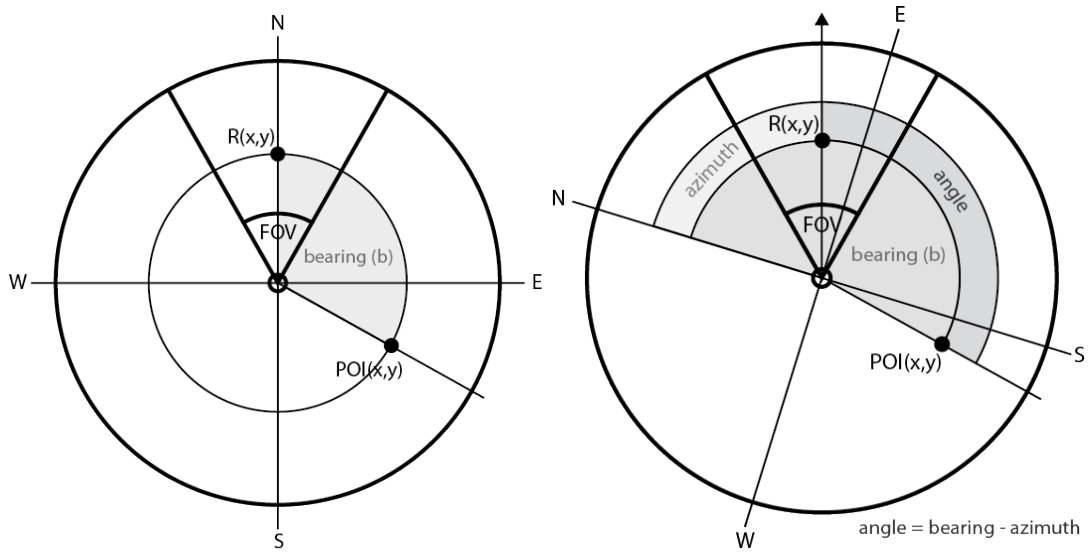

FIGURE 4.13: Visualizing off-screen objects in a minimap

In order to illustrate a resource in the minimap, the distance of the resources to the user has to be determined and set in relation to the radius of the visible area. For instance, if a resource has a distance of 50 meters and the total radius is 100 meters an invisible circle with a radius half of the total radius is drawn. Next, the resource has to be illustrated at a point on this circle. Therefore, we define a reference point R on the circle. A point on a circle can be determined by Equation 4.23. If a point on the circle is defined as R and the rotation angle is α, then the new point $P^{map}(x, y)$ is defined as:

$$\begin{aligned} X_P^{map} &= (X_R - X_M) * \cos(\alpha) - (Y_R - Y_M) * \sin(\alpha) + X_M \\ Y_P^{map} &= (X_R - X_M) * \sin(\alpha) + (Y_R - Y_M) * \cos(\alpha) + Y_M \end{aligned} \tag{4.23}$$

In order to calculate the coordinates of a point in the minimap X_P^{map}, Y_P^{map} as depicted in Equation 4.23, a reference point R on the circle as well as an α value that functions as angle between the reference point on the circle and the resource displayed in the minimap has to be determined. The reference point R can be calculated by using the x-coordinate of the circle's center. The y-coordinate can be calculated by using the distance of the resource and set it in relation to the compass of the mini-map.

In Figure 4.14 we describe the calculation of the reference point's screen coordinates. Point R is the reference point whose coordinates are calculated. Therefore, we use the following formulas:

$$\begin{aligned} X_R &= X_M \\ Y_R &= \frac{-d * (Y_N - Y_M)}{-r} + Y_M \end{aligned} \tag{4.24}$$

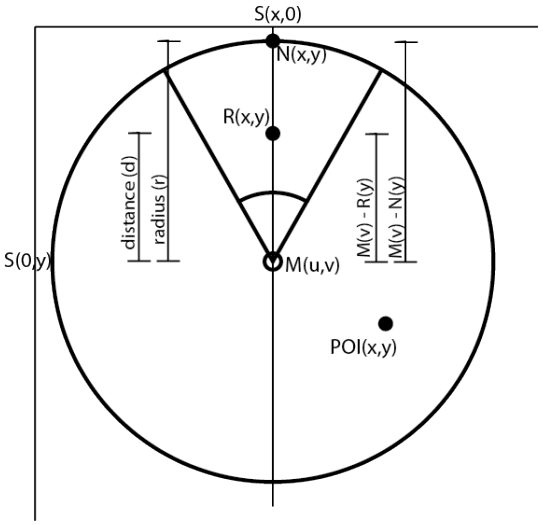

FIGURE 4.14: Calculation of distances in the minimap

The formula for Y_R can be derived the following way:

$$\begin{aligned} \frac{Y_M - Y_R}{Y_M - Y_N} &= \frac{d}{r} \\ Y_M - Y_R &= \frac{d * (Y_M - Y_N)}{r} \\ -Y_R &= \frac{d * (Y_M - Y_N)}{r} - Y_M \\ Y_R &= \frac{-d * (Y_N - Y_M)}{-r} + Y_M \end{aligned} \tag{4.25}$$

After calculating the coordinates of the reference point R the angle α has to be determined, as depicted in Figure 4.13. The left figure illustrates that if the camera is pointing towards the North, the angle will be exact the bearing value. The right figure shows that if the camera is not pointing towards north, the azimuth value represents the difference between the bearing and the direction the camera is pointing at. Thus, the angle is computed by

$$\alpha = b - \text{azimuth} \tag{4.26}$$

After determining the screen coordinates of the reference point X_R, Y_R as well as the angle α, the screen coordinates of a resource in the minimap X_P^{map}, Y_P^{map} are calculated by using Equation 4.23.

4.6 Summary

In this chapter, we focused on our approach for displaying resources from LOD sources in a camera's field of view and accessing LOD sources for getting information about the resources. First, we provided an overview about our conceptional model in Figure 4.1.

Our approach basically, includes the acquisition of sensor data such as the camera's position or orientation. The position is required for querying resources that are located in the camera's environment and the orientation is used in the mathematical model for calculating viewing coordinates.

In Section 4.2, we described a process for querying LOD sources in order to retrieve all resources that are located within a predefined radius around the camera's position by using SPARQL queries. After querying resources from different LOD sources, identical resources from different LOD sources are consolidated, as described in Section 4.3. Thereby, we propose to use a framework such as SILK that allows to determine links between resources.

Next, we describe our mathematical model for calculating screen coordinates in Section 4.4. First, we gave an overview of the basic calculations of bearing and distance between two geographical coordinates. We use the distance as well as the bearing between the camera's position and a specific resource for determining the screen coordinates and for visualizing the resources in the mini-map. Next, we describe the transformation from world coordinates to actual device coordinates that can be displayed in the camera's field of view. First, each object has its own modeling coordinates that do not correlate with coordinates from other objects. Since we have the geographic position of each resource, the coordinates are already in world coordinates. Next, we calculate the viewing coordinates because the objects are viewed by a camera. Therefore, we have to set the camera's position in relation to the resources' positions as described in Chapter 4.4.3. Afterwards, the coordinates have to be rotated around each axis in order to consider the orientation of the camera. The viewing coordinates of each resource are determined after transforming and rotating the coordinates. Next, a perspective projection is applied in order to calculate the projection coordinates. In Section 4.4.6, we summarized the steps of a perspective projection. Afterwards, the projection coordinates have to be transformed to device coordinates as illustrated in Figure 4.12. The device coordinates of the resources can be displayed in the camera interface.

Additionally, we presented an approach for visualizing off-screen objects in a mini-map in order to improve usability. In Chapter 4.5, all necessary calculations are defined.

Chapter 5

System Architecture and Implementation

This chapter is structured into two basic parts: first, we provide and overview of our system architecture and in the second part we explain the implementation of the mathematical model for calculating the screen coordinates in more detail as well as the implementation of additional features.

The first part of this chapter focuses on the system architecture of our framework. Therefore, we define several design goals that are realized by the system. In Section 5.2, we give an overview of the conceptional system architecture. In Section 5.3, we provide a system overview with focus on the Android architecture and the basic activities integrated in our AR framework. After providing an overview of the system architecture, we explain the implementation of the mathematical model more detailed in Section 5.4.4. Thereby, the focus is on getting the orientation of the camera, obtaining the position of the camera and the resources in the surroundings, accessing LOD databases in order to get further information about the resources, and transforming the coordinates to screen coordinates. Additionally, we explain the implementation of the minimap. In the last part of this chapter, we describe some additional features that are implemented for improving usability such as the visualization of data in a Google Maps View or the integration of the Mobile RDFBrowsing Interface.

5.1 Design Goals

In the following, we define several design goals that are realized by the framework.

Extensibility The framework is extensible. Users have the possibility to add new components and capabilities. Thus, further features with only slight modifications can be included. For instance, the framework can be adjusted by adding more LOD sources cf. Figure 5.3.

Accurate representation In order to represent the resources as accurate as possible in the user interface we consider the following aspects. An accurate representation of the resources can be achieved by providing a correct position of the user and the resources as well as an exact orientation of the camera. In order to check the GPS accuracy, we allow the user to view the positions on a map. Moreover, the calculation of the screen coordinates has to be accurate. Therefore, we have to consider the accurateness of the data types used within the calculations. For instance, we use the data types `float` and `double` for determining exact latitude and longitude values of a GPS position.

Completeness Available information about the resources shall be retrievable by the user. Therefore, we have to consider the fact that some LOD resources are incomplete. We use an external web service to retrieve information that sometimes can not be queried from an LOD source such as the height above the sea level. Moreover, the user has the possibility to query background information about the resources. Therefore, we integrate an RDF Browsing Interface.

Usability The user interface shall be usable for experienced as well as unexperienced users. Particularly, the representation of the resources in an AR interface serve as an orientation guide and it allows to get a better overview of the position of the resources than in a conventional 2D map. Therefore, the resources have to be clearly recognizable in the camera's field of view. Moreover, we implement a minimap in order to serve as an additional orientation guide.

5.2 Conceptional Components

After presenting the design goals we give an overview of the conceptional components of our system architecture. Therefore, the components are summarized in Figure 5.1. In this section, we will describe all basic components of our system architecture. Generally, the AR framework integrates different components for getting sensor data or transforming coordinates and a component that accesses different SPARQL endpoints for querying resources from LOD sources. In Figure 5.1, we illustrate only one SPARQL endpoint although, any number of endpoints can be added.

Start Interface This component represents an Android activity, as described in the following Section 5.3. Basically, it provides data for representing LOD resources such as the radius to query and which LOD sources have to be queried. Moreover, it references the LocationManager for getting a current position of the camera. The `Camera Interface` uses this data for querying and representing the resources.

Camera Interface The `Camera Interface` builds one of the main components since it implements the camera view and represents the resources in the camera's field of

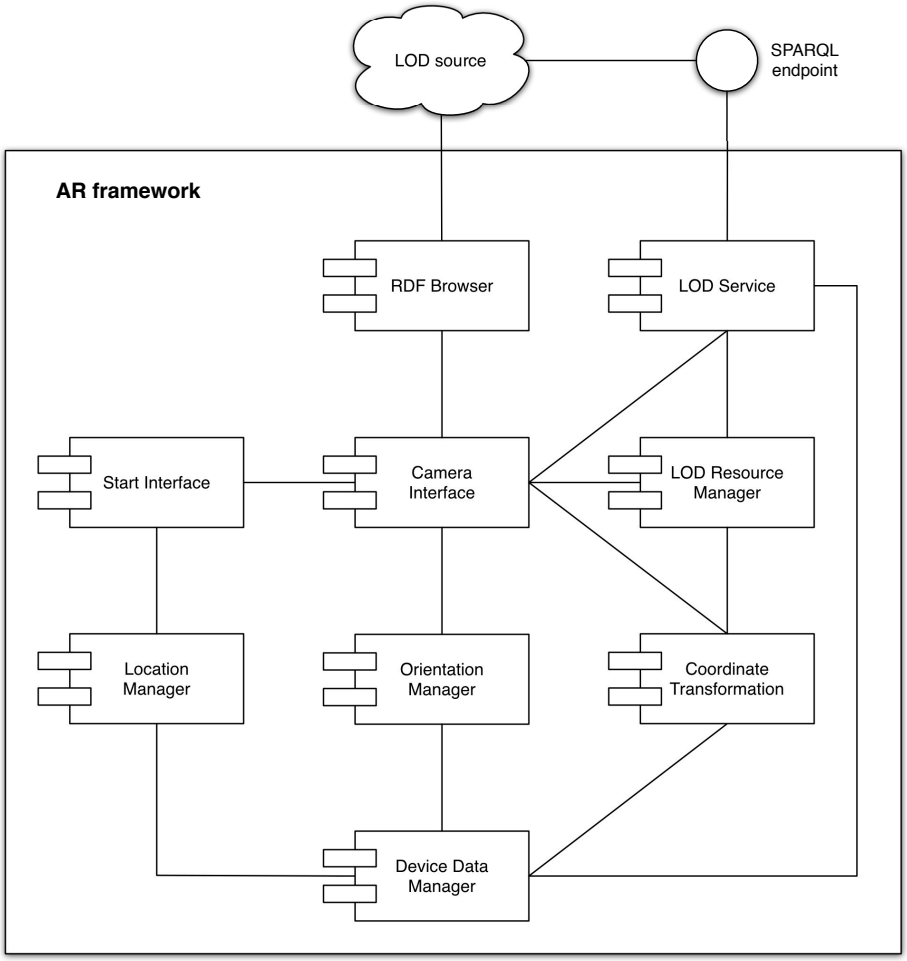

FIGURE 5.1: Conceptional system architecture of the AR framework

view. Therefore, it has to reference the `Orientation Manager` for getting the camera's orientation. Moreover, it calls the `LOD Service` for querying all resources nearby the camera's position. Once all resources are queried, the component `Coordinate Transformation` for getting screen coordinates of the resources is called. Afterwards, the `Camera Interface` represents the resources in the camera's live view.

Location Manager The `Location Manager` is responsible for getting the current position of the camera. The `Location Manager` is registered during the start of the AR framework and continuously updates the position. It provides the geographic coordinates (latitude and longitude) in decimal degrees and the altitude in meters.

Orientation Manager The `Orientation Manager` provides information about the camera's orientation. The orientation is expressed in azimuth, pitch, and roll as described in Section 4.4.4. The `Orientation Manager` continuously tracks the camera's orientation and provides the current orientation in degrees.

Device Data Manager This component is responsible for storing all required information for calculating the screen coordinates and representing the resources in the camera's field of view. Therefore, it stores the position, orientation, and parameters (e.g. horizontal and vertical field of view, display height and width in pixels, or compass radius) of the camera. This information is used by the component `Coordinate Transformation` for calculating the screen coordinates and the `Camera Interface` for displaying the resources.

LOD Service Another component is the `LOD Service` that represents the interface for querying SPARQL endpoints in order to get resources located nearby the camera's position. This components allow the creation of SPARQL queries and query any number of LOD sources that provide a SPARQL endpoints. Thus, it can be extended by various LOD sources. Once the resources are queried, the `LOD Service` creates LOD resources and stores them by using the `LOD Resource Manager`.

LOD Resource Manager The component `LOD Resource Manager` is responsible for storing all relevant information about resources such as a unique identifier, geographic coordinates, screen coordinates, or additional information such as a summary, the type, or a picture link. This information is used primarily by the component `Camera Interface` for representing the resources in the camera's view.

Coordinate Transformation Basically, this component realizes the mathematical model for calculating the screen coordinates of the resources. It transforms geographic coordinates of the resources into screen coordinates that can be represented in the camera's field of view. Therefore, it uses information provided from the components `Device Data Manager` as well as `LOD Resource Manager`.

RDF Browser This component allows to query additional information about resources that are displayed in the camera's field of view. Therefore, it directly sends HTTP requests to LOD sources. The response is stored in RDF graphs that are represented in an interface. Generally, it provides a Browsing interface for searching for information about resources. The `RDF Browser` is described more detailed in Section 5.5.4.

5.3 System Overview

We decided to implement the framework on the Android platform. In Chapter 2.2.1 we discuss the differences between several development platforms. Since the Android platform is an open operating environment, we choose the Android platform for realizing the framework. First, we give an overview about the frameworks workflow. Because activities and views are fundamental components of the Android platform, first, we shortly explain them. Moreover, the visualization implies custom views for displaying resources or the minimap.

Definition 5.1. [View] *Views are user interface (UI) elements that form the basic building blocks of a user interface. Views are hierarchical and they know how to draw themselves. A view could be a button or a label or a text field, or lots of other UI elements. [...] (Hashimi et al. , 2010)*

Definition 5.2. [Activity] *An activity is a user interface concept. An activity usually represents a single screen in your application. It generally contains one or more views, but it doesn't have to. [...]. (Hashimi et al. , 2010)*

Definition 5.3. [Custom View] *Basically, a custom view [63] is a view cf. Definition 5.1 that applies unique features and it can be adapted individually.*

Generally, an activity provides a screen interface that allows users to interact with. One activity functions as main activity that is called once the application is launched. In our framework we implement the following four activities providing windows with a user interface:

StartActivity The activity `StartActivity` is called on the first launch of the framework. It is an abstract activity that can be extended by any activity for overriding methods and providing additional functionality. Basically, it provides a menu where the user can select the radius to query as well as the LOD source from which the data is queried. Moreover, the activity `StartActivity` implements the functionality for getting the current GPS position of the mobile device. Once an accurate position has been retrieved, the activity provides an interface for calling the activity `MapViewActivity` for displaying the position in a map as well as the activity `LiveViewActivity` for displaying the resources position in an AR interface.

MapViewActivity The activity `MapViewActivity` visualizes a 2D map for displaying the current position of the mobile device. Thereby, the user can assure herself that the GPS coordinates are accurate.

LiveViewActivity The activity `LiveViewActivity` implements a camera interface and provides a live preview. First, the resources `LinkedDataService` is called for querying the selected LOD sources. Afterwards, the screen coordinates of the resources are indicated in the preview in an AR interface. In order to display the resource's position in the camera interface, the orientation of the mobile device has to be determined. Moreover, the screen coordinates have to be calculated. Additionally, the activity allows to select resources displayed in the camera interface and shows information about it in a popup window. Once the popup window displaying information about the resource is selected, the activity `RDFBrowserActivity` can be called.

RDFBrowserActivity The activity `RDFBrowserActivity` can be called for querying additional RDF resources and extracting multimedia content from existing resources.

The interface displays information or images of the resources in a list and and it provides functionality for querying related resources.

In order to illustrate the workflow of our system, we stepwise summarize the activities and actions in Figure 5.2.

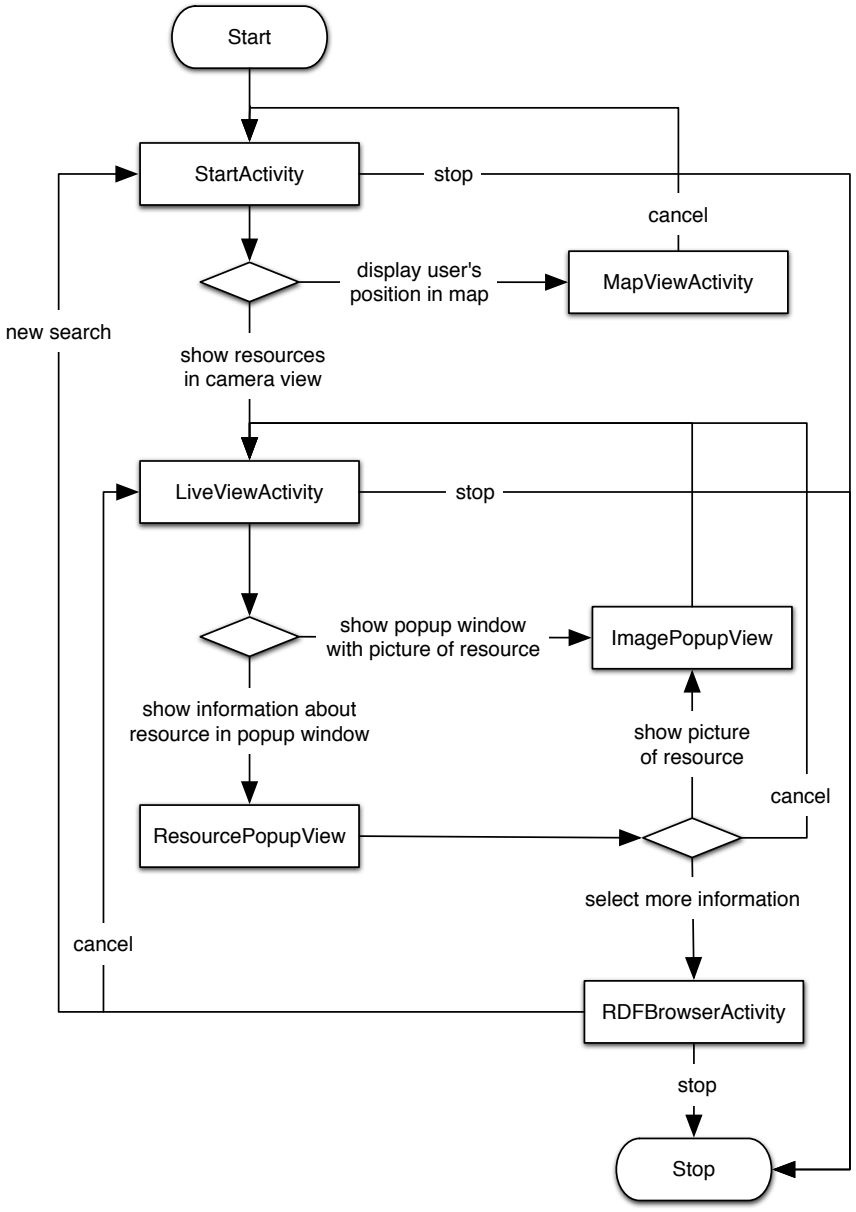

FIGURE 5.2: Workflow of the AR framework illustrating the activities interaction

5.4 Implementation of the Mathematical Model

The implementation of the mathematical model implies the access to the sensors of a mobile device and to LOD sources as well as the calculation of the resource's screen coordinates. First, we describe how to access the accelerometer and magnetic field sensors in order to get the orientation of a mobile device. After that, we explain how to obtain the geographical position of the camera as well as the positions of the resources located in the surroundings. Thereby, we focus on the access of LOD sources for obtaining information about resources. Now, we have all necessary information for calculating the screen positions of the resources. Finally, we describe the implementation of the minimap.

5.4.1 Getting the Orientation of the Camera

As described in Chapter 4.4.4, the orientation of the camera is indicated by the values azimuth cf. Definition 4.10, pitch cf. Definition 4.11, and roll cf. Definition 4.12. In order to calculate the orientation of the camera, we decided to use the following method provided by Android.

```
1 public static float[] getOrientation (float[] R, float[] values)
```

This method bases on a rotation matrix that can be computed by:

```
1 public static boolean getRotationMatrix (float[] R, float[] I, float[] gravity,
      float[] geomagnetic)
```

The method calculates the inclination matrix as well as the rotation matrix and it needs the gravity as well as the geomagnetic values as parameters. The gravity vector expressed in the devices's coordinates can be retrieved by the sensor `TYPE ACCELEROMETER`. The geomagnetic vector expressed in the device's coordinates can be retrieved by the sensor `TYPE MAGNETIC FIELD`. For that, first, we acquire a reference to the `SensorManager` that enables accessing the sensors of the device. Listing 5.1 shows how to access the accelerometer as well as the magnetic field sensor. Additionally, the registration of the `SensorListener` is presented.

```
1  public static SensorManager sM;
2  public static Sensor accel;
3  public static Sensor magnet;
4
5  sM = (SensorManager) getSystemService(SENSOR_SERVICE);
6  accel = sM.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
7  magnet = sM.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
8
9  sM.registerListener(this, accel, SensorManager.SENSOR_DELAY_NORMAL);
10 sM.registerListener(this, magnet, SensorManager.SENSOR_DELAY_NORMAL);
```

Listing 5.1: Accessing the senors with the `SensorManager`

Once the `SensorListener` is registered, the method `onSensorChanged()` is called. This method has to execute the following steps: first, the values of the accelerometer as well as the magnetic field sensor have to be determined. Afterwards, the rotation matrix can be calculated. Since we are using the screen in landscape mode the axes of the coordinate system have to be remapped after determining the rotation matrix. Once the coordinate system is remapped according to the screen orientation, the orientation values (azimuth, pitch, roll) can be obtained. Before storing the orientation values, the azimuth has to be adapted to fit in a range of 0 to 360 degrees. Moreover, the orientation values have to be converted from radians to degrees. Finally, the calculated orientation values can be used for calculating the screen coordinates of the resources.

5.4.2 Acquisition of Position Data

Android offers two different technologies for obtaining the user's position. Since Android offers to use the `GPS PROVIDER` as well as the `NETWORK PROVIDER` at the same time, we decided to implement both methods. Anyhow, usually the `NETWORK PROVIDER` does not provide that accurate data as the `GPS PROVIDER`. That is why, we recommend to use the application outdoors where a GPS connection is possible.

In order to determine the user's position we integrate the class `LocationManager` in our framework which provides access to the system location services. Listing 5.2 illustrates the access of location data. First, a reference to the `LocationManager` has to be acquired. After that, a listener for receiving location updates is registered. In doing so, we request location updates from the `NETWORK PROVIDER` as well as the `GPS PROVIDER` as frequently as possible. After that, we define the `onLocationChanged()` method for responding to location updates. Thus, we can request location information such as latitude, longitude, altitude and accuracy. The accuracy indicates the radius of an area in which the actual position can be. If the value of the accuracy is 5 the position of the user can be determined with an accuracy of 5 meters. Sometimes the altitude information can not be determined. Therefore, we use external web services such as the Google Elevation API cf. Section 5.5.2 or Earthtools cf. Section 5.5.3.

```
1  private LocationManager lManager = null;
2  lManager=(LocationManager)this.getSystemService(Context.LOCATION_SERVICE);
3  lManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,0,0,this);
4  lManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,0,0,this);
5  public void onLocationChanged(Location location) {
6     latitude = location.getLatitude();
7     longitude = location.getLongitude();
8     altitude = location.getAltitude();
9     accuracy = location.getAccuracy();
10    if(altitude == 0.0){  // call Google Elevation API and Earthtools
11       altitude = getPoints.getAltitude(latitude, longitude);
12    }
13 }
```

LISTING 5.2: Implementation of acquiring the camera's position

5.4.3 Accessing LOD Databases

After determining the position of the camera, we query all resources that are situated in the surrounding area from LOD sources. Therefore, we define SPARQL queries for obtaining data from LOD sources such as DBpedia or LinkedGeoNames. Listing 5.3 shows an extract of how to execute these queries. We use the class `QueryFactory` for generating a query. The query is executed by calling a SPARQL endpoint offered by different LOD sources. This endpoint returns a `ResultSet` that can be iterated and the required information can be selected. Besides the identifier, a label, and the geographic coordinates, we query for basic information such as the abstract, a type and a picture link in separate queries.

```
1  ArrayList<String[]> queriedResources = new ArrayList<String[]>();
2  Query query = null;
3  try {
4     query = QueryFactory.create(queryString);
5  } catch (Exception e) {
6     Log.v("Error", "Error creating query");
7  }
8  QueryExecution qexec = QueryExecutionFactory.sparqlService(getEndpoint(), query);
9  try {
10    if (qexec != null) {
11       ResultSet results = qexec.execSelect();
12       qexec.close();
13       for (; results.hasNext();) {
14          QuerySolution soln = results.nextSolution();
15          String resourceID = soln.get("subject").toString();
16          String longit = soln.getLiteral("long").getLexicalForm();
17          String latit = soln.getLiteral("lat").getLexicalForm();
18       }
19    }
20 } catch (Exception e) {
21    Log.v("Error", "Error executing query");
22 } finally {
23    qexec.close();
24 }
```

Listing 5.3: Query geo-information about resources in the environment

An important aspect of our framework is that it can be extended by any LOD source that provides an endpoint for querying their resources. In Figure 5.3, we show the integration of a LOD source in our framework. In order to provide an overview of the integration we briefly describe all related classes.

DemoStartActivity The class `DemoStartActivity` extends the class `StartActivity` which is the main activity that is called on application launch as described in Section 5.3. `DemoStartActivity` can be modified by overriding methods of the StartActivity class. In order to provide more than one LOD source the method for displaying checkboxes representing each LOD source can be extended. Thereby, checkboxes

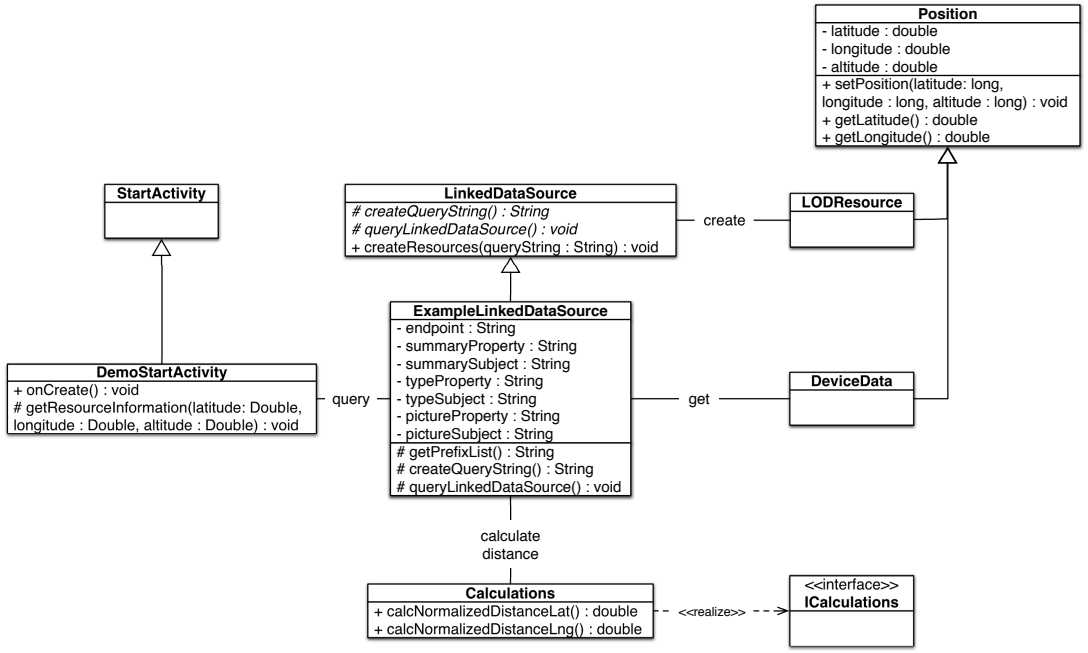

FIGURE 5.3: Class infrastructure for querying resources from LOD sources

can be provided that allow a selection of LOD sources which will be queried. Moreover, the class `DemoStartActivity` includes a method for querying LOD sources called `getResourceInformation(Double latitude, Double longitude, Double altitude)`. Overriding this method allows to query from different LOD sources.

LinkedDataSource The class `LinkedDataSource` is an abstract class that provides methods for querying LOD sources. In order to add different LOD sources this class has to be extended. The class `LinkedDataSource` provides general methods for querying LOD sources and storing the queried information, e.g, the method `getPrefixList()` provides a list of commonly used prefixes. The prefixes are used to generate a valid SPARQL query.

ExampleLinkedDataSource The class `ExampleLinkedDataSource` represents an example of a LOD source that is queried for resources. Adding additional LOD sources implies to create a class that extends the class `LinkedDataSource` for each LOD source. Each of these classes has to provide at least an endpoint that can be queried by SPARQL. Moreover, additional information such as properties and subjects of RDF triples for querying a summary, a type, or a picture can be defined. In order to provide a specific query string or additional prefixes the methods `createQueryString()` or `getCommonPrefixList()` can be used.

LODResource The class `LODResource` represents a resource that is queried, generated, and stored. For instance, basic information or position data is stored in this class.

Once resources are queried by the class `LinkedDataSource`, an instance of this object is generated for each resource.

DeviceData In the class `DeviceData` information about the camera is stored. For instance, in order to query resources that are located nearby the camera's position, the geographic position of the camera as well as the device's orientation have to be determined and stored.

Calculations In order to create a valid query string the class `Calculations` provides several methods for calculating distances. It realizes the interface `ICalculations`. The method `calcNormalizedDistanceLat()` calculates the distance between the latitude value of a resource's position and the latitude value of the camera's position.

As soon as, general information about the resources are queried, we provide methods for querying additional information about them. Therefore, we need to access the LOD databases for obtaining all available information about a specific resource. For that, we decided to use the HTTP protocol that enables to access the resources by their identifiers. Listing 5.4 exemplifies a method that sends an HTTP request and accepts RDF data as response. The returned data is stored in an RDF model. This RDF model is used by the Mobile RDF Browser for displaying information about the resources. Thereby, the user can browse for further information about specific resources.

```
1  private Model fetchUri(String uri) throws ClientProtocolException, IOException {
2     Model model = ModelFactory.createDefaultModel();
3     RDFReader r = model.getReader("RDF/XML");
4
5     HttpClient client = new DefaultHttpClient();
6     HttpGet get = new HttpGet(Uri.parse(uri).toString());
7     get.addHeader("accept","application/rdf+xml");
8     HttpResponse res = client.execute(get);
9
10    InputStreamReader isr = new InputStreamReader(res.getEntity().getContent());
11    try {
12       r.read(model,isr,uri);
13    } catch (Exception e) {
14       Log.d("Error", "Error reading model");
15    }
16    return model;
17 }
```

Listing 5.4: Creating an RDF model of Linked Data from a specific resource

5.4.4 Coordinate Transformation

After getting the position and orientation of the camera and the positions of the resources in the surroundings, we can implement the mathematical model for calculating the screen coordinates. Basically, the transformation of the coordinates is implemented

in the class `Calculations` that realizes the Interface `ICalculations`. In Figure 5.4 we provide an overview of the class infrastructure for transforming the coordinates. The methods in the class `Calculations` are called from the class `DisplayPointsView` that is responsible for displaying the resources on top of the camera interface.

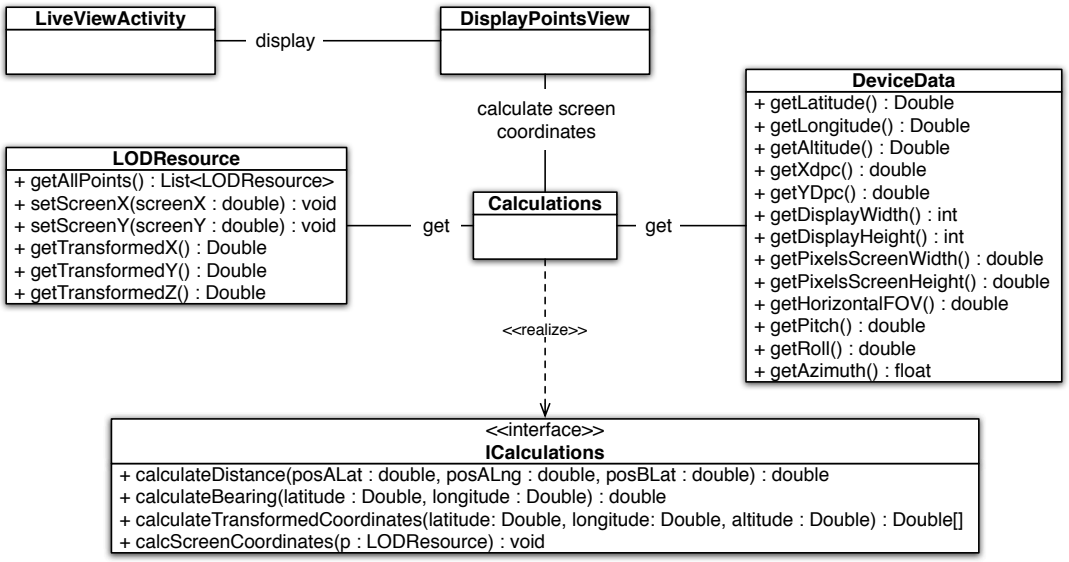

FIGURE 5.4: Class infrastructure for screen coordinate calculation

In order to execute the transformations, several values need to be extracted from the Android device such as the display width and height, pixels per mm, and the horizontal and vertical field of view cf. Definition 4.2 in Chapter 4. Android offers two classes called `DisplayMetrics` and `Display` for describing general information about the display. Listing 5.5 shows how these classes are used for retrieving the display width and height as well as the pixels per mm. Additionally, we determine the absolute height and width of the display in pixels.

```
1  DisplayMetrics metrics = new DisplayMetrics();
2  getWindowManager().getDefaultDisplay().getMetrics(metrics);
3
4  // Convert from dots per inch to dots per centimetre
5  double xdpc = (double) (metrics.xdpi / 25.4);
6  double ydpc = (double) (metrics.ydpi / 25.4);
7
8  double heightPixels = metrics.heightPixels;
9  double widthPixels = metrics.widthPixels;
10
11 Display display = getWindowManager().getDefaultDisplay();
12 Point size = new Point();
13 display.getSize(size);
14 int displayWidth = size.x;
15 int displayHeight = size.y;
```

LISTING 5.5: Android's display metrics and display

Moreover, we need to determine the horizontal and vertical field of view in our calculations. For that, Android offers a class called `Camera`. Listing 5.6 shows how to access the required information about the camera. First, an instance of `Camera` has to be created. After that, settings such as the horizontal and vertical field of view angle can be obtained. The parameters return `float` values and describe the angle of view in degrees.

```
1 private Camera camera = Camera.open();
2 if(camera != null){
3    Parameters p = camera.getParameters();
4    float hFOV = p.getHorizontalViewAngle();
5    float vFOV = p.getVerticalViewAngle();
6 }
```

LISTING 5.6: Accessing information about the camera

In the following we will describe the four methods provided by the class `Calculations` for calculating the screen coordinates of the resources.

`calculateDistance` This method calculates the distance between two geographic coordinates. The implementation realizes the approach described in Section 4.4.2.

`calculateBearing` The method is used to calculate the bearing between the current camera's position and an optional geographic position. It realizes the approach described in Section 4.4.1.

`calculateTransformedCoordinates` The coordinates have to be transformed according to the camera's position as described in Chapter 4.4.3. Thereby, we calculate the distances and change the signs according to the bearing. In Listing 5.7, we show the transformation of the coordinates according to the camera's position.

```
1
2 public Double[] calculateTransformedCoordinates(Double latitude, Double longitude
     , Double altitude) {
3    Double[] worldA = new Double[3];
4    Double distanceXLat = calculateDistance(device_latitude,
5          device_longitude, device_latitude, longitude);
6    Double distanceYLng = calculateDistance(device_latitude,
7          device_longitude, latitude, device_longitude);
8    Double distanceZAlt = altitude - device_altitude;
9
10   double b = calculateBearing(latitude, longitude);
11   if (b >= 0 && b <= 90) {
12      worldA[0] = distanceYLng;
13      worldA[1] = distanceXLat;
14      worldA[2] = distanceZAlt;
15   } else if (b > 90 && b <= 180) {
16      worldA[0] = -distanceYLng;
17      worldA[1] = distanceXLat;
18      worldA[2] = distanceZAlt;
19   } else if (b > 180 && b <= 270) {
```

```
20          worldA[0] = -distanceYLng;
21          worldA[1] = -distanceXLat;
22          worldA[2] = distanceZAlt;
23      } else if (b > 270 && b < 360) {
24          worldA[0] = distanceYLng;
25          worldA[1] = -distanceXLat;
26          worldA[2] = distanceZAlt;
27      } else {
28          worldA[0] = distanceYLng;
29          worldA[1] = distanceXLat;
30          worldA[2] = distanceZAlt;
31      }
32      return worldA;
33  }
```

LISTING 5.7: Resource coordinate transformation according to camera's position

`calcScreenCoordinates` This method allows the calculation of the resources' screen coordinates after the coordinates are transformed. Therefore, the coordinates are rotated around each axis in order to meet the camera's orientation, as described in Listing 5.8. Thereby, the camera's coordinates `(ax, ay, az)`, the camera's orientation in radians `(thetaX, thetaY, thetaZ)`, and the resource's coordinates `(cx, cy, cz)` are used.

```
1  // camera orientation angles:
2  double cos0x = Math.cos(thetaX);
3  double sin0x = Math.sin(thetaX);
4  double cos0y = Math.cos(thetaY);
5  double sin0y = Math.sin(thetaY);
6  double cos0z = Math.cos(thetaZ);
7  double sin0z = Math.sin(thetaZ);
8
9  // rotation around each axis
10 double cameraX = cos0y * (sin0z * (ay - cy) + cos0z
11                         * (ax - cx)) - sin0y * (az - cz);
12 double cameraY = sin0x * (cos0y * (az - cz) + sin0y
13                         * (sin0z * (ay - cy) + cos0z * (ax - cx))) + cos0x
14                         * (cos0z * (ay - cy) - sin0z * (ax - cx));
15 double cameraZ = cos0x * (cos0y * (az - cz) + sin0y
16                         * (sin0z * (ay - cy) + cos0z * (ax - cx))) - sin0x
17                         * (cos0z * (ay - cy) - sin0z * (ax - cx));
```

LISTING 5.8: Resource coordinate rotation according to camera's orientation

Next, we implement the projection transformation. Therefore, we use the Equations 4.14 and 4.15 from Chapter 4.4.6. In Listing 5.9, the implementation is summarized. First, we are calculating the width and height of the screen as well as the values for S_x and S_y, as illustrated in Equations 4.14 and 4.15 from Chapter 4.4.6. Next, we use the value $P'_{z'}$ for checking if the resource is situated in front of the camera or behind the camera. If the resource is located in front of the camera we calculate the projection coordinates and adapt them to device coordinates. These coordinates are displayed in the camera's field of view of the mobile device as described in the following chapter.

```
1  double pixelmX = xdpc*1000;
2  double pixelmY = ydpc*1000;
3  double screenWidth = displayWidth/xdpc;
4  double screenHeight = displayHeight/ydpc;
5  double viewWidth = pixelsScreenWidth;
6  double viewHeight = pixelsScreenHeight - 25;
7
8  double Sx = (screenWidth/2)/(Math.tan(Math.toRadians(hFOV/2)))/1000;
9  double Sy = (screenHeight/2)/(Math.tan(Math.toRadians(vFOV/2)))/1000;
10
11 // cameraX, cameraY, cameraZ are viewing coordinates of the resource
12 if (cameraZ < 0){
13    double deviceX = cameraX / cameraZ * Sx * pixelmX + viewHeight / 2;
14    double deviceY = -cameraY / cameraZ * Sy * pixelmY + viewWidth / 2;
15    if(deviceX>0 && deviceX<=viewHeight && deviceY>0 && deviceY<viewWidth){
16       p.setScreenX(deviceX);
17       p.setScreenY(deviceY);
18    }
19    else {
20       p.setScreenX(-1.0);
21       p.setScreenY(-1.0);
22       }
23    } else {
24       p.setScreenX(-1.0);
25       p.setScreenY(-1.0);
26 }
```

Listing 5.9: Perspective projection and calculation of screen coordinates

5.4.5 Implementation of the Off-Screen Objects in a Minimap

Basically, the implementation of a minimap that visualizes the off-screen objects is improving usability since it provides an orientation guide to the user. The approach for displaying a minimap is described detailed in Chapter 4.5. Just as drawing the points in the preview of the camera, we decided to use a custom view for visualizing the minimap. The minimap consists of the following parts that have to be illustrated: the points representing the resources' position, a circle that indicates the radius, a central point that depicts the camera's position, and two lines that constrain the field of view. Additionally, the radius is written below the minimap.

In order to realize the minimap, we draw a point in the center and a circle indicating the compass radius. Moreover, we draw two lines restricting the camera's field of view. Therefore, we calculate the endpoints of the two lines for once. The calculations are described more detailed in Chapter 4.5. Once we have drawn the basic minimap, we set the position of the resources and display them in the minimap.

The method illustrated in Listing 5.10 shows the calculation of the points in the minimap. The method needs the following input parameters: the first two parameters `(centerX, centerY)` are the center point of the minimap, the next two parameters

`(rPointX, rPointY)` indicate the reference point on the circle, and the last parameter `angle` is the angle between the reference point and the resource. The calculations of the reference point's coordinates as well as the angle between the reference point and the resource are described in Chapter 4.5.

```
1  private Float[] calculateCompassPoint(Float centerX, Float centerY, Float rPointX
       , Float rPointY, double angle) {
2      Float[] point = new Float[2];
3      angle = Math.toRadians(angle);
4
5      point[0] = (float) ((rPointX - centerX) * Math.cos(angle)
6          - (rPointY - centerY) * Math.sin(angle) + centerX);
7          point[1] = (float) ((rPointX - centerX) * Math.sin(angle)
8          + (rPointY - centerY) * Math.cos(angle) + centerY);
9
10     return point;
11 }
```

Listing 5.10: Calculating coordinates of POIs in the minimap

Figure 5.5 depicts an example of the minimap. It is located on the upper left corner of the camera view.

Figure 5.5: Example of the minimap

5.5 Additional Features

Besides the visualization of the resources in the camera view of a mobile device, we integrate some additional features in order to improve usability. These functions allow to get a better overview of the resources that are in the surrounding area and they make it possible to query for further information.

5.5.1 Visualizing Data in Google Maps View

We decide to implement a view that shows the resources located in the environment on a map. On the one side the user can utilize the map to check if his location is determined correctly and on the other side the user can look up the exact positions of

the resources in the map. Therefore, we use the Map APIs[1] for Android provided by Google[2]. Google provides an extension to the Android SDK development environment. The Maps external library allows to add mapping capabilities to an Android application. The external library, `com.google.android.maps`, offers the class `MapView` which enables to display maps with data provided from the Google Maps service. It offers many capabilities such as zooming or focusing and it allows to create Overlay Items.

Figure 5.6 depicts the map we implemented for visualizing the position of the camera as well as the positions of the resources in the surrounding area. Moreover, the resources are clickable and provide basic information such as a label and the geographic coordinates.

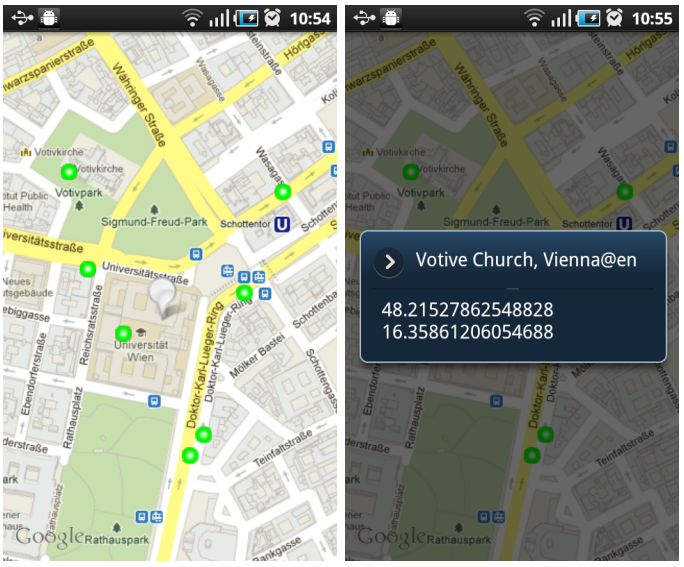

FIGURE 5.6: Google Maps view

The Google Maps View can be called from the activity `StartInterface` as well as from the options menu of the activity `LiveView`. The activity `StartInterface` offers additionally a static map that displays only the current position of the user that is determined by the GPS signal. We use the Google Static Maps API[3] for creating the static map. The API allows to create a picture of a specific map that can be easily downloaded. Therefore, we need to create a URL such as:

```
http://maps.googleapis.com/maps/api/staticmap?center=48.213349,16.360531&zoom=
14&size=256x256&markers=color:blue|label:S|48.213349,16.360531&sensor=false
```

The returned image of this URL is illustrated in Figure 5.7.

[1]Google Projects: Android: `http://code.google.com/android/add-ons/google-apis/index.html`
[2]Google: `http://www.google.com`
[3]Static Maps API: `http://code.google.com/intl/de/apis/maps/documentation/staticmaps`

FIGURE 5.7: Google Maps static map of the camera position

5.5.2 Google Elevation API - Requesting missing Elevation Data

Due to the fact, that sometimes GPS or the Android specific NETWORK PROVIDER does not provide data that determines the altitude of the resources, we decide to use a web service called Google Elevation API[4]. The service allows to query elevation data for any desired location. In order to get the elevation data we send the following request containing the latitude and longitude of the position.

```
http://maps.googleapis.com/maps/api/elevation/xml?locations="+latitude+",
"+longitude+"&sensor=true
```

An example of the request is:
`http://maps.googleapis.com/maps/api/elevation/xml?locations=48.306719,14.285703&sensor=true`

This service returns an XML file that contains the elevation in meters. Listing 5.11 shows the response of the example query.

```
1  <ElevationResponse>
2     <status>OK</status>
3     <result>
4        <location>
5           <lat>48.3067190</lat>
6           <lng>14.2857030</lng>
7        </location>
8        <elevation>261.2418823</elevation>
9        <resolution>152.7032318</resolution>
10    </result>
11 </ElevationResponse>
```

LISTING 5.11: Response from sending a request to Google Elevation API

[4]The Google Elevation API: `https://developers.google.com/maps/documentation/elevation`

5.5.3 Earthtools - Requesting missing Elevation Data

Earthtools[5] provides a web service called "Elevation/Height Above Sea Level" for determining the altitude value for arbitrary positions. We use this tool additionally to the Google Elevation API in case that we do not receive valid altitude data from the Google Elevation API. In order to get the height above sea level a request with latitude and longitude parameters is sent to the service. The service can be accessed with the following URL:

`http://www.earthtools.org/height/<latitude>/<longitude>`

An example query is:
`http://www.earthtools.org/height/48.213349/16.360531`

This service returns an XML file that contains the elevation in meters and feet. The value is rounded to one decimal place. Listing 5.12 shows the response of the example query.

```
<height xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.earthtools.org/height.xsd">
        <version>1.0</version>
        <location>
                <latitude>48.213349</latitude>
                <longitude>16.360531</longitude>
        </location>
        <meters>180</meters>
        <feet>590.6</feet>
</height>
```

Listing 5.12: Response from querying data from `www.earthtools.org`

Additionally, some Linked Data sources do not cover altitude values of the POIs. Due to that, we use the service to complete missing data.

5.5.4 Integration of the Mobile RDFBrowser Interface

The Mobile RDFBrowser Interface allows to query data from different data sources and it provides several methods for browsing Linked Data Sources. Due to that, we decide to integrate the Mobile RDFBrowser Interface in our framework. The interface can be called through the popup window that displays basic information about a resource. If the user selects "further information" the Browser Interface is called. Once information about a resource is queried, the URI is fetched and the returned RDF data is stored in an RDF model. Afterwards the Mobile RDFBrowser displays all information that is stored in the model.

[5]EarthTools: `http://www.earthtools.org/`

Figure 5.8 shows the interface of the RDFBrowser that displays all information about a specific resource. Additionally, the Browser offers several features for querying for more information. The triples are structured the following way: first, all the RDF triples are listed and thereafter, the literals are summarized. All triples that share the same predicate are clustered together. Moreover, links that contain pictures are highlighted with an image icon and the properties `owl:sameAs` and `rdf:seeAlso` are emphasized too. The pictures can be viewed by calling the context menu. Furthermore, the geographic coordinates are represented on a static map. Therefore, the Static Map API provided by Google is used. The static map displayed in the RDF Browsing Interface depicts the user's position as well as the position of the POI. Additionally, if the user selects the static map, a Google Maps View Interface is called that allows features such as zooming. Moreover, the markers of the maps view are clickable and display the exact latitude and longitude values. Generally, the RDF Browser allows to query for more triples by clicking on RDF objects. In doing so, a new search is started with the object. Additionally, the RDFBrowser Interface implements the service offered by `http://sameas.org`. This service allows to query for identical resources.

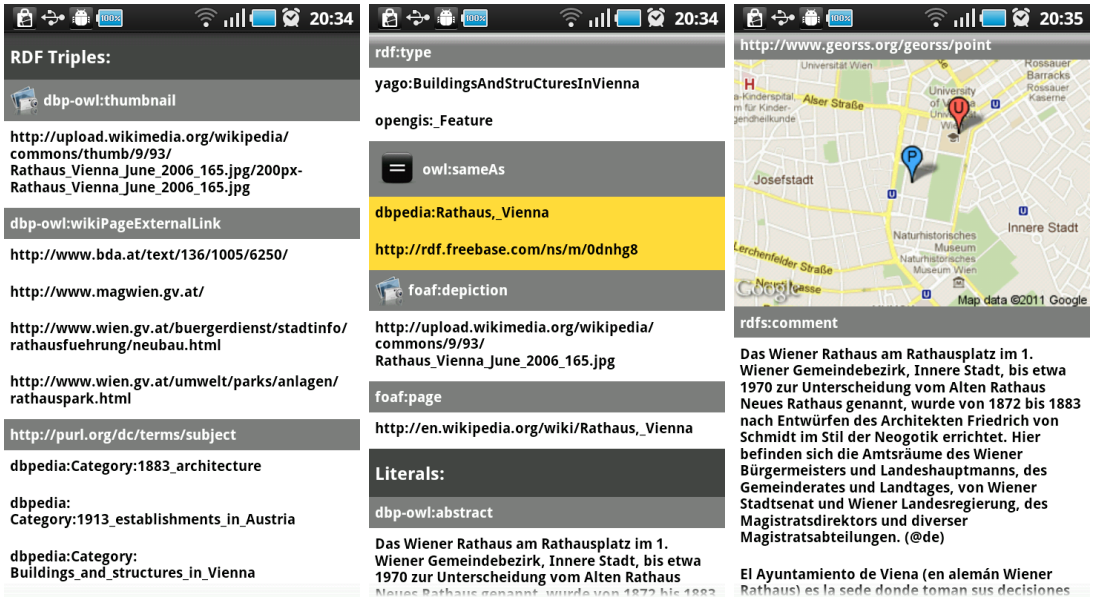

FIGURE 5.8: Mobile RDFBrowser Interface

5.6 Summary

In this chapter, we provided an overview of the system architecture of the AR framework. Moreover, implementation details regarding the mathematical model are discussed. First, we described the four design goals that are realized by our framework: extensibility, accurate representation, completeness, and usability. Our primary goals are to provide a framework that can be extended by adding new LOD sources and that

represents the resources as accurate as possible in the user interface. Additionally, we described the constitutional components of our AR frameworks. Some of the components are a LocationManager as well as an OrientationManager for getting the position and orientation data of the camera. A Linked Data Service provides access to SPARQL endpoints and it can be extended by any LOD source that provides an endpoint.

In Section 5.4.4, we described the realization of the mathematical model for displaying resources. Therefore, we focus on getting sensor data from a mobile device such as the orientation data. In order to get the orientation of the device we use the accelerometer as well as the magnetic field sensor. Additionally, we describe how the position of the camera as well as the resources in the environment can be retrieved. We use the `GPS PROVIDER` as well as the `NETWORK PROVIDER` offered by Android for getting the geographic coordinates. Before calculating the exact screen coordinates of the resources, we determine specific parameters of the camera such as the horizontal and vertical field of view as well as the screen resolution. In order to calculate the device coordinates, we transform the geographic coordinates of the resources and rotate them according to the camera's orientation. Afterwards, we perform a perspective projection. In order to present the resources in the camera's field of view they have to be transformed to device coordinates and be drawn in the interface. Finally, we described the implementation of the off-screen objects in a minimap.

In the last part of this section, we described some additional features such as the integration of the Mobile RDFBrowser Interface. The interface allows to query for further information about resources. Since the GPS sometimes returns an inaccurate elevation data, we integrated web services from Google Elevation API and Earthtools in order to get the altitude of an arbitrary position. Additionally, an implementation of a Google Maps map shall improve usability and allow the user to check her position and the position of the resources in the environment on a map.

Chapter 6

Implementation for a Specific Domain - Integrating RDF data from DBpedia

Because of the fact, that the DBpedia project is one of the biggest datasets of Linked Data, we want to give an understanding of the approach and the implementation by describing the integration of the LOD source DBpedia in our AR framework. Therefore, we describe implementation details of the integration of a new LOD source in Section 6.1. In the following Section 6.2, a proof of concept implementation of the framework that integrates DBpedia is presented.

6.1 Integration of the DBpedia project in the AR framework

In this section we want to show the extension of our AR framework by adding the DBpedia project as LOD source. We decide to use the DBpedia project because it provides a SPARQL endpoint and additional information such as pictures. Moreover, it returns data in RDF format and allows to access the resources by their identifiers.

As already described in Section 5.4.3, new LOD sources can be added by integrating a new class, e.g., `DBpediaLinkedDataSource` that extends the abstract class `LinkedDataSource`. Let the new class implement the abstract methods and provide information about the new LOD source. Therefore, we define the values summarized in Table 6.1 for a LOD source.
Once these values are defined, we override the method `createQueryString()` in order to provide a valid query that is used for querying DBpedia. The implementation of this method is shown in Figure 6.1. In order to define the boundary within the resources can be located, we use the methods `calcNormalizedDistanceLat()` and `calcNormalizedDistanceLng()` from the class `Calculations`. This methods realize

Type	Variable	Value	Description
String	endpoint	http://dbpedia.org/sparql	defines the endpoint that is queried
String	summaryProperty	dbpedia-owl:abstract	defines the property of an RDF triple to query for a short summary
String	typeProperty	rdf:type	defines the property of an RDF triple to query for the resource's type
String	pictureProperty	ontology:thumbnail	defines the property of an RDF triple to query for a picture

TABLE 6.1: List of variables to define for each LOD source

the approach described in Chapter 4.2 for determining the maximum distance between two coordinates. Once the values for the minimum and maximum latitude as well as the minimum and maximum longitude are determined, they can be used in the SPARQL query. Moreover, the SPARQL query uses a predefined prefix list. The values `latitude` and `longitude` define the position of the camera. The summary, type, or picture are automatically queried in a separate query. Therefore, only the values have to be defined.

```
1  @Override
2  protected String createQueryString() {
3     double valueLat = calc.calcNormalizedDistanceLat();
4     double valueLng = calc.calcNormalizedDistanceLng();
5
6     double minValueLat = latitude - valueLat;
7     double maxValueLat = latitude + valueLat;
8     double minValueLng = longitude - valueLng;
9     double maxValueLng = longitude + valueLng;
10
11    String queryString = "";
12    queryString = getCommonPrefixList()
13    + "PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> "
14    + "SELECT ?subject ?lat ?long ?label WHERE { "
15    + "?subject geo:lat ?lat. "
16    + "?subject geo:long ?long. "
17    + "?subject rdfs:label ?label. "
18    + "FILTER((" + minValueLat +" <= ?lat && ?lat <= "+ maxValueLat
19    + " && "+ minValueLng +" <= ?long && ?long <= "+ maxValueLng +")"
20    + " && lang(?label) = \"en\" "
21    + "). " +
22    "}";
23    return queryString;
24 }
```

LISTING 6.1: Create a query for DBpedia

After providing a SPARQL query, we override the method

`queryLinkedDataSource()`

that allows to execute a query. Therefore, we call

`createResources(createQueryString())`

from the class `LinkedDataSource` and pass the query as parameter. In order to run a query, this method is called from a class (e.g. `TestLiveViewActivity`) that extends the `LiveViewActivity`. Therefore, we override the method

`getPointsInformation(Double latitude, Double longitude, Double altitude)`,

create an instance of the new LOD source, and call
`queryLinkedDataSource()` for querying the DBpedia Linked Data source.

6.2 Presentation of the AR Framework Integrating the DBpedia Project

The basic scenario around which our system is designed is to provide information about resources from LOD databases in an AR view. For instance, a user is traveling in a foreign city and is standing in front of a big historical building. The user wonders what building it is and how old it is. She launches our application on her mobile phone and selects at the application launch that she wants to query the DBpedia database because it provides much information and pictures of the buildings. The building is located about 50 meters away. Thus, the user selects a radius of about 100 meters. Afterwards, she starts the camera preview. Looking around with the camera application she can find an icon providing information about the building right in front of it. She only needs to click on the icon if she wants to get more information about it such as the year of completion or the architectural style. In this section, we present a short walkthrough of the prototype implementing the DBpedia LOD source. We show how to run the prototype in order to present information about resources from the DBpedia project.

Figure 6.1 illustrates the start interface of the prototype. First, it presents the GPS data of the user's position. Once GPS information is available a map indicating the user's position appears at the bottom. The map is clickable and allows the user to detect if the retrieved GPS information is matching her real position. Beneath the GPS position, a SeekBar provided by Android allows to define the radius to query for resources. Under the bar the user can select which LOD databases she wants to query. For querying data from the DBpedia project she has to check the checkbox for DBpedia. Multiple choices are possible. After the GPS data is retrieved and the radius as well as the databases are

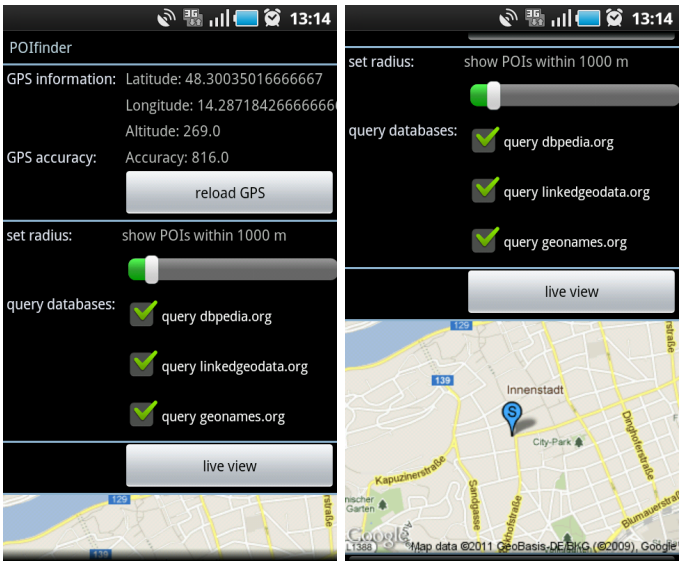

FIGURE 6.1: Use Case - Start Interface

selected, the user can select the button "liveview" for displaying the camera's current live view.

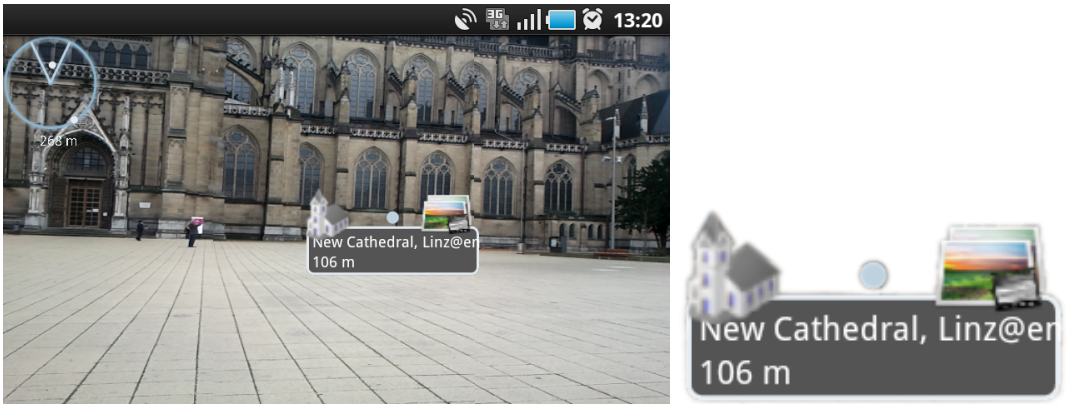

FIGURE 6.2: Use Case - camera live view

The camera view is illustrated in Figure 6.2. On the upper left corner a mini-map appears that indicates the positions of the off-screen objects. While launching the live preview a hint appears on the bottom right corner telling "loading information..." . Generally, the camera interface displays the position of the resources as depicted in Figure 6.2. The visualization contains the following elements: a point, that illustrates the exact screen coordinates of the resource, the label of the resource, the distance indicating the linear distance to the resource, a picture signifying that a photo of the resource is available, and an icon that assigns a certain category to the resource. The image icon is clickable and the picture appears in a popup window, as illustrated in Figure 6.3.

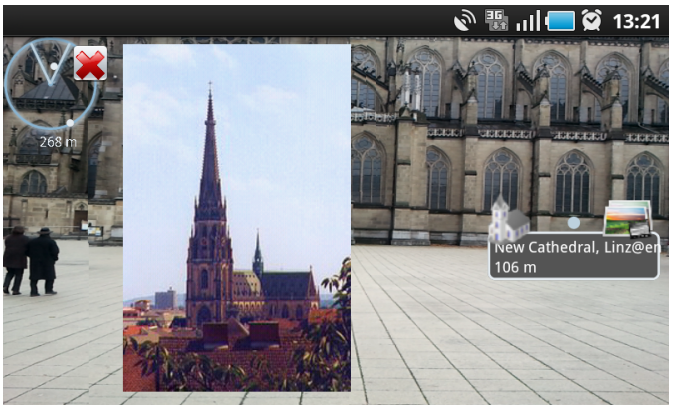

FIGURE 6.3: Popup window for image icon

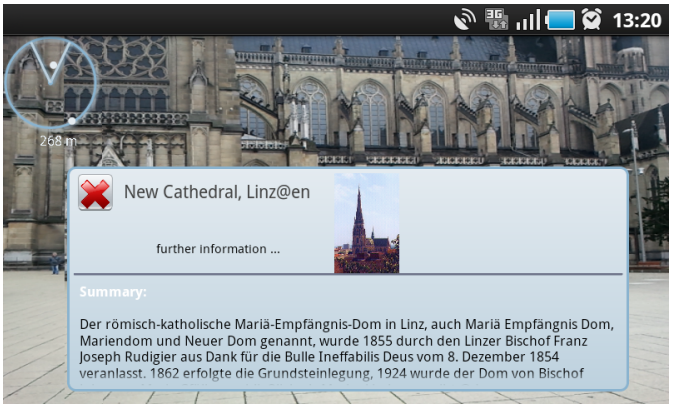

FIGURE 6.4: Popup window

Moreover the resource is clickable and a popup window with basic information about the resource appears. Figure 6.4 shows the popup window that contains a label displayed on top. Below the label, there is a link that can be called for getting more detailed information about a resource. This link calls the Mobile RDFBrowser interface for querying further information. The interface allows to process data from DBpedia because the data is in RDF format. The interface is described more detailed in Chapter 5.5.4. Next to the link, a thumbnail of the resource is displayed if a picture is available. This thumbnail is clickable and it opens a popup window presenting the photo. Below the link and the photo there is a scroll view located that contains a short summary, the types of the resource, the distance to the resource as well as the geographic coordinates.

Chapter 7

Evaluation

7.1 Evaluation of Proof-of-Concept Implementation

In this chapter, we focus on the evaluation of our proof-of-concept implementation in order to analyze our approach. First, we compare our prototype to DBpedia Mobile cf. Chapter 3, Section 3.2. Thereby, the number of queried and displayed resources as well as additional functionality is compared.

In Section 7.1.2, we analyze the accuracy of geographic information from LOD resources. The correct representation of the resources in the camera's field of view depends on several aspects such as the accuracy of the geographic data, the precision of the orientation sensors, or the correctness and exactness of the calculations for transforming the coordinates. In this part, we focus on the accuracy of the resources' positions queried from LOD sources. Therefore, we compare the queried geographic positions of LOD resources with their equivalent position on a Google Maps map. This evaluation gives some indication of whether our location-based approach exhibits the necessary accuracy for displaying LOD resources in an AR interface. Moreover, the data quality of geographic information of LOD resources is analyzed.

In the last section, we focus on the evaluation of the times for the calculation of screen coordinates and representation of resources in the camera's field of view. Thereby, we evaluate which aspects impact representation and how many resources can be displayed without affecting the usability. Measuring the times for processing data, allows to define how fluently the data is represented in the camera interface.

7.1.1 Comparison to DBpedia Mobile

In this section, we compare our framework to DBpedia Mobile. DBpedia Mobile is described more detailed in Chapter 3, Section 3.2. First, we check if the displayed data in our prototype differs compared to data from DBpedia Mobile. In order to get test data for comparing the results, we launch our framework at six different places and on

two different devices. Moreover, DBpedia Mobile is launched with the same preconditions (geographic position and radius). After comparing all resources with DBpedia Mobile, we come to realize that the same resources are queried and we can not find any differences.

In contrast to DBpedia Mobile, our framework enables to define a radius within which the resources are queried. The comparison of the queried resources is difficult, since DBpedia Mobile does not provide a functionality for querying resources within a specified radius. Thus, all resources have to be compared directly by considering the distance of the resource's position to the camera's position.

Contrary to DBpedia Mobile, our application allows directly to query for resources from other LOD services such as LinkedGeoData or GeoNames. Thus, the integration of different LOD services allows to display more resources in the AR interface than DBpedia Mobile. Just as DBpedia Mobile, we enable to query for further information about the displayed resources.

7.1.2 Accuracy of LOD Resources' Geographic Positions

Since the accuracy of the resource's position affects the precision of the resource's representation in the camera's field of view, we analyze the geographic data received from three different LOD sources. Basically, the smaller the distance between the resources and the camera is the more accurate the geographic data has to be. Thus, we determine the average deviation of the resources positions retrieved from LOD sources compared to positions on a Google Maps map. The next section focuses on the requirements and the realization of this evaluation and in Section 7.1.2.2, the test results are presented and discussed.

7.1.2.1 Testing Environment and Realization

In Table 7.1, the LOD sources as well as the number of tested resources are represented. We tested LOD resources from three different sources.

LOD source	# of resources
DBpedia	50
LinkedGeoData	50
GeoNames	20

Table 7.1: Number of resources tested for determining the accuracy of their position

In order to determine the accuracy of the resources summarized in Table 7.1, the average deviation is calculated. The average deviation is determined by comparing the resources'

geographic position X_P, Y_P to the equivalent position on an actual Google Maps map X_M, Y_M. The deviation is calculated by defining the distance $f_{distance}(P, M)$.

7.1.2.2 Results and Discussion

The average deviations of resources are represented in Figure 7.1 and 7.2 the results from all three LOD sources are combined. On average, resources from DBpedia differ 27,80 m from the equivalent position on Google Maps. Moreover, for 50 % of the geographic positions a deviation of 13,18 m to 43,10 m was determined. Compared against resources from DBpedia, the geographic positions of resources from LinkedGeoData are more accurate. On average, the resources' positions vary 14,90 m and 50 % of the resources differ between 11,49 m and 30,18 m. In comparison to the LOD sources DBpedia and LinkedGeoData, GeoNames shows the most scattered results since 50 % of the geographic positions of the resources differ from 11,55 m to 59,61 m. Thus, the average deviation is determined by 31,23 m. Still, there exist a few resources containing geographic information that differ about 100 m as represented in Figure 7.1.

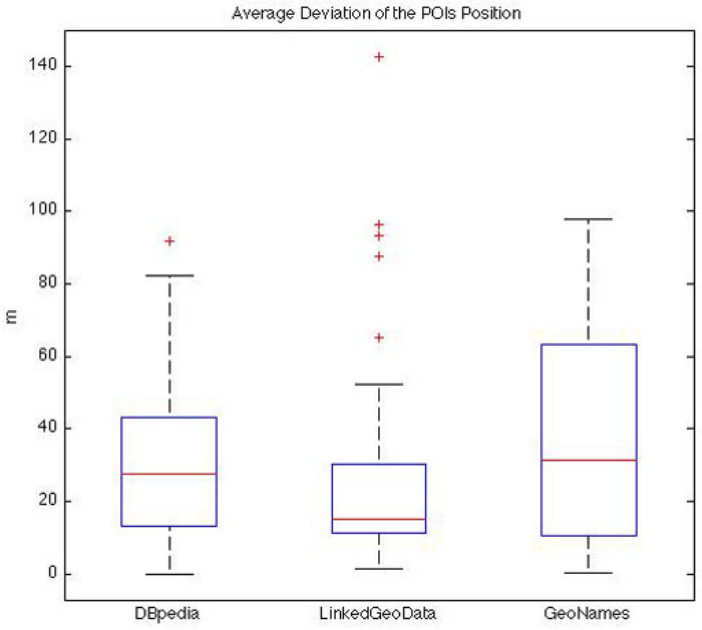

FIGURE 7.1: Average deviation of the resources' position

With a few exceptions, we obtain a reasonable result considering that buildings cannot be specified by an exact geographic position. For instance buildings range between several meters. These measurements illustrate that the accuracy of the presentation in an AR interface can be inexact due to inaccurate geographic information. Generally, most of the data is quite accurate and can be represented adequate. Generally, geographic information from LOD sources is quite precise and can be used for a location-based approach.

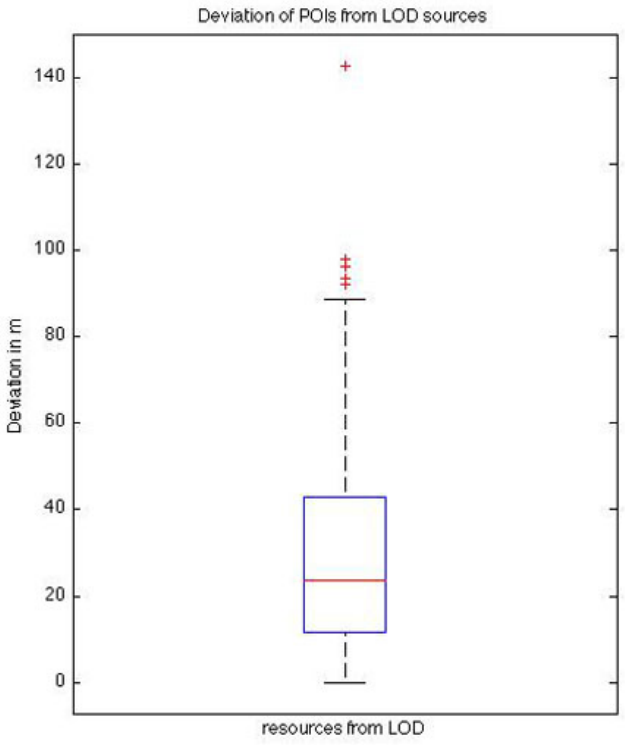

FIGURE 7.2: Average deviation of the resources' position from 3 LOD sources

7.1.3 Analyzation of Time for Processing and Displaying Data

In this section, we measure the time for processing and displaying data in the camera's field of view. Generally, the processing of data in our framework is divided into three different phases. First, the data is queried from LOD sources. Afterwards, LOD sources are consolidated and stored. Next, the screen coordinates have to be calculated and the resources have to be displayed in the camera's interface. We focus on the calculation and representation of the resources in the field of view in this section. Therefore, we describe the testing requirements and the realization of the evaluation in the next chapter. In Section 7.1.3.2, we present the results of our tests.

7.1.3.1 Testing Environment and Realization

First, we define which parts of the process of data handling are measured. In Figure 7.3, we summarize all steps for realizing data in the camera's field of view. Basically, we measure the coordinate conversion as well as the representation of the resources marked in blue in Figure 7.3. The coordinate conversion consists of the coordinate rotation, the perspective projection, and the transformation to screen coordinates. The coordinate transformation according to the camera's position is not considered since the coordinates are only transformed once at the launch. Moreover, the representation of the resources is included in the time measurements.

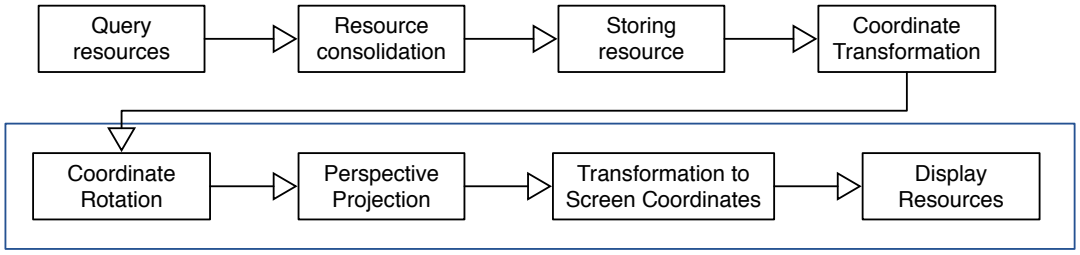

FIGURE 7.3: Processing steps for visualizing resources in camera interfac

After defining the measured process steps, we summarize the locations in Table 7.2. The performance is tested at six different locations. Additionally, we define a radius at 100m, 200m, 300m, 500m, and 1000m. Thus, the performance can be measured with a different number of resources and the impact of the number of resources to the times for processing and displaying data can be analyzed.

Location	Latitude	Longitude	Altitude (m)
Donauinsel Vienna	48.226588	16.411858	160.5
Linz Donau	48.308318	14.284930	267.5
Linz Central Station	48.290692	14.290295	265.5
Schönbrunn Palace	48.186225	16.312970	187.5
Stephansplatz Vienna	48.208434	16.372051	188.5
University of Vienna	48.213019	16.361014	181.5

TABLE 7.2: List of places for evaluation

For our tests, we use three mobile devices: Samsung GT-I9000 Galaxy S, LG P970 Optimus Black, and LG Nexus 4 E960. Thus, the measured data is not device depended. In the following Table 7.3, details of the test devices are summarized:

Additionally, we decide to query LOD resources from DBpedia as well as LinkedGeoData. The time for displaying resources depends on the number of pictures that are presented in the camera's field of view, too. Therefore, we decide to measure data from both LOD sources. Resources queried from DBpedia include pictures, whereas resources from LinkedGeoData do not display pictures.

The measurement of the processing times is built as follows: The prototype was launched at each devices and at each location. Additionally, each location is launched 5 times with a different radius each time. One launch measures continuously the time for transforming and displaying the resources. These times are logged and interpreted. In the next section the results are discussed.

	Samsung GT-I9000 Galaxy S	LG P970 Optimus Black	LG Nexus 4 E960
Release date	2010, June	2011, May	2012, November
Display size	4 inch	4 inch	4.7 inch
Display resolution	800 x 400	800 x 400	1280x768
CPU	1 GHz	1 GHz	1,5 GHz Quad-Core
Memory	512 MB RAM	512 MB RAM	2 GB RAM
Camera	5 MP	5 MP	8 MP
API Level	16	16	17
Android Version	4.1.1 Jelly Bean	4.1.1 Jelly Bean	4.2 Jelly Bean

Table 7.3: List of mobile test devices

7.1.3.2 Results and Discussion

As described in the previous section, we measure the performance for calculating screen coordinates and displaying resources. In Table 7.4, the number of resources queried from DBpedia and in Table 7.5 the number of resources from LinkedGeoData at each location and each radius is shown. Generally, LinkedGeoData provides more resources, e.g., at Stephansplatz in Vienna 16 resources are queried at a radius of 100 m and displayed from DBpedia, whereas LinkedGeoData provides 65 resources at the same location and the equivalent radius. Generall, LinkedGeoData provides more resources than DBpedia. However, DBpedia provides more information about specific resources and, e.g., allows to query for pictures.

Location	100 m	200 m	300 m	500 m	1000m
Donauinsel Vienna	0	1	1	1	12
Linz Donau	1	4	5	7	16
Linz Central Station	1	1	2	2	3
Schönbrunn Palace	0	1	2	4	9
Stephansplatz Vienna	16	23	40	84	160
University of Vienna	2	5	12	36	141

Table 7.4: Number of resources queried from DBpedia at specific radius

In Figure 7.4, the result of processing data from DBpedia is summarized. The values are measured when no LOD resources are located in the actual field of view (FOV) of the camera. Thus, no resources are displayed and only the coordinates are calculated. Generally, a difference between the values retrieved from different devices is visible. Since the Nexus 4 has the most CPU power as well as the most memory compared to the other two devices, the times for data processing are lower compared to the GT-I9000

Location	100 m	200 m	300 m	500 m	1000m
Donauinsel Vienna	0	1	2	16	101
Linz Donau	0	14	76	176	429
Linz Central Station	8	18	18	35	150
Schönbrunn Palace	1	6	9	16	101
Stephansplatz Vienna	65	139	250	559	1000
University of Vienna	7	27	62	148	769

TABLE 7.5: Number of resources queried from LinkedGeoData at specific radius

and the P970 Optimus. The calculation of screen coordinates for 160 resources needs 7.16 ms executed on the Nexus 4. Compared to that, the calculations last 30.75 ms on the P970 Optimus and 108.41 ms on the GT-I9000. In Figure 7.5, we present the results of processing data from LinkedGeoData on condition of displaying no resources in the current camera's field of view. Likewise the previous Figure 7.4, the best results are reached by the Nexus 4. Calculating the screen coordinates for 1000 resources takes 71.80 seconds on the Nexus 4. Contrary to that the calculations last 195.37 ms on the P970 Optimus and the worst result is achieved by the GT-I9000 with 842.59 ms.

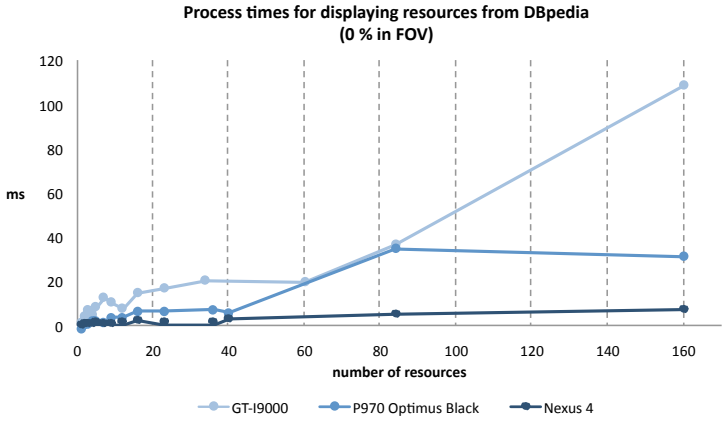

FIGURE 7.4: Average times for calculating screen coordinates and visualizing 0% of the resources in the camera's FOV from DBpedia

In Figures 7.6 and 7.7, we show the results of calculating and representing resources from DBpedia and LinkedGeoData on condition of displaying 10 % of the resources in the current camera's interface. The process times increase the more resources are calculated. Visualizing up to 10 % of 160 resources in DBpedia takes 808.31 ms on GT-I9000, 251.39 ms on P970 Optimus, and 68.87 ms on Nexus 4.

The results for calculating and displaying 10 % to 20 % of the resources are illustrated in the Figures 7.8 and 7.9. Moreover, the Figures 7.10 and 7.11 visualize the results for

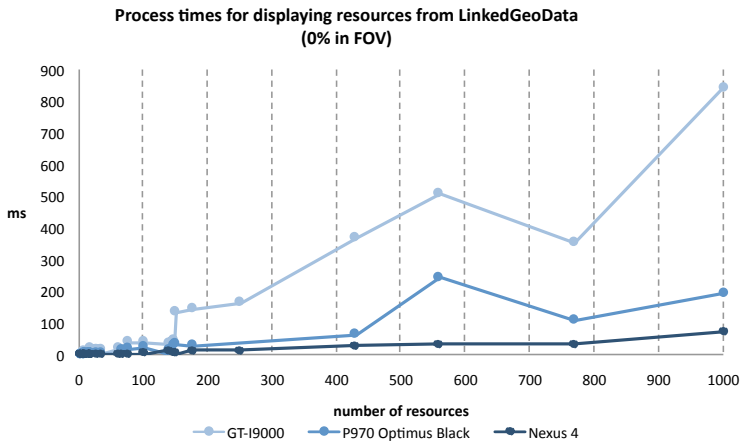

FIGURE 7.5: Average times for calculating screen coordinates and visualizing 0% of the resources in the camera's FOV from LinkedGeoData

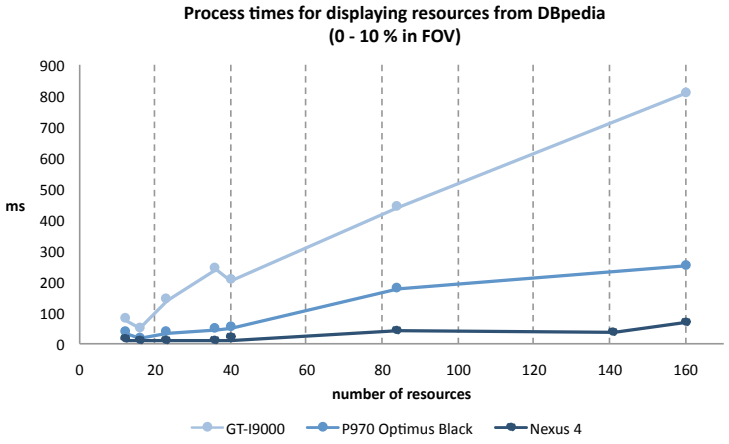

FIGURE 7.6: Average times for calculating screen coordinates and visualizing 0-10% of the resources in the camera's FOV from DBpedia

calculating and representing 20 % to 30 % of the resources in the camera interface from DBpedia and LinkedGeoData. Finally, the process times for coordinates conversion and representation from more than 30 % are summarized in the Figures 7.12 and 7.13.

In order to visualize the impact of the number of resources displayed in the current camera's field of view on the times for processing and displaying resources, we calculate the mean values of the results from all devices and show the values in Figure 7.14. We use resources from LinkedGeoData to illustrate the impact. As visualized, the more resources are displayed in the camera interface, the longer the calculation and representation takes. Representing none of 429 resources in the camera's field of view, takes 153.52 ms on average, whereas the representation of more than 30% of the resources lasts 2804.06 ms.

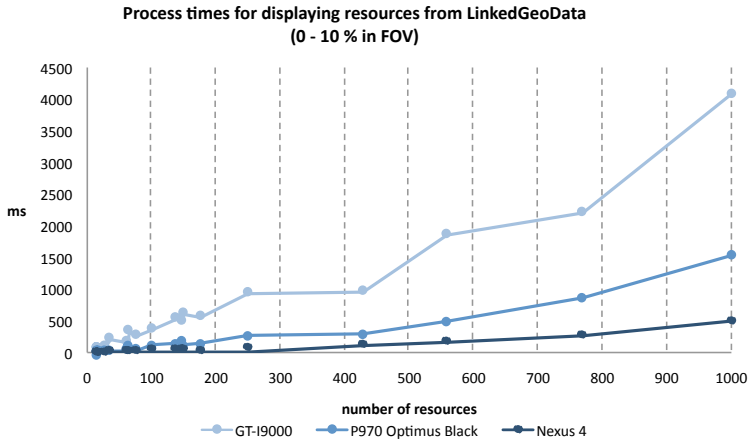

FIGURE 7.7: Average times for calculating screen coordinates and visualizing 0-10% of the resources in the camera's FOV from LinkedGeoData

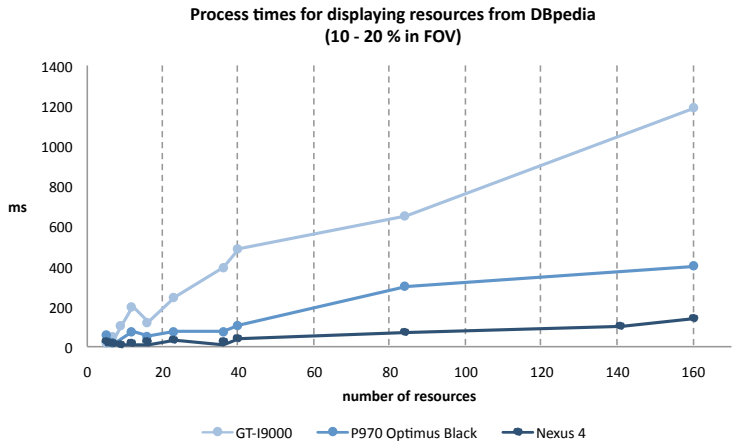

FIGURE 7.8: Average times for calculating screen coordinates and visualizing 10-20% of the resources in the camera's FOV from DBpedia

Additionally, we compare the times for processing data from DBpedia and LinkedGeoData, since DBpedia displays pictures too. In Table 7.6, we summarize the times for calculating the screen coordinates of 16 resources and displaying them from DBpedia and LinkedGeoData. All resources from DBpedia contain pictures while the resources from LinkedGeoData do not contain pictures. As shown in Table 7.6, the calculation and representation of resources from DBpedia last longer on average than of resources from LinkedGeoData. Thus, the more picture and information about the resources is displayed, the longer the process times are. Generally, the more resources are displayed in the current camera's field of view, the longer the process times are.

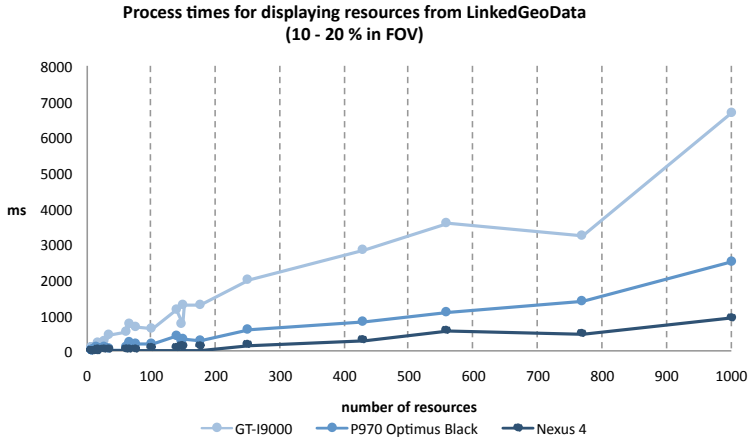

FIGURE 7.9: Average times for calculating screen coordinates and visualizing 10-20% of the resources in the camera's FOV from LinkedGeoData

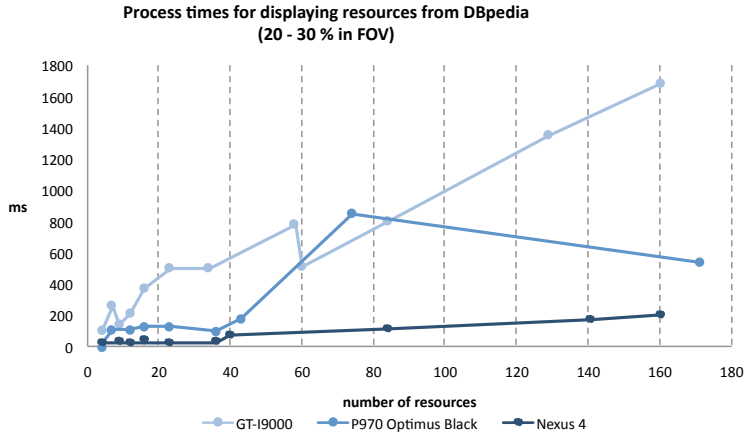

FIGURE 7.10: Average times for calculating screen coordinates and visualizing 20-30% of the resources in the camera's FOV from DBpedia

	% in FOV	0%	0-10%	10-20%	20-30%	+ 30%
DBpedia	Nexus 4	2.29	10.13	19.01	38.05	41.97
	GT-I9000	14.23	49.85	123.41	366.71	442.23
	P970 Optimus	6.02	17.62	52.87	125.00	158.18
	all devices	7.51	25.92	65.10	176.59	214.13
LinkedGeoData	Nexus 4	0.90	6.05	18.56	28.06	34.00
	GT-I9000	11.80	59.40	200.68	328.42	393.91
	P970 Optimus	2.06	23.20	63.13	66.17	93.10
	all devices	4.92	29.55	94.12	140.88	173.67

TABLE 7.6: Process times for calculating and displaying 16 resources from DBpedia and LinkedGeoData

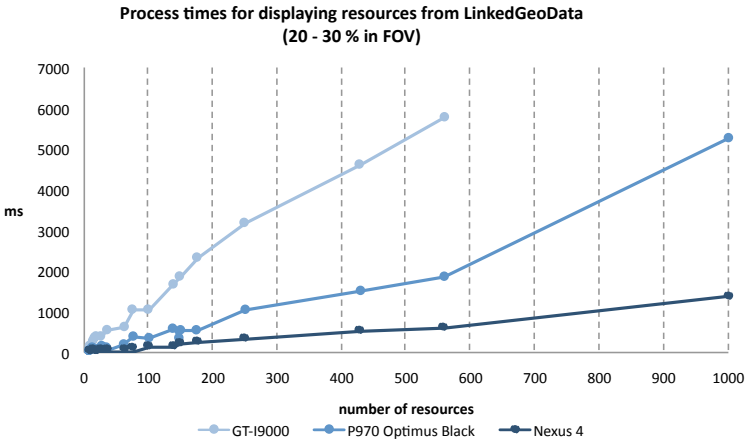

FIGURE 7.11: Average times for calculating screen coordinates and visualizing 20-30% of the resources in the camera's FOV from LinkedGeoData

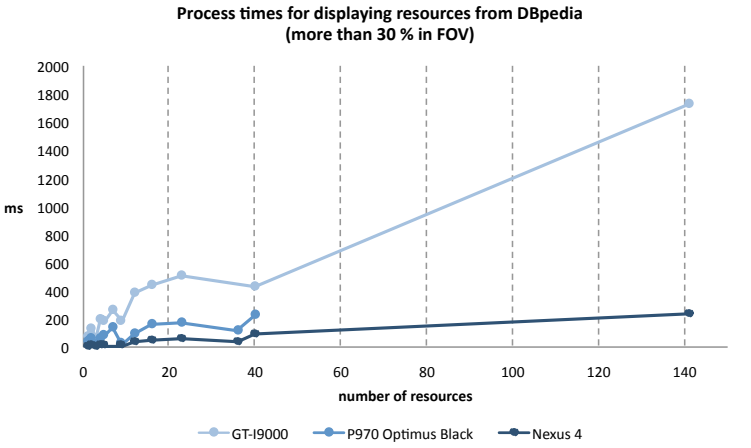

FIGURE 7.12: Average times for calculating screen coordinates and visualizing more than 30% of the resources in the camera's FOV from DBpedia

After presenting the results of testing the processing times of our proof-of-concept implementation, we summarize the most important aspects. Basically, the fluently representation of resources in the camera's field of view depends on several aspects. The better the device's performance is, the more resources can be displayed in a reasonable quality. Moreover, the number of resources that are currently displayed in the FOV affects the processing times significantly. Beside these aspects, we find a difference between representing data from DBpedia and LinkedGeoData, which concludes that the type and amount of information about each resource impacts the calculation and presentation times, additionally. Resources that contain pictures show higher process times. Considering the aspect that in a typical user scenario not more than 50 resources are queried, the resources are represented at all three devices fluently and with reasonable calculation and representation times.

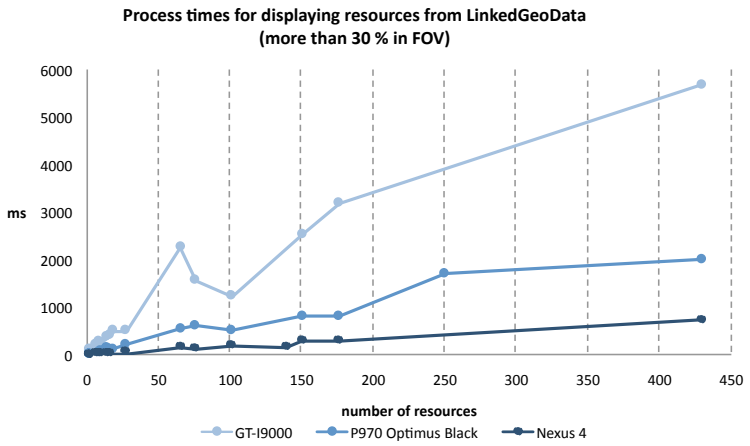

FIGURE 7.13: Average times for calculating screen coordinates and visualizing more than 30% of the resources in the camera's FOV from LinkedGeoData

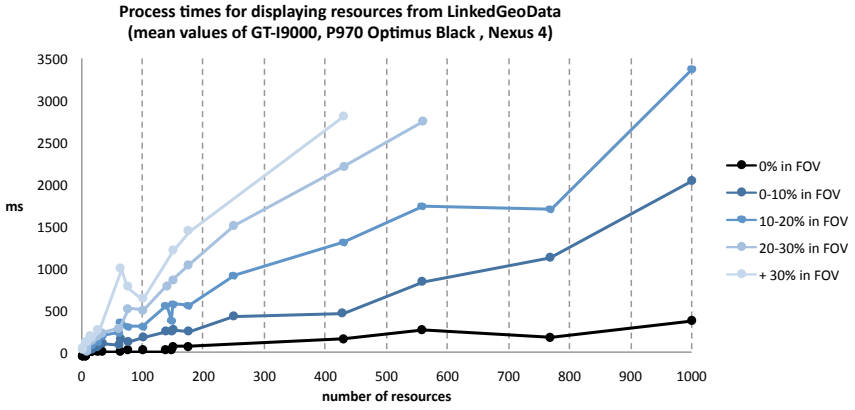

FIGURE 7.14: Average times for calculating screen coordinates and displaying a specific percentage of resources in the FOV

7.2 Problems

In this section, we discuss problems that encountered during the preparation of this master thesis. As already mentioned in Chapter 7.1, LOD sources often provide geographic information about resources that is incomplete or not accurate. Thus, this causes a wrong visualization of the resources' coordinates in the camera's interface. Generally, LOD datasets do not guarantee that the information about the resources is complete and correct. Thus, incomplete information about resources can exist. We had the problem that especially DBpedia does not always provide information about the altitude of a resource. Since the height above the sea level of all queried resources is required for calculating the correct screen coordinates, we need to implement another option for determining the altitude. Due to this, we decided to use the Google Elevation API as well as the web service of Earthtools for determining the height above the sea

level of a resource. As described in Chapter 5.5.3, the web service allows us to query the height above the sea level of any latitude and longitude values. The web service returns a value that describes the altitude right on the ground. However, we do not want that the icon of the resource is displayed at a position directly on the ground. The building has a specific height that shall be considered.

Another problem was that sometimes the web servers from DBpedia or LinkedGeoData throw the error `HTTP error 500 - Internal server error`. This problem affected the implementation and execution of the prototype because it was not possible to send any SPARQL requests. Fortunately, the defect was corrected by no later than two days.

Furthermore, the reception of the GPS signal represented a problem. Sometimes, it was not possible to find a signal or an inaccurate position was returned. As a result, the screen positions of the resources are inaccurate, particularly if the resource is located nearby the camera's position.

7.3 Summary

In this chapter, we analyze three different aspects of the proof-of-concept implementation. In the first section, we compare the data queried by our prototype to data from DBpedia Mobile. Therefore, we test our application as well as DBpedia Mobile at six different places and compare the results. Generally, in our tested scenario the same resources are depicted in our prototype and in DBpedia Mobile. Still, our framework allows to query additional LOD sources and features such as the radius setting are not supported by DBpediaMobile.

Afterwards, we evaluate the accuracy of the data representation in the camera's field of view by focusing on the aspect of the LOD resources' data quality. Thereby, we compare geographic data from LOD resources queried from three different LOD sources to their equivalent position on a Google Maps map. After determining the average deviation of the resource's position, we compared the data from the three different LOD sources to each other. In order to test the accuracy of the resources' positions, we compared 50 positions from DBpedia, 50 positions from LinkedGeoData, and 20 positions from GeoNames. Since most of the resources stretches for several meters, the positions are quite accurate. Generally, geographic information from LOD sources is appropriate for the use in a location-based AR approach.

In the last part of this chapter, we focus on the evaluation of processing times for calculating screen coordinates and representing resources in the camera's field of view. Generally, the times depend on the following aspects: the performance of the mobile

device, the total number of queried resources, the number of resources currently displayed in the FOV, and the kind of data to display. During our measurements, we find significant differences between the processing times of each mobile device. Moreover, the more resources are processed by the framework, the longer the calculation takes. Equally, the more resources are displayed in the current camera interface, the longer the visualization takes. Additionally, the comparison of the processing of data from DBpedia to data from LinkedGeoData implies that the kind of data that is processed makes an impact, too. For instance, if the data contains pictures the processing time lasts longer.

In the last section, we described several problems that occurred during the preparation of this master thesis and explained our solution approaches.

Chapter 8

Conclusions and Future Work

Within this thesis we combine two approaches, Augmented Reality and Linked Data. Although, there already exist some AR applications, none of them uses the concept of Linked Data for accessing data about the resources. All in all, the main objective of this thesis was to develop a mathematical model that combines these two research fields and to implement a framework that visualizes Linked Data about resources in a camera-based AR user interface. Therefore, nearby resources have to be detected and represented correctly in the AR user interface.

Generally, the mathematical model bases on a location-based approach. We decided to only use the geographic coordinates of the camera and the resources as well as the orientation of the camera to calculate the screen coordinates of the resources in the camera's user interface. In order to realize a mathematical model, we access the sensors of a mobile device and query data from LOD databases. The position and orientation data of the mobile device was obtained by using the camera, the GPS, and the following sensors: accelerometer and magnetic field sensor. The position of the resources in the surrounding area was determined by defining SPARQL queries. One important aspect of our framework is that it is expandable. Thus, it can be extended by any LOD source that provides a SPARQL endpoint such as DBpedia or LinkedGeoData. The mathematical model realizes several steps in order to transform the geographic coordinates to actual screen coordinates. First, we calculate the position of the resources relative to the camera's position and set the camera's position to $(0, 0, 0)$. After that, we rotate the coordinates around each axis. After rotating the coordinates, we are able to compute the screen coordinates by applying a perspective projection and transforming the coordinates according to the screen resolution.

In order to prove the usefulness of our approach, we develop an AR framework that realizes our approach. As the framework can be extended by an arbitrary LOD source that offers a SPARQL endpoint, we illustrate the extension of our framework by adding the DBpedia project. The application allows to choose the radius within which the resources will be queried and the database to query. The implemented camera interface

displays the resources that are located in the environment. If the user points the camera towards a resource, an icon indicating the resource's position will appear on the user interface. The icon is clickable and allows to query for further information about the resource. In order to represent information about the resource we integrated the Mobile RDFBrowser Interface. It serves as browser and allows to query much more information about the resources. Thus, we could take advantage of the principle of Linked Data by enabling the user to query LOD databases. Moreover, we decided to implement some additional features that improve usability and exploit the advantages of AR and Linked Data. We implemented a minimap that displays the off-screen objects. With that, the user knows in which direction the resources are located. Besides the integration of the Mobile RDFBrowser Interface that allows querying Linked Data in RDF format, we queried for images and displayed them in the user interface.

Additionally, we evaluated our proof-of-concept implementation and tested the accuracy of geographic data from DBpedia, LinkedGeoData, as well as GeoNames. Since the precision of the data representation in our AR framework depends on the accuracy of the geographic position, too, we measured the average deviation of the resources position queried from LOD sources compared to their equivalent position on an actual Google Maps map. If we consider the fact that the resources stretch for several meters, the positions are quite accurate. Unfortunately, sometimes the positions vary too much to present the resources accurate in the camera's field of view. Moreover, we tested the times for calculating and displaying the resources in the camera's current field of view. Thereby, we focus on all parts that have to be recalculated if the orientation changes such as the coordinate rotation, the perspective projection, the calculation of the screen coordinates, as well as the representation of the resources in the camera preview. This time particularly depends on the amount of resources that is totally processed as well as the number of resources that are presented in the camera's current view area. As we could recognize differences during the representation of resources from DBpedia and LinkedGeoData, the processing time depends on the kind of data, too. Contrary to LinkedGeoData, DBpedia visualized pictures. If we consider a typical user scenario where not more than 50 resources are displayed we retrieved acceptable processing times. The resources can be represented fluently in the live camera preview.

Generally, the AR framework designed in this master thesis is a basic approach for visualizing RDF data in an AR view. It can be extended in various ways. For instance, it is possible to integrate more LOD databases. Thus, more data can be queried. Basically, each LOD database that stores geographic information about objects and provides a SPARQL endpoint can be integrated. A promising LOD source is called revyu[1]. Revyu is a LOD source that allows to create reviews and ratings. In [64], the authors describe the web site. The site uses Semantic Web technologies like RDF and SPARQL. Furthermore, links to other LOD databases such as DBpedia are provided.

[1]Revyu: `http://revyu.com/`

In this master thesis, we could demonstrate that it is possible to use Linked Data for implementing an AR application. The RDF data format enabled a clear and simple data handling. Although, LOD sources offer data that is sometimes inaccurate or incomplete, we think that the data is still properly for the use in an AR application. The LOD sources contain a huge number of geographic positions and descriptions. Moreover, the data sources are interlinked to other LOD sources and data can easily be queried. Generally, the presented framework is a first approach to combine the two research fields of AR and Linked Data. Especially, it highlights the advantages that implicate these two technologies.

Appendix A

Implementation Resources

To this work a CD containing all relevant implementation resources is attached. The CD includes the source files of the AR framework as well as a text file (README.txt) describing installation details.

Additionally, the QR code allows to download a prototype implementation of the AR framework directly to a mobile device. The AR framework processes RDF data from the LOD sources DBpedia and LinkedGeoData.

Appendix B

Curriculum Vitae

Evelyn Hinterramskogler

Student at
University of Vienna, Faculty of Computer Science

Personal

Born	April 3, 1987 in Amstetten, Lower Austria
Nationality	Austria

Education

2010 - present	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, Key course element: Spanish

Professional experience

2012 - present	Software developer at AVAST Software Austria, responsible for mobile development on the Android platform and web development

Bibliography

[1] R. T. Azuma. A survey of augmented reality. *Teleoperators and Virtual Environments 6*, 4:355–385, 1997.

[2] Wikitude. Available: `http://www.wikitude.com/`. [Online. Accessed January 2013].

[3] Layar. Available: `http://www.layar.com/`. [Online. Accessed January 2013].

[4] mixare - Open Source Augmented Reality Engine. Available: `http://www.mixare.org/`. [Online. Accessed January 2013].

[5] Linked Data. Available: `http://www.w3.org/DesignIssues/LinkedData.html`, . [Online. Accessed January 2013].

[6] C. Becker and C. Bizer. Dbpedia mobile: A location-enabled linked data browser. *LDOW2008*, April 2008.

[7] DBpedia. Available: `http://www.dbpedia.org`, . [Online. Accessed January 2013].

[8] Linking Open Data - W3C SWEO Community Project. Available: `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`, . [Online. Accessed January 2013].

[9] How to Publish Linked Data on the Web. Available: `http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/`, . [Online. Accessed January 2013].

[10] The RDF Advantages Page. Available: `http://www.w3.org/RDF/advantages.html`. [Online. Accessed: January 2013].

[11] M. Raubal and I. Panov. A formal model for mobile map adaption. *Location Based Services and Telecartography II*, pages 11–34, 2009.

[12] B. Furht, editor. *Handbook of Augmented Reality*. Springer Science + Business Media, 2011.

[13] P. Milgram and F. Kishino. A taxonomy of mixed reality visual displays. *IEICE Transactions on Information Systems*, E77-D, No. 12, Dezember 1994.

[14] R. Yang. The study and improvement of augmented reality based on feature matching. *IEEE*, pages 586–589, 2011.

[15] J. Carmigniani, B. Furht, M. Anisetti, P. Ceravolo, E. Damiani, and M. Ivkovic. Augmented reality technologies, systems and applications. *Springer Science+Business Media*, 51:341–377, December 2010.

[16] H. López, A. Navarro, and J. Relano. An analysis of augmented reality systems. *Fifth International Multi-conference on Computing in the Global Information Technology*, pages 245–250, 2010.

[17] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, R. Choudhury, and A. T. Campbell. A survey of mobile phone sensing. *IEEE Communications Magazine*, pages 140–150, September 2010.

[18] Alasdair Allan. *Programming iPhone Sensors*. O'Reilly Media, Inc., November 2011.

[19] M. Andrejasic. Mems accelerometers. Seminar, University of Ljubljana, Faculty for mathematics and physics, 2008.

[20] J. Fraden. *Handbook of Modern Sensors*. Springer Science + Business Media, 4 edition, 2010.

[21] Android Developers API Guides. Available: `http://developer.android.com/guide`, . [Online. Accessed January 2013].

[22] Android platform. Available: `http://www.android.com`, . [Online. Accessed January 2013].

[23] W. F. Ableson, R. Sen, and C. King. *Android in Action*. Manning Publications Co., second edition edition, 2011.

[24] M. Zechner. *Beginning Android Games*. Apress, 2011.

[25] Android Developers Reference. Available: `http://developer.android.com/reference`. [Online. Accessed January 2013].

[26] Android platform. Available: `http://developer.android.com/reference/android/hardware/SensorEvent.html`, . [Online. Accessed January 2013].

[27] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Special Issues on Linked Data, International Journal on Semantic WEb and Information Systems*.

[28] Uniform Resource Identifier (URI): Generic Syntax. Available: `http://tools.ietf.org/html/rfc3986`, 2005. [Online. Accessed January 2013].

[29] HTTP - Hypertext Transfer Protocol. Available: `http://www.w3.org/Protocols/`, 2012. [Online. Accessed January 2013].

[30] T. Heath and C. Bizer. *Linked Data - Evolving the Web into a Global Data Space*. Morgan Claypool publishers, 2011.

[31] C. Becker and C. Bizer. Exploring the geospatial semantic web with dbpedia mobile. *Web Semantics: Science, Services and Agents on the World Wide Web*, pages 278–286, 2009.

[32] Resource Description Framework (RDF). Available: `http://www.w3.org/RDF/`. [Online. Accessed January 2013].

[33] Resource description framework (rdf): Concepts and abstract syntax, February 2004. URL `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

[34] T. Segaran, C. Evans, and J. Taylor. *Programming the Semantic Web*. O'Reilly Media Inc., July 2009.

[35] RDF/XML Syntax Specification. Available: `http://www.w3.org/TR/REC-rdf-syntax/`, 2004. [Online. Accessed January 2013].

[36] RDFa. Available: `http://www.w3.org/TR/rdfa-syntax/`, 2008. [Online. Accessed January 2013].

[37] Notation 3 Logic. Available: `http://www.w3.org/DesignIssues/Notation3`, 2011. [Online. Accessed January 2013].

[38] N-Triples. Available: `http://www.w3.org/TR/rdf-testcases/#ntriples`, 2004. [Online. Accessed January 2013].

[39] Turtle - Terse RDF Triple Lanauge. Available: `http://www.w3.org/TeamSubmission/turtle/`, 2011. [Online. Accessed January 2013].

[40] SPARQL Query Language for RDF. Available: `http://www.w3.org/TR/rdf-sparql-query/`, 2008. [Online. Accessed January 2013].

[41] E. Della Valle and S. Ceri. *Handbook of Semantic Web Technologies - Querying the Semantic Web: SPARQL*. Springer-Verlag, 2011.

[42] The Linking Open Data cloud diagram. Available: `http://richard.cyganiak.de/2007/10/lod/`, 2011. [Online. Accessed January 2013].

[43] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. *Proceedings of the 6th International Semantic Web Conference*, pages 722–735, 2007.

[44] Interlinking DBpedia with other Data Sets. Available: `http://wiki.dbpedia.org/Interlinking`, . [Online. Accessed January 2013].

[45] The DBpedia Data Set. Available: `http://wiki.dbpedia.org/Datasets`, . [Online. Accessed January 2013].

[46] LinkedGeoData. Available: `http://www.linkedgeodata.org/`, . [Online. Accessed January 2013].

[47] D. Hearn and M. P. Baker. *Computer Graphics C Version*. 2 edition, 1996.

[48] David Salomon. *Transformations and Projections in Computer Graphics*. Springer-Verlag London, 2006.

[49] David Eberly. Conversion of left-handed coordinates to right-handed coordinates. *Geometric Tools LLC*, 2008.

[50] P. Belimpasakis, P. Selonen, and Y. You. Bringing user-generated content from internet services to mobile augmented reality clients. *Cloud-Mobile Convergence for Virtual Reality Workshop IEEE*, pages 14–17, March 2010.

[51] D. Gray, I. Kozintsev, Y. Wu, and H. Haussecker. Wikireality: Augmented reality with community driven websites. *IEEE*, pages 1290 – 1293, 2009.

[52] S. Gammeter, A. Gassmann, L. Bossard, T. Quack, and L. Van Gool. Server-side object recognition and client-side object tracking for mobile augmented reality. *IEEE*, pages 1–8, 2010.

[53] M. Choubassi, O. Nestares, Y. Wu, I. Kozintsev, and H. Haussecker. An augmented reality tourist guide on your mobile devices. In S. Boll et al., editor, *LNCS 5916*, pages 588 – 602. MMM 2010, Springer-Verlag Berlin Heidelberg, 2010.

[54] T. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. *MIR08 Vancouver ACM*, October 2008.

[55] G. Reitmayr and D. Schmalstieg. Location based applications for mobile augmented reality. *Vienna University of Technology.*

[56] D. Omercevic and A. Leonardis. Hyperlinking reality via camera phones. In *Machine Vision and Applications*. Springer-Verlag, July 2010.

[57] C. van Aart, B. Wielinga, and W. R. van Hage. Mobile cultural heritage guide: Location-aware semantic search. *EKAW 2010, LNAI 6317*, pages 257–271, 2010.

[58] B. Hecht, M. Rohs, J. Schöning, and A. Krüger. Wikeye – using magic lenses to explore geo-referenced wikipedia content. *Proceedings of the 3rd International Workshop on Pervasive Mobile Interaction*, May 2007.

[59] C. Bizer, M.Gaedke, G.Kobilarov, and J.Volz. Silk - a link discovery framework for the web of data. *LDOW*, April 2009.

[60] Calculate distance, bearing and more between Latitude/Longitude points. Available: `http://www.movable-type.co.uk/scripts/latlong.html`. [Online. Accessed: January 2013].

[61] R.W. Sinnott. Virtues of the haversine. *Sky and Telescope*, 68(2):159, 1984.

[62] T. Schinke, N. Henze, and S. Boll. Visualization of off-screen objects in mobile augmented reality. *MobileHCI'10*, pages 313–316, September 2010.

[63] Android Developers. Available: `http://developer.android.com`, . [Online. Accessed January 2013].

[64] T. Heath and E. Motta. Revyu: Linking reviews and ratings into the web of data. *Web Semantics: Science Services and Agents on the World Wide Web*, pages 266–273, 2008.