Media Engineering and Technology Faculty
German University in Cairo

Institute for Visualization and Interactive Systems
Stuttgart University

An expectation-based editing interface for OpenStreetMap

Bachelor Thesis

Author:	Ahmed ElSafty
Supervisors:	Prof. Thomas Ertl
	Dipl.-Inf. Bernhard Schmitz
Submission Date:	31 August, 2013

Media Engineering and Technology Faculty
German University in Cairo

Institute for Visualization and Interactive Systems
Stuttgart University

Universität
Stuttgart

An expectation-based editing interface for OpenStreetMap

Bachelor Thesis

Author:	Ahmed ElSafty
Supervisors:	Prof. Thomas Ertl
	Dipl.-Inf. Bernhard Schmitz
Submission Date:	31 August, 2013

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

Ahmed ElSafty
31 August, 2013

Abstract

Building an open-source world map was one of the main reasons OpenStreetMap (OSM) was founded. Over 1.3 million contributors participate in editing the the world map collaboratively. Unfortunately, there is no support or any assistive technology solutions that helps blind and visually impaired users to blend into the OSM community. The aim of this thesis is to provide them with an assistive OSM editing application with an adaptive user interface that matches their needs. A mobile application for OSM editing was developed with an assistive recommendation system that helps predicting changes users might need to commit. The thesis describes in details the application design, decisions made, workflow and modularity.

Acknowledgments

First of all, I would like to thank my supervisor Dipl.-Inf. Bernhard Schmitz for his assistance, priceless guidance and advice. Also, I would like to thank the Visualization and Interactive Systems department (VIS) and the International center (IZ) in Stuttgart Universität for providing me with a good environment and needed facilities to complete this project.

I would like to express my deep gratitude to Jolly for her continuous support and invaluable motivation to continue this bachelor thesis, I couldn't have done it without you. Finally, I would like to acknowledge the tremendous sacrifices my parents made to ensure that I had an excellent education. For this and much more, I am forever in their debt.

Contents

Chapter 1

Introduction

1.1 Motivation

From the beginning of time, travelers and wanderers relied heavily on landmarks, trends and even the locations of the stars to identify their trading routes. In order to transfer knowledge and information from one generation or culture to another, cartography was needed. Cartography, or the art of map creation, has been playing a substantial part in human history. In the last decades, commercial mapping corporations have been building up their credibility, selling their maps with high costs and limiting the use of the data by copyrights and licences.

Building an open-source world map was one of the main reasons OpenStreetMap (OSM) was founded. In April 2006, the OpenStreetMap foundation was established to encourage the development and utilization of free geospatial data to provide it for anybody to use and share. A great number of participants (over 1.3 million) contribute in editing the world map collaboratively using the OSM infrastructure, and the number of contributors is growing every month [24].

However, there is a group of contributors who haven't been involved yet in OpenStreetMap. In 2012, the World Health Organization estimated that there were approximately 39 million people who are blind and 285 million who are visually impaired worldwide [32]. Unfortunately, there is no support or any assistive technology solutions that helps them blend into the OSM community.

1.2 Aim

The aim of this thesis is to provide blind and visually impaired users with an assistive OpenStreetMap editing application with an adaptive user interface that matches their needs. The application has to be mobile so that users can edit OSM features on the fly while navigating in an environment. Also, it has to include a logical and sensible recommendation system that acts as an assistive solution for blind users which helps predicting changes users might need to commit.

1.3 Thesis Structure

The thesis is divided into the following chapters:

- *Chapter 2* gives a short background description on technologies used in the thesis, including OpenStreetMap, XML parsers, semantic web, the Resource Description Framework and the SPARQL query language.
- *Chapter 3* gives an overview of the main research topics used, including context-aware recommendation systems, OpenStreetMap editing applications and user interface for blind people. Also, it provides brief notion on related work and currently implemented applications.
- *Chapter 4* introduces the system design and decisions that were taken throughout the thesis along with the design workflow.
- *Chapter 5* gives an introduction on the most considerable technical scenarios implemented in the thesis, including authentication process, implementation of the recommendation system, the boundaries detector and more.
- *Chapter 6* provides a summary and final conclusion as well as discussing future work.

Chapter 2

Background

This chapter will describe the background of the technologies used in this thesis. It will start by giving an overview of OpenstreetMap . Thereafter, XML and parsers used in this approach will be demonstrated along with the Semantic Web, Resource Description framework (RDF) and related examples. Finally, the SPARQL query language on triple databases is explained.

2.1 OpenStreet Map

OpenStreetMap (OSM) aims to create a free world map and freely licensed geographic data. Data is being collected by contributors about roads, railways, rivers, forests, houses and everything else that is commonly seen on the world map. OSM provides editing tools that allow users to add, update or delete geographical features through an intuitive and easy interface. A great number of participants (over 1.3 million) edit the world map collaboratively using the OSM infrastructure [24]. OSM data is available for further use across different application domains, software platforms and hardware devices.

OpenStreetMap was founded in July 2004 by Steve Coast. In April 2006, the OpenStreetMap Foundation (OSMF) was established to encourage the growth, development and distribution of free geo-spatial data and provide it for anybody to use and share. In December 2006, Yahoo confirmed that OSM could use its aerial photography as a backdrop for map production [7].

OpenStreetMap data can be exported in different formats. Tiles can be downloaded as OSM data, PostGIS[1] or shapefiles then rendered using a software called **Mapnik**[2].

[1] More info http://postgis.net/

[2]Mapnik is an open source toolkit for rendering maps. It is used to render the four main Slippy Map layers on the OpenStreetMap website. It supports a variety of geospatial data formats and provides flexible styling options for designing many different kinds of maps. more info http://mapnik.org/

Moreover, raw data can be downloaded in XML format As shown in the XML document below:

Listing 2.1: OpenStreetMap XML output structure

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
 <bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913
 900" maxlon="12.2524800"/>
 <node id="1831881213" version="1" changeset="12370172" lat="54.
 0900666" lon="12.2539381" user="lafkor" uid="75625" visible="tr
 ue" timestamp="2012-07-20T09:43:19Z">
  <tag k="name" v="Neu Broderstorf"/>
  <tag k="traffic_sign" v="city_limit"/>
 </node>

 <way id="26659127" user="Masch" uid="55988" visible="true" vers
 ion="5" changeset="4142606" timestamp="2010-03-16T11:47:08Z">
  <nd ref="292403538"/>
  <nd ref="298884289"/>
  <nd ref="261728686"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Pastower Strae"/>
 </way>

 <relation id="56688" user="kmvar" uid="56190" visible="true" ve
 rsion="28" changeset="6947637" timestamp="2011-01-12T14:23:49Z"
 >
  <member type="node" ref="364933006" role=""/>
  <member type="way" ref="4579143" role=""/>
 <tag k="name" v="Kstenbus Linie 123"/>
  <tag k="network" v="VVW"/>
  </relation>

</osm>
```

OSM represents its data as a list of nodes, ways and relations (also known as **elements**). Each **node** defines a geospatial point using its longitude, latitude and a set of features that define the node. A town, mall or shop can be represented as a node, with tags like *amenity=shop* to give the node more characteristics.

The second primitive element is the **way**, which is a list of two or more nodes. Ways can be used to represent linear features like roads and ways, or polygons like buildings and areas using distinct tags, i.e, *building=yes*.

Finally, the **relation** groups a set of nodes, ways and even other relations to describe a certain connection between them. For example, a bus route could be described as a relation of *type=route*, *route=bus* and *operator=** tags containing the ways over which the bus travels, along with the bus stop nodes.

2.2 XML

XML stands for“e**X**tensible **M**arkup **L**anguage”. Its a “markup” language i.e, a language that contains special ”markup” elements to describe the structure and format the document. However, unlike most markup language, XML is “eXtensible”: the elements used in the document are not fixed, giving XML its flexibility to store data in a clean structured way as it allows the user to define his own tags [9].

Although XML design focuses on documents, it is widely used for the representation of arbitrary data structures. A Simple XML document would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<name>
       <first>John</first>
       <last>Snow</last>
</name>
```

Even from this simple example, it is clear why markup languages like XML are called “self-describing”. The first line is called the XML declaration. It contains information about which version is in use. It also contains information regarding the character encoding of the document. Looking at the data, information can be interpreted with ease. A tag called <first> with parent node <name> intuitively resembles first name, the same is true with the last name tag. Thus, it is advised to use meaningful tag names when creating XML documents, while following a proper format and choice of wording throughout the whole document [15].

XML documents have to satisfy few syntax rules, as some of them are shown in the example. A single “root” element contains all the other elements, i.e, the name tag is the root containing first and last names. Also, beginning and ending tags should be correctly nested, with none missing or overlapping. This allows computer programs to read the XML document and easily tell information from its markup, an XML ’processor’ is commonly called a *parser* [15].

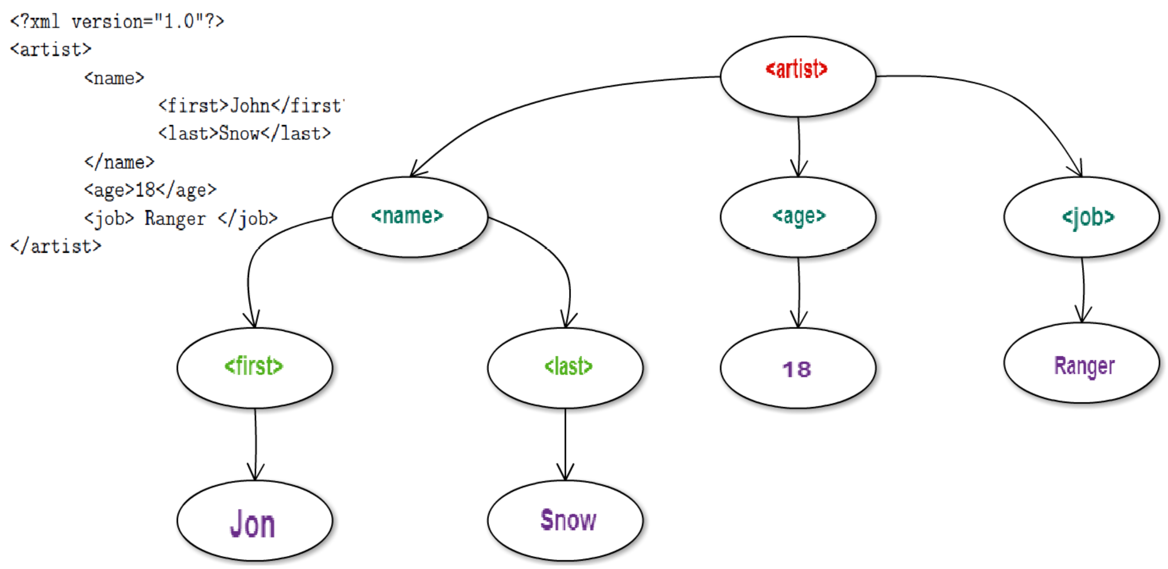
2.3 XML Parsers

XML parsers simply read XML documents and provide the application with any information it needs. It reads through the characters in the document and determines which represents markup characters and which are data [15]. There are two main parsers which are included in android. Each parser has its pros and cons. They are called SAX and DOM parsers:

DOM Parser

DOM stands for **D**ocument **O**bject **M**odel. DOM builds an in-memory tree representation of the XML document. Generally it is an application for validating XML Documents. [20] It defines the objects and properties of all elements in xml and the methods to access them, where the elements and attributes are represented as nodes in a hierarchical tree structure as shown in figure 2.1.

Each node stores the element's name and attributes along with pointer information indicating the parent-child-sibling relationship and the node value. Although it is memory inefficient, this memory structure provides DOM with its basic feature, random access. Unlike SAX sequential parsing, DOM provides fast random access to any and every node, due to its tree hierarchical structure [12].



```
<?xml version="1.0"?>
<artist>
        <name>
                <first>John</first
                <last>Snow</last>
        </name>
        <age>18</age>
        <job> Ranger </job>
</artist>
```

Figure 2.1: DOM parser workflow

However, DOM has one major flaw: the data in the tree can only be accessed when the tree is finished i.e the parser is done with parsing the whole XML document. Thus, complex and large XML documents will not be available anytime before the needed memory and space are allocated and the parsing is complete [31].

SAX Parser

Sax stands for **S**imple **A**PI for **X**ML. Sax is an event based XML parser, as it parses XML documents step by step. It is considered a light-weight and fast parser with low memory consumption when compared to a DOM parser due to the fact that, instead of creating hierarchical structure representation of all elements and maintaining pointers to indicate the parent-child-sibling relationship between them, it fires when it encounters opening tag, element or attribute and work accordingly. When a certain event is triggered, it calls the corresponding function to handle the event as shown in figure 2.2.

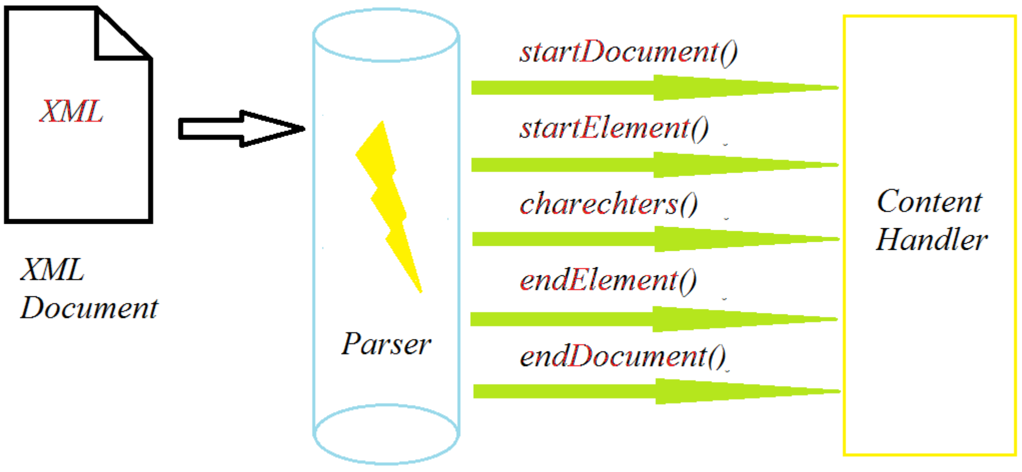


Figure 2.2: SAX parser workflow

A SAX parser is recommended with large XML files that do not require frequent modification, because SAX doesn't require to load the whole XML file in Java if the application needs to parse specific parts in the XML document. Also, the memory consumption is not affected by XML document size since the objects associated with the triggered events could be destroyed on a regular basis. Moreover, it supports partial data access before the XML document parsing is done [12].

However, DOM is more powerful. It is more suitable for complex and frequent updates despite its high memory consumption. Since a DOM parser uses a hierarchical tree structure for the data representation of the XML document, it is much easier to add, modify or delete nodes in the DOM tree. This could be achieved by pointer maneuvers between the tree nodes for fast editing, insertion and deletion. On the other hand, SAX is more useful with applications with limited memory that require simple or rare modifications [12].

2.4 Semantic Web

The semantic web - as W3C stated - is a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. The semantic web has many applications. Examples include data integration, knowledge representation and analysis, cataloging services, improving search algorithms and methods, social networks and more. In order to share data across different applications, it was substantial to define and describe relations among data. Thus, The Resource Description Framework was used to depict and delineate these connections [13].

Resource Description Framework

The **R**esource **D**escription **F**ramework (RDF) is a data model that was proposed by the World Wide Web Consortium (W3C) as a standard for representing metadata about semantic data. [14] RDF was mainly designed to express and exchange information about Web resources. Moreover, the framework is designed to be read and interpreted by computers, which allows computers to self-integrate information from the web. RDF is one of the main pillars of the so-called Linked Data Web, known nowadays as the Semantic Web [5].

RDF describes the relationship of information among resources using a set of statements. An RDF statement is represented by a triple, consisting of a subject, its predicate and Object. The triple can be simply interpreted as a subject having a specific relationship with its object [14]. For example, “Leopard” would be the Subject, “eat” would be the relation and “meat” would be the Object in a “Leopard eat meat” RDF statement.

- Subject: is anything that can have a URI, such as “http://www.informa tik.uni-stuttgart.edu/RDF/”
- Predicate: is a string that describes the relation between a subject and its object.
- Object: is the data value of the Subject, it could be a literal value such as “study” or a URI as in “http://www.informatik.com”

URI is short for Uniform Resource Identifier. It is a string of characters that acts as a name or a locator for a resource. It is used to enable interaction with representations of a resource over the World Wide Web using specific protocols. [30]

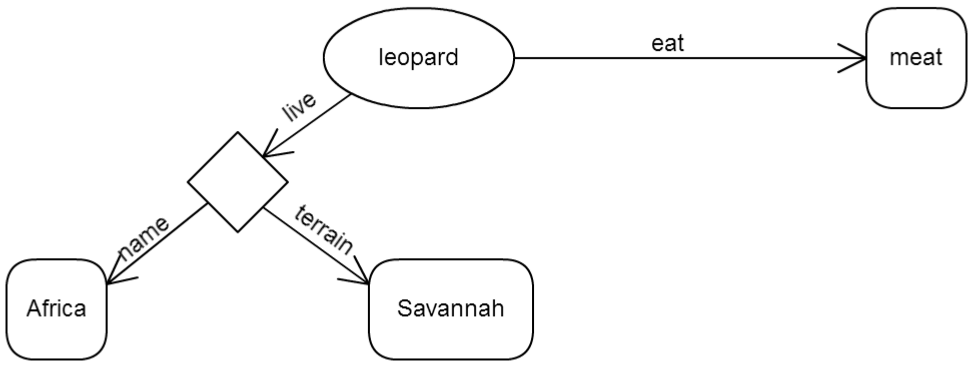


Figure 2.3: RDF graph example

As shown in figure 2.3, RDF can be represented in a form of directed graph data model, RDF uses a graph data model, where different entities are vertices in the graph and relationships between them are represented as edges. Information about an entity is represented by directed edges emanating from the vertex for that entity, where the edge connects the vertex to other entities, or to literal that represents the value of a particular attribute for that entity. All the information provided builds up the RDF vocabulary also known as RDF *schema* [14] the following example shows how to represent information in RDF XML:

Listing 2.2: RDF database example

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns\#"
xmlns:cd="http://www.recshop.fake/cd\#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
  <cd:artist>Bob Dylan</cd:artist>
  <cd:price>10.90</cd:price>
  <cd:year>1985</cd:year>
</rdf:Description>
</rdf:RDF>
```

Few notes can be easily interpreted from the above XML Document:

- The first line is the XML declaration. The XML declaration is followed by the root element of RDF documents: <rdf:RDF>.
- The xmlns:rdf namespace, specifies that elements with the RDF prefix are from the namespace "http://www.w3.org/1999/02/22-rdf-syntax-ns".

- The <rdf:Description> element contains the description of the resource identified by the rdf:about attribute.
- The elements <cd:artist> and <cd:price> are among properties of the resource.

The following can be concluded from the above RDF document: The album is Empire Burlesque by Bob Dylan, 1985 edition and it costs 10.90 unit currency.

Subject	Predicate	Object
http://www.recshop.fake/cd/Empire Burlesque	cd:artist	Bob Dylan
http://www.recshop.fake/cd/Empire Burlesque	cd:price	10.90
http://www.recshop.fake/cd/Empire Burlesque	cd:year	1985

Table 2.1: Triples in data model

Most RDF-triple-stores (also known as RDF database) represent the graphs as a table of triples. The Triples take the form <subject,predicate,object>. Many RDF-stores uses a single relational table containing three columns to store the triples as illustrated in table 2.1. A group of relations form an *ontology* [14].

An *Ontology* - as Gruber clarified - is "a specification of conceptualization" or " a set of representational primitives with which to model a domain of knowledge or discourse". It is used for classifying terms used in particular application or group of applications, identifying possible relations and defining constraints on such relationships. The primitives used for description are usually concepts, properties and relationships. Ontologies are used in the semantic web, artifical intellegence, system engineering or in our case, it can be used for expectation based and simple recommendation systems. RDF is one of the languages used to describe such ontologies [11].

2.5 Java RDF libraries

There are few Java RDF libraries for modeling RDF datasets. KAON (Karlsruhe ontology) - for example - is an ontology infrastructure developed by the University of Karlsruhe and the Research Center for Information Technologies in Karlsruhe. It provides an API for OWL ontologies, a module for extracting ontology instances from relational databases and more. However, it strays from pure standards with their own special syntax and extensions [22].

The Jena and Sesame Framework share the same features. However, according to the Berlin SPARQL Benchmark Results, It takes 90 minutes to load 100 million triples of data using Jena framework and 3 days 6 hours to load them using Sesame's [6]. Moreover, Jena framework has the best support and widest community of the three frameworks, growing everyday. Therefore, Apache's Jena RDF framework was picked for our approach in this thesis.

Apache Jena Framework

Jena is a Java framework for building semantic web applications. It provides programmatic environment for RDF, RDFS (schema), OWL (Web Ontology Language), SPARQL (Sparql Protocol And RDF Query Language) and includes a rule based reference engine [17]. It has a huge library that allows developers to create, manipulate and execute RDF graph queries. The Jena framework includes:

- An API for reading, processing and writing RDF data in XML, N-triples and Turtle formats.
- An ontology API for handling OWL and RDFS ontologies.
- Stores to allow large numbers of RDF triples to be efficiently stored on disk.
- A query engine compliant with the latest SPARQL specification.
- Servers to allow RDF data to be published to other applications using a variety of protocols, including SPARQL.

Jena has object classes to represent graphs, resources, properties and literals. Also, a graph is called a model and is represented by the Model interface. Basically, an empty model is initialized, thereafter - for example - the leopard resource is created and a property added to it. [28]

```
Model model = ModelFactory.createDefaultModel();
Resource leopard = model.createResource(leapardURI).addProperty
(live, Africa);
```

There are various ways to extract the data from RDF model to be used in the application. One way is to iterate over all the triples in the model and extract the data. However, this iteration technique would be tiresome if the model had different or distinct triples. As mentioned earlier, the triple-store acts like a triples database, so a more flexible solution would be to use queries to educe the data. Luckily, Jena framework has a built-in Query engine called ARQ that supports the SPARQL RDF Query Language.

2.6 SPARQL

As explained in the official W3C Recommendation documentation for SPARQL [27], most forms of SPARQL queries contain a set of triple patterns called a basic graph pattern. Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable. A basic graph pattern matches a sub-graph of the RDF data when RDF terms from that sub-graph may be substituted for the variables. Hence, executing SPARQL queries generally involves graph pattern matching [25].

SPARQL is an RDF query language, it is a recursive acronym which stands for "SPARQL Protocol and RDF Query Language". SPARQL supports triple patterns and optional graph patterns along with their conjunctions and disjunctions. The results of SPARQL queries can be results sets or RDF graphs [27].

An RDF dataset that shows information regarding Ludwig van Beethoven is shown below. Ludwig van Beethoven is the Subject, placeOfBirth is the predicate and Bonn, Germany is the literal Object. The prefix notation followed is called Notation3, the complex URIs like "http://dbpedia.o rg/property/place OfBirth" can be divided into user-defined prefixes dbpprop: <http://dbpedia.org/property/> and then prefix dbpprop is concatenated with placeOfBirth to shorten the complex URI to just dbpprop:placeOfBirth in what known as QNames [4].

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX dbres: <http://dbpedia.org/resource/>

dbres:Ludwig_van_Beethoven dbpprop:placeOfBirth "Bonn,Germany".
```

SPARQL queries are simple in concept, they match a triple like < ludwig,place,germany> to <x,y,z>. If the developer wants to know the birthplace of Beethoven, the following query should be used in the SPARQL Endpoint:

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX dbres: <http://dbpedia.org/resource/>

SELECT ?place
WHERE
 {
 dbres:Ludwig_van_Beethoven dbpprop:placeOfBirth ?place .
 }
```

This example tries to find entities in the data set that have at least one edge emanating from them, labeled "placeOfBirth" and connected to the "Ludwig van Beethoven" entity. Entities that match this pattern are referred to using the variable name ?place (the ? character is used to indicate pattern variables). For these entities, SPARQL looks for matched patterns and returns the edge if it exists.

The SPARQL syntax is pretty simple and self-explanatory like MYSQL's. Both share common clauses like Where, Select, Filters and Groupby. Select clause has the output entities the user would like to see , Where clause has the requested query pattern(s), Filter clause adds more restrictions and constrains to the output results.

A more complex example could be that the user wants to know all Composers for piano who were born between 1784 and 1884, as shown in the query below. First, SPARQL tries to match the first predicate that gets all composers. Then it joins the results of the previous query with the current one, for each composer it tries to match the second predicate and get the date of birth, then applies the filter.

```
SELECT ?composer ?dateofbirth
WHERE
 {
 category:Composers_for_piano dcterms:subject ?composer.
 ?composer dbpedia-owl:birthPlace ?dateofbirth.
 FILTER(xsd:dateTime(?birth) >= "1784-01-01T00:00:00Z"^^xsd:dat-
 eTime &&
       xsd:dateTime(?birth) <= "1884-00-00T03:00:00Z"^^xsd:date-
       Time) .
 }
```

The nature of SPARQL - as seen in earlier examples - allows second, third and Nth-levels of pattern matchmaking, making it one of the optimal methods for relations discovery, or in our approach, a reasonable solution for recommendation and preferences.

Chapter 3

State of the Art

Our research is positioned at the intersection between Context-aware Recommendation Systems, OpenSteetMap Editors and User Interface for blind users. This chapter provides a short overview of these areas and related work.

3.1 Recommendation System

"The smaller the difference between what a client wants and what he gets, the better the quality of the service. The value co-created by the pair provider-client through the use of services has received great attention from researchers in the past few years." [18].

Recommendation systems are implemented to predict the 'rating' and 'preferences' that users would give to a specific item such as music records, books and movies. Also, to predict connections of people and points of interests when it comes to social networks [26].

Recommendation systems have become extremely common in recent years. Netflix suggests movies that a user might like to watch based on the user's previous ratings and watching habits. Pandora Radio picks up an input of a song or musician and plays music in the same genre with similar characteristics. Amazon.com will recommend additional items based on what other shoppers bought along with the currently selected item [26].

In general, recommendation systems can use very simple and basic data, such as user ratings/evaluations for items, also known as knowledge poor recommendation systems. Different techniques involves using ontological information regarding users, items, constraints , social relations or activities. These techniques are more knowledge dependent, and offers accurate preferences predictions and forecast. At any rate, data used by recommendation systems circles around the relations between users and items [26].

Recommendation systems are made to produce a list of alternatives and recommendations using one of two ways, collaborative or content-based filtering.

Collaborative filtering

Collaborative filtering is based on analyzing and crowdsourcing large amounts of data and information on users' activities or preferences and predicting what users are likely to choose based on what other similar users already picked. One advantage of using collaborative filtering is that it relies on user similarities and therefore it does not need any understanding of the item itself. Many algorithms are used to measure user-item similarities in recommendation systems, K-nearest Neighbour approach [8], Pearson Correlation [10] and more.

Content-based filtering

Content-based filtering is based on information and the characteristics of the items which will be recommended. Different algorithms regarding information filtering and retrieval try to suggest items related to what the user liked, shared, bought or searched about in the past as shown in figure 3.1.

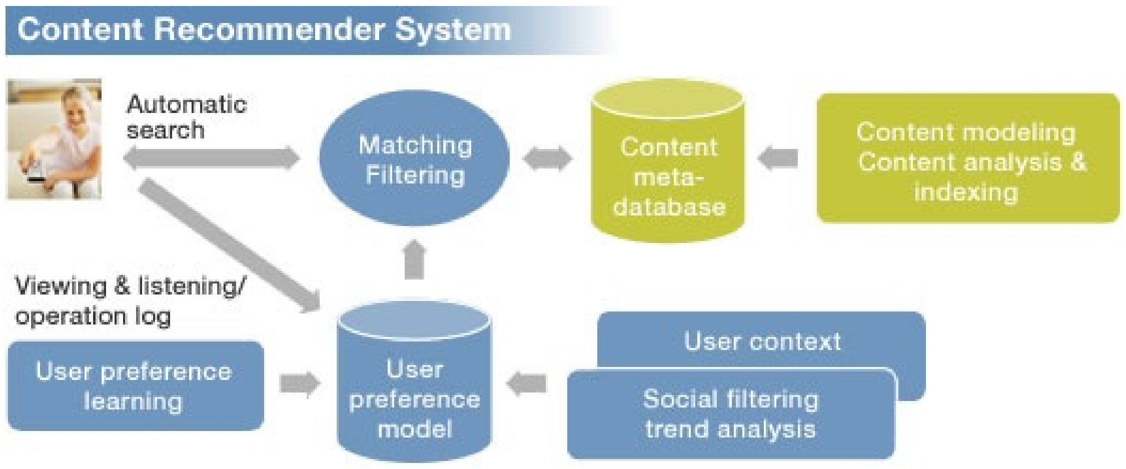

Content Recommender System
Automatic search
Matching Filtering
Content meta-database
Content modeling Content analysis & indexing
Viewing & listening/ operation log
User preference learning
User preference model
User context
Social filtering trend analysis

Figure 3.1: Content-based filtering search recommender (taken from Sony Technologies - content recommender system[1])

Basically, Items are "broken" down to set of features. The system creates a model based on how important the item feature is. The weight of the feature can be determined by simple approaches like rated or boolean questions. It can also be determined by complex techniques such as decision trees and artificial neural networks to guess the likelihood of the user picking a specific item [19].

[1] "Http://www.sony.net/SonyInfo/technology/technology/theme/contents_01.html"

Related Work

Many research and industry efforts are focusing on location and context aware resource discovery in mobile computing. Ontology-based annotations have been considered as a tool for the implementation of recommendation systems because of its properties and flexibility. However, the main challenge is to provide paradigms and techniques that are efficient, resilient and intuitive enough to be of practical interest for a potentially wide user base.

Noppens et al. [23] implemented a mobile application for semantic-based mobile service discovery. His aim was to create a flexible graph-based representation that allows OWL ontology browsing. However, preference specification required a rather long interaction process, which could be unwieldy on mobile devices due its limited capabilities in hardware and user interface.

Beckar and Bizer [3] introduced a mobile application, which allows a user to search for resources located nearby, using data extracted from DBpedia or any Linked Open Data (LOD). The system also allows users to publish pictures and reviews that add more details to the point of interest. Moreover, the user may use SPARQL queries to narrow down search results on the map. Yet, users should have prior experience in SPARQL syntax and how to build its queries.

Auer et al. [1] presented an open source framework, that permits the OpenStreetMap data to be translated and imported to an Open Linked Data repository in RDF. Consequently, as mentioned in Chapter 2, users were able to use the SPARQL RDF query language at a public endpoint on the Web, in order to retrieve geodata of a particular region. However, no semantic matchmaking or SPARQL-aid was offered.

Ballatore et al. [2] have developed an OSM Wiki Crawler, that extracts a semantic network from the OSM wiki website in the form of an RDF graph as shown in 3.2. Moreover, co-citation algorithms were used to find similarities in a graph of inter-linked objects, based on the intuition that similar objects are referenced together. Then the RDF result was imported to a Linked Open Data repository and accessible for SPARQL querying.

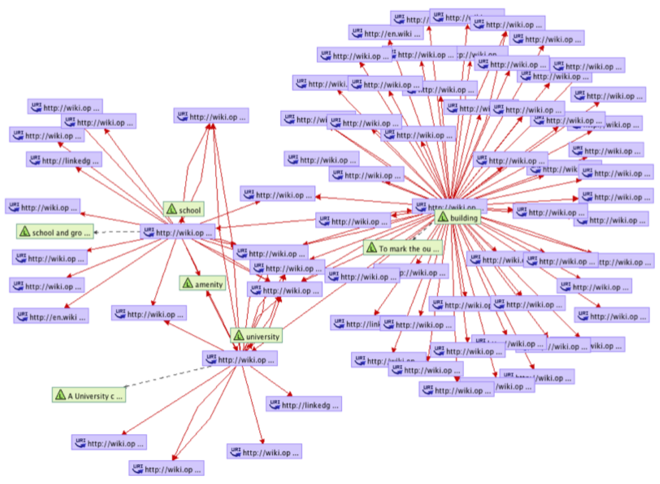


Figure 3.2: A subset of the OSM Semantic Network with concepts ‘amenity’, ‘university’, ‘building’ and ‘school’ (Taken from OSM semantic network[2])

On the other hand, due to the the fact that OSM is open for huge amount of contributors, OSM rules and syntax are not followed from time to time, and there are no restrictions by OSM servers, i.e, anyone can add any tag or value, the Wiki and XML data might have tons of redundancies waiting to be resolved by OSM developers.

3.2 Openstreetmap Editor

OpenStreetMap editors are designed to manipulate OSM geodata. Editors allow users to add roads, pubs or change features of streets, buildings and points of interest. There are several editors already implemented to edit OSM features. However, none was implemented for the sake of visually impaired users or with a blind-friendly user interface.

Vespucci

Vespucci 1.0 was one of the first mobile platform applications to ever allow users to edit OpenStreetMap on android devices. It allows them to create and edit new nodes, ways, tags, append nodes to existing ways, edit geometries and more. However, it has no support for relations editing [21].

ShareNav

ShareNav is a cross platform open source navigation application, tracker, map editor and map viewer software. The first version of ShareNav is based on GpsMid code version

[2]Http://wiki.openstreetmap.org/wiki/OSM_Semantic_Network

0.8.2 (which does not differ much from ShareNav, except it has support for non smart phones). It has even less editing capabilities than Vespucci as it does not allow users to edit geometries. However, it has more navigational features like GPS point-to-point guidance and more [16].

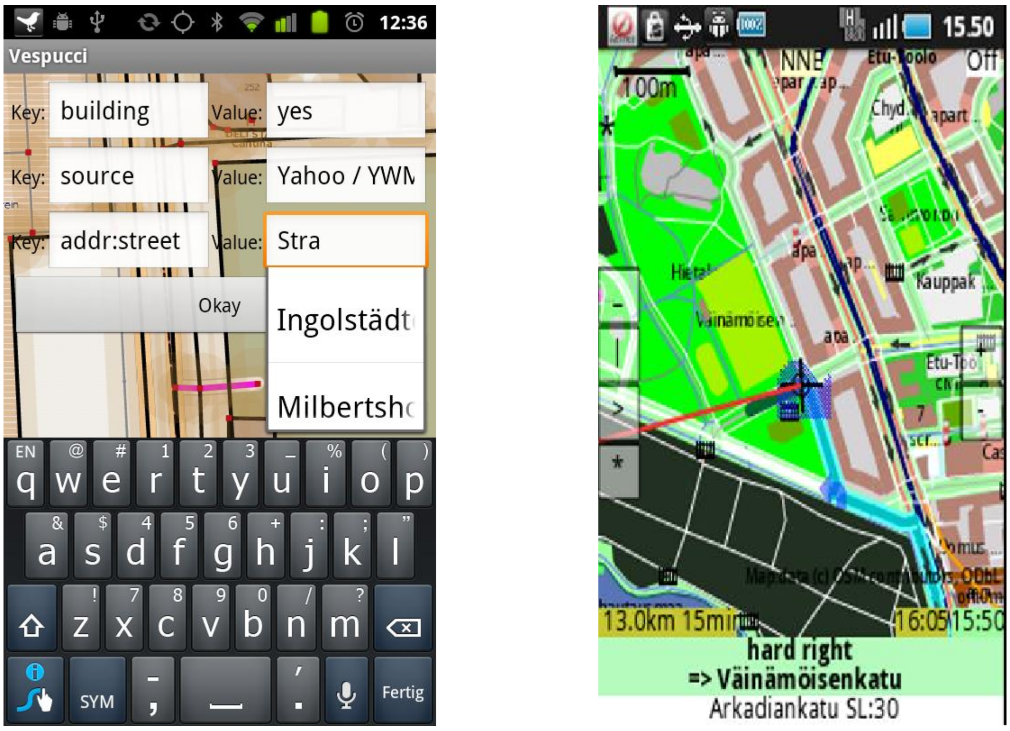

Key: building Value: yes
Key: source Value: Yahoo / YWM
Key: addr:street Value: Stra

Figure 3.3: Screenshots from Vespucci and ShareNav

OSMapTuner

OSMapTuner is an OSM editable application available for android. Its main and only feature is editing existing POI and arbitrary tags of existing OSM objects. No POI addition, no geometry or relations editing.

There are tons of OSM editing applications with support to Windows, Linux, OS X, android and IOS. Most share common features, some have unrelated extras, e.g, using the app as a navigator and not as an editor, and much more. However, as mentioned earlier, blind users did not have the chance to contribute to OpenStreetMap, as there are no editing interfaces implemented in a way that matches their needs.

3.3 Blind Friendly User Interface

In 2012, the World Health Organization (WHO) estimated that there were approximately 39 million people who are blind and 285 million who are visually impaired in the world who face many challenges that arise in daily life. Therefore, assistive technology solutions, especially regarding mobile devices, are needed to overcome these daily challenges in a reliable, easy-to-use, functional way [32].

One of the most obvious yet credible solutions is the Text-to-Speech (TTS) engines. Speech synthesis artificially converts normal language text into speech. TTS proved to be one of the most fruitful user interface feature for visual impaired users. Also, the use of widened buttons that covers equal squared portions of the layout and exploits every available space on screen, greatly avoid miss-clicks and triggering unintended events. Finally, the use of magnifiers, mobile built-in vibration feature and dialog alerts for verification helps blind users finish their intended tasks with as little confusion as possible [29].

In the modules shown above, reliable technologies have been introduced regarding recommendation systems, OSM editing applications and functional techniques to make life easier for blind users. However, there are no applications that get blind and visual impaired users involved in OSM editing with the assistance of recommendation systems or without.

Chapter 4

Design

Before implementation, a schema was needed to make the application as assistive as possible for blind and visually impaired users. This chapter will introduce the approach followed, the decisions made and the design workflow. Also, it discusses the user interface design and gives a simple application walkthrough.

4.1 Approach

This section will introduce the system design and decisions that were taken throughout the thesis duration along with the design workflow. Designing an OpenStreetMap editing application for blind users involves three main parts, a core editing application, a simple recommendation system and a blind-friendly user interface.

4.1.1 Designing the core editing application

The first step towards building the editing application is the core editing interface for OpenStreetMap using its API. Thus, the application proposed was implemented to fulfill these goals:

- The user should be able to sign in with his username and password to the OSM in order to authenticate his editing activities.
- The user should be able to get all nearby Points of Interest around him in a specific radius.
- The user should be able to select, add, edit or delete any tag, key and value of any Point of Interest around him.
- The user should be able to add a new node at his current location, longitude and latitude, in the OpenStreetMap database.
- The user should be able to save and upload his changes to the OpenStreetMap database.

4.1.2 Designing the recommendation system

One of the aims of this thesis is to make the application as assistive as possible to blind and visually impaired users. Thus, users are given a list of preferences and recommendations he can choose from when editing or creating a new Point of Interest. Consequently, the list should be determined by few rules:

- The recommendation list has to be linked and related. When the user edits or creates a new tag and value in a point of interest, the recommendation list should be modified to match the related **existing** tags inside the POI. The more the user keeps feeding data and further enrich the POI, the more specific and accurate the recommendation gets.
- The recommendation list has to be rational and sensible. e.g, a tree doesn't grow inside a house or a park doesn't exist within a crossroad. Thus, every tag inside a POI should have its "unrelated" tags removed from the recommendation list.
- If the user wants to add a new POI within boundaries, e.g, building or park, the tags of the boundaries are added with its related tags to the recommendation list, and any invalid tags are excluded. This can be achieved using the current location and a simple geometry library to determine whether the user is located within a closed area or not.
- A list of recommended values are offered whenever the user is editing a specific key in a tag.

As mentioned in 2.6, the nature of SPARQL allows second, third and Nth levels of pattern matchmaking, making it one of the optimal methods for relations discovery. Therefore, SPARQL was used to query the model of expectations (see section 5.3).

4.1.3 Designing a blind-friendly user interface

The proposed application aims to make life easier for blind and visually impaired users. Thus, most features mentioned in 3.3 should be implemented in a sensible and intuitive way. Such features include:

- The use of wide buttons that fill the screen in equal rectangular areas, seven wide buttons per layout to avoid misclicks and triggering unintended events.
- A text-to-speech engine to facilitate and simplify utilizing the application. A side swipe will tell the user more information about every button in the layout.
- Changing scroll views to view flippers with next and previous buttons, as scrolling wouldn't be appropriate for blind users.
- Exploiting vibration features and alert dialogues to constantly make sure that the user is doing his intended task correctly with the least confusion possible.

4.1.4 Design workflow

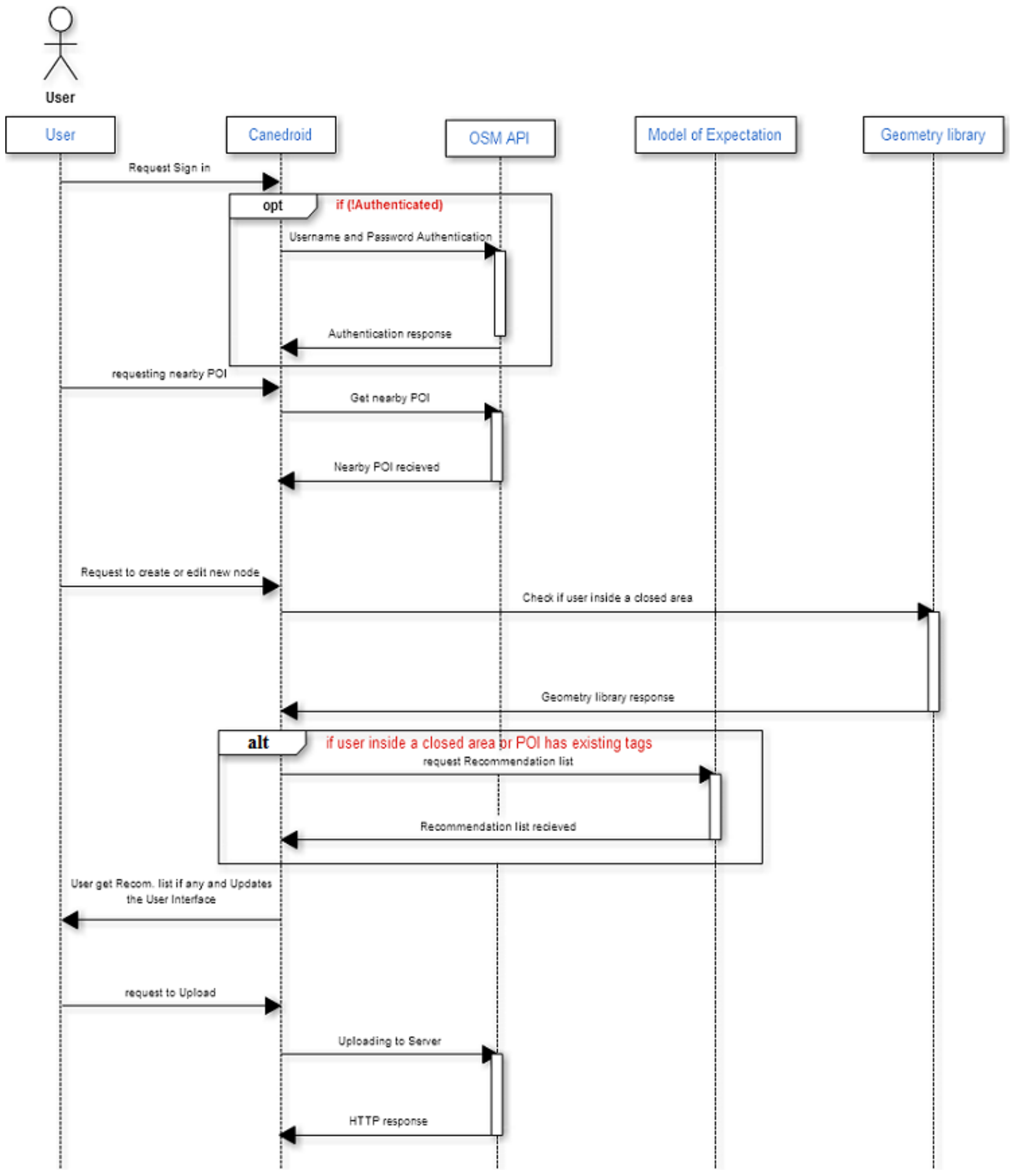

User
User
Canedroid
OSM API
Model of Expectation
Geometry library
Request Sign in
opt
if (!Authenticated)
Username and Password Authentication
Authentication response
requesting nearby POI
Get nearby POI
Nearby POI recieved
Request to create or edit new node
Check if user inside a closed area
Geometry library response
alt
if user inside a closed area or POI has existing tags
request Recommendation list
Recommendation list recieved
User get Recom. list if any and Updates the User Interface
request to Upload
Uploading to Server
HTTP response

Figure 4.1: Application Workflow

An overview of the workflow is shown in figure 4.1. First, the user is requested to sign in using his credentials. A geometry library call is sent with the user's current position to determine if he is bounded by a closed area or not. If he is, the related tags of the closed area are added to the list of recommendations along with the existing tags. After that the user can edit a nearby node from a list of nearby nodes in a specific radius, or create a new node using the help of the recommendations given. Finally, the user can click on the upload button to commit his changes and the application will save and upload to the OSM database while receiving the HTTP success response.

4.2 User Interface Design

This section introduces the user interface design in order to make life easier for blind users while navigating and using the application. Thus, Canedroid will be discussed along with the decisions taken regarding the user interface options.

4.2.1 Introduction

The application starts with the introduction layout as shown in figure 4.2, where the Text-to-speech engine greets the user, “Welcome to Canedroid”. The introduction layout lasts until the application is done with converting the model of expectation RDF file to a java model in background for recommendation system querying and matching.

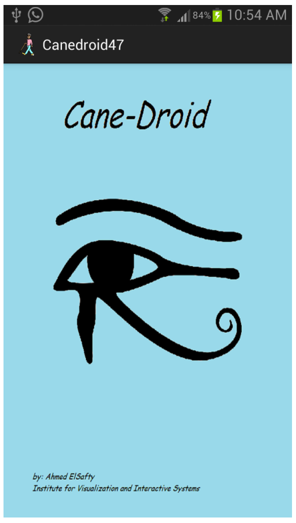


Figure 4.2: The introduction layout greets the user until the model of expectation is loaded in the background

4.2.2 Authentication layout

The text-to-speech engine informs the user that the layout is divided equally into three main parts, the username and password text areas and the sign-in button. Also, the engine informs the user to enter his credentials when he clicks anywhere inside the text areas. Finally, the user is asked to click the sign-in button at the end of the screen as shown in figure 4.3.

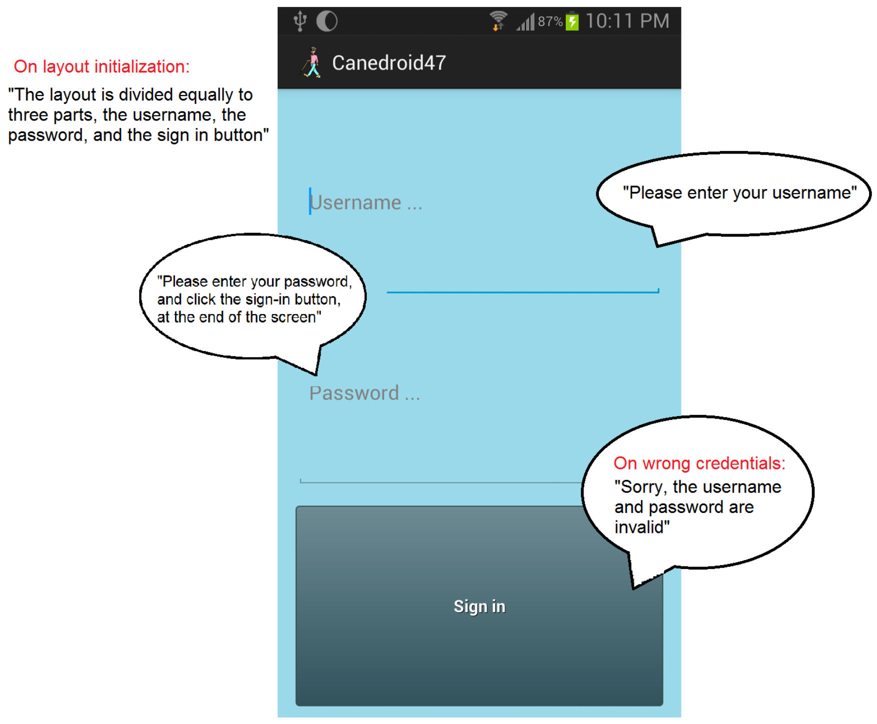

On layout initialization:
"The layout is divided equally to three parts, the username, the password, and the sign in button"
10:11 PM
Canedroid47
Username ...
"Please enter your username"
"Please enter your password, and click the sign-in button, at the end of the screen"
Password ...
On wrong credentials:
"Sorry, the username and password are invalid"
Sign in

Figure 4.3: Authentication layer text-to-speech scenario

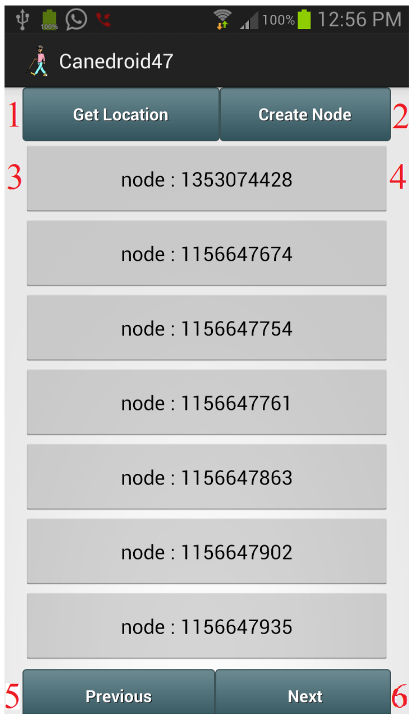
The user's credentials are checked using the OSM API with the server. If his credentials are correct, the application proceeds to the nearby nodes layout. The TTS notifies the user and asks him to enter his username and password again.

4.2.3 Nearby Nodes layout

The layout is designed to list all nearby points of interest to the user for applying changes and further actions. As shown in figure 4.4, the design contains the following:

- The user can swipe the **Margins** to get more information regarding the layout buttons or the data in the listed nodes using the Text-to-Speech engine.
- The **Get Location** button lists all nearby points of interest in a specific radius. Enabling GPS and Wifi gives better results by pinpointing the user location for more precise and accurate OpenStreetMap API calls.
- The **Body** contains all nearby points of interest in a radius around the user, they are limited to seven wide buttons per view that occupy proper portions of the layout to avoid triggering unintended events. Each button refers to a point of interest that switches the user to the Node Details layout - when clicked - for further changes.

- If the user is not satisfied by the list and not interested in amending any of the nearby points of interest around him, the **Create Node** button can be clicked to create a new node using his current longitude and latitude.
- Tests showed that using layouts with sliders was not the most suitable navigation technique for blind and visually impaired users, as it drives them to miss POIs and get lost in the scrollable view. Viewflippers were used with **Previous** and **Next** Buttons to navigate a fixed layout and give the user full insight on the list.



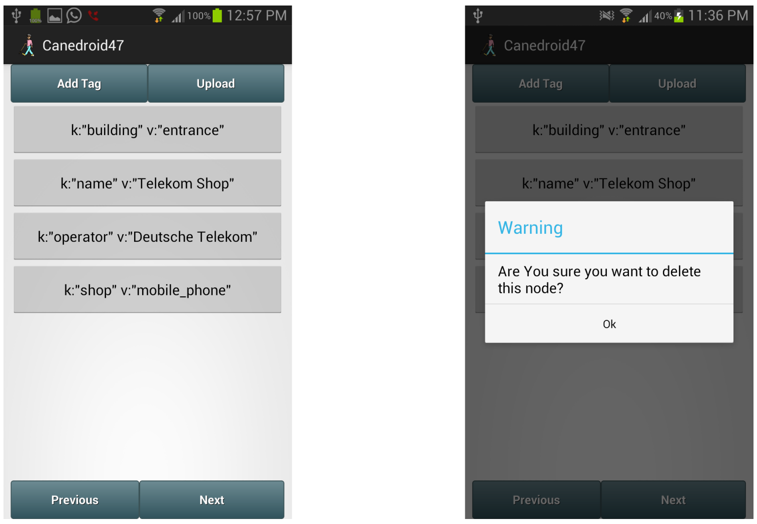
node : 1353074428
node : 1156647674
node : 1156647754
node : 1156647761
node : 1156647863
node : 1156647902
node : 1156647935

Figure 4.4: Text-to-speech engine responses to swiping different areas in the nearby nodes layout. TTS announcer when clicked on locations shown above: 1) Click to get nearby points of interest, 2) Click to create new point of interest, 3&4) Building, entrance, telekom shop, operator, Deutsche Telekom, shop, mobile shop

4.2.4 Node layout

After getting all nearby points of interest, the user has to follow two routes, creating a new node or amend an existing one. If the user wants to enhance an existing node, he gets switched to the node layout containing current tags in a list. An empty list is shown if he wants to create a new node.

In both cases, a layout similar to figure 4.5 will be displayed, where the layout shares almost all UI features of the nearby nodes layout. The margins are used to read the tags information using the Text-to-Speech Engine. Moreover, when the user long clicks on the tag button, an alert dialogue will prompt to double check if the user wants to delete this tag along with the TTS voice "Are you sure you want to delete this node?".



k:"building" v:"entrance"
k:"name" v:"Telekom Shop"
k:"operator" v:"Deutsche Telekom"
k:"shop" v:"mobile_phone"

Are You sure you want to delete this node?

Figure 4.5: The node representation in the editing layout, the tags are listed in adequately sized buttons, a deletion prompt will pop up to the user on long click

When the user adds a new tag, the recommendation layout is rendered with available suggestions based on the model of expectation. The user can identify these tags with swiping margins and going through the list using the previous and next buttons. However, if the user is unsatisfied with the list of recommendations, he can click on the 'add manually' button, which allows him to enter his own tag and value as shown in figure 4.6.

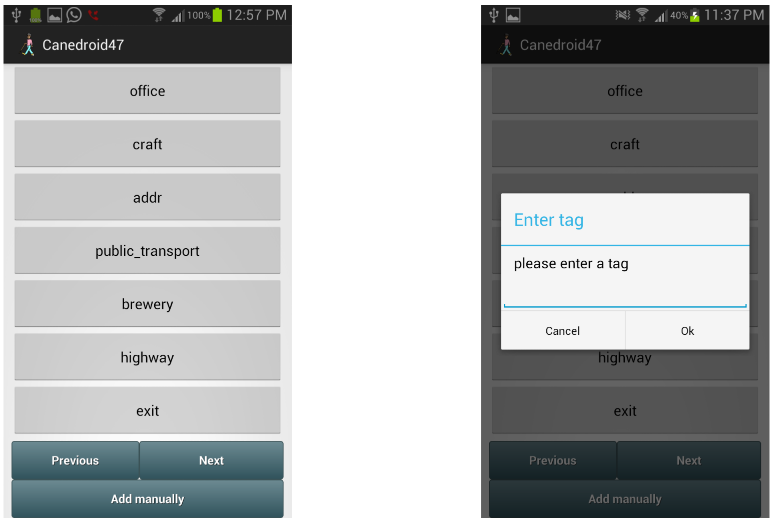


Figure 4.6: The recommendation list contains all suggested items, the user can add his own tags if he is unsatisfied with the list

At any point, the user can click the back button in the device to get back to the previous layout. Finally, when the user is done creating a new node or satisfied with the changes he did to a specific point of interest, the upload button in figure 4.5 can be clicked to send his contributions to the OpenStreetMap database using his credentials, and get back to the nearby POIs layout.

Chapter 5

Implementation

This chapter explains several technical details regarding the implementation of Canedroid. It gives an overview of the authentication process, the implementation of the points of interest layout , the recommendation system structure, the boundaries detector and the uploading phase.

5.1 Authentication

Authentication is the first phase in Canedroid after the loading screen. The user is asked to enter his username and password and click the sign in button at the end of the screen. There are two main techniques that can be used to authenticate credentials, Oauth and Basic Authentication.

Oauth

OAuth is an open protocol that provides a standardized, secure API-Authentication for desktop, web and mobile applications. It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials using a server-client protocol.

Basic Authentication

HTTP Basic authentication (BA) implementation is a simple technique for enforcing access controls to web resources. It does not need cookies, session tokens or sign-in pages. HTTP Basic authentication uses static, standard HTTP headers which means that no handshakes have to be done in the process.

Basic authentication was used in Canedroid for its simplicity , plainness and having the basic functions needed for straightforward authentication concerning certified editing in OSM. First, the username and password are concatenated, then the resulting String is attached to HTTP requests as a header in Base64[1] encoded format.

OpenStreetMap provides a set of API calls using HTTP REST methods. For instance, a GET request can be used to return the permissions granted to the current API connection. If the API client is not authorized, an empty list of permissions will be returned.

```
URL url = new
       URL("http://api.openstreetmap.org/api/0.6/permissions");
URLConnection urlconn = url.openConnection();
urlconn.setRequestProperty("Authorization", "Basic "+
authStringEnc[0]);

InputStream is = urlconn.getInputStream();
```

As shown in the code snippet above, a URL connection is opened with the encoded credentials to authenticate the signed-in user permissions. If the user is authorized, the resulting tags from the connection should contain the permissions needed to continue editing, e.g, <permission name="allow_write_gpx" />.

However, a user must request a permission to access the network, which is why the following permission should be added to the Android manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Moreover, the *Asynctask* class is used with HTTP requests as it enables proper and easy use of the UI thread. It allows to perform background operations and publish results on the UI layout without having to manipulate threads or handlers. On authentication, the user is allowed to do various changes with full control on the OSM database.

5.2 Nearby points of interest

In the second phase in Canedroid, after the user passes the authentication test, a list of nearby points if interest will appear to the user, that allow him to pick a node and start editing it. The relevant parts of this phase will be explained in details: The location manager to get the user's longitude and latitude, the bounding box to get nearby points of interest, and XML parsing to extract the data to represent the nodes as buttons in the layout.

[1] more info https://tools.ietf.org/html/rfc4648

Location Manager

The location manager is used to determine the user's geo-location that is used to get nearby points of interest. First, an instance of the *LocationManager* class is instantiated as shown in the code snippet below. If GPS or WIFI are enabled, the *LocationManager* class calls requestLocationUpdates(). Finally, *getLongitude()* and *getLatitude()* are called if the last known location is valid.

Listing 5.1: Location Manager

```
Location Manager locationManager =
        myContext.getSystemService(LOCATION_SERVICE);
locationManager.requestLocationUpdates(
        LocationManager.NETWORK_PROVIDER,
        MIN_TIME_BW_UPDATES,
        MIN_DISTANCE_CHANGE_FOR_UPDATES, this);

if (locationManager != null) {
        location = locationManager.getLastKnownLocation(
                LocationManager.NETWORK_PROVIDER);}

if (location != null) {
        latitude = location.getLatitude();
        longitude = location.getLongitude();}
```

Bounding Box

After successfully getting the user's geo-location, the longitude and latitude are added to an input String of a GET method to the API server.

```
String input =
        "http://api.openstreetmap.org/api/0.6/map?bbox="
                + left + "," + bottom + "," + right + "," + top;
```

The variables left, button, right and top are computed from the current longitude and latitude of the user by adding and subtracting 9 meters radius, which proved to display adequate amount of POIs around the user. The GET request above returns all nodes that are inside a given bounding box, any relations that reference them, all ways that reference at least one node that is inside the given bounding box, relations that reference these ways, and any nodes outside the bounding box that these ways may reference.

XML parsing

The GET methods' output mentioned earlier is in XML format. As mentioned in section 2.3, XML DOM parsers are used to parse the OpenStreetMap data coming from the server. The XML is parsed and processed to get all node and way IDs, and set a button in the nearby nodes layout for each of them. An *OnclickListener()* is attached to each button to switch the user to the Node details for further amends.

When the user clicks on a specific node to edit it, a GET request with the node's ID is sent to the OSM API. The response containing all tags and information regarding this node is parsed and displayed to the user as a set of buttons waiting for further changes as shown in section 4.2.4.

5.3 Recommendation systems

One of the main tasks of the thesis was providing the users with a list of suggestions when they want to add or edit new tag. First, the model is sketched to match the design in Chapter 4. The Model is transformed into RDF format and SPARQL query language was used to query the RDF model for recommendations.

Model of Expectation

The model of expectation was designed to be as related and sensible as possible as shown in figure 5.1. Every tag has zero or more related tags, e.g, Building is related to construction, amenity or entrance. Moreover, the model is designed to include irrelevant and invalid tags, e.g, a building can not have a highway feature or a railway stations can not exist in a football court.

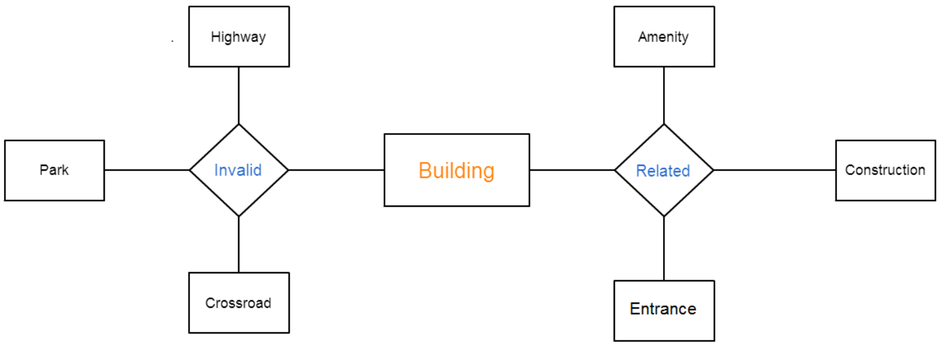


Figure 5.1: Model of Expectation design

The model is transformed into a triple XML format file, also known as the RDF file, as illustrated in the example.

Listing 5.2: RDF file format

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<rdf:Description rdf:about="term/k:building">
       <skos:related rdf:resource="term/k:amenity"/>
       <skos:related rdf:resource="term/k:construction"/>
       <skos:related rdf:resource="term/k:entrance"/>
       <skos:invalid rdf:resource="term/k:highway"/>
       <skos:invalid rdf:resource="term/k:park"/>
       <skos:invalid rdf:resource="term/k:crossroad"/>
</rdf:Description>
```

Finally, to allow the SPARQL query language to query and represent the results, the RDF file is processed by Apache Jena library to convert it into a java model for further actions as clarified in the snippet below.

```
Model m = FileManager.get().loadModel("rdf_file.rdf");
```

SPARQL

As mentioned in section 2.6, the SPARQL query language was used because of its distinct nature, as it allows second, third and Nth-levels of pattern matchmaking, making it ideal for relations discovery across RDF models, and a reasonable solution for recommendations and preferences.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select *
where { <term/k:"building"> skos:related ?o. }
```

The SPARQL syntax is fairly easy to comprehend, as shown in the query above. The query tries to match all related items to key:building. Eventually, the list of recommendation will be filled with all related items to all the Tags inside a point of interest. However, invalid items has to be removed from the list by querying the relation "skos:invalid". The list will contain with all the recommended tags that need to be displayed for the user.

Prioritized Recommendation list

The final recommendation list is not yet sorted and still contains a few redundant tags. Therefore, the list is prioritized according to the number of occurrences, i.e, common related tags to existing ones are more likely to be on top of the list. Thus, the more the existing tags are somehow related, the more the recommendation list get more specific and precise. Tree maps are used in implementing the prioritization as stated in the following code snippet.

Listing 5.3: Priority recommendation list algorithm

```
//create hashmap for tags along with their keys
Map<String, Integer> map = new HashMap<String, Integer>();

//increment tag key whenever a redundant tag is found
for (String s11 : Results) {
       if (map.containsKey(s11))
              map.put(s11, map.get(s11) + 1);
       else
              map.put(s11, 1);
}


//Comparable was implemented to compare keys
ValueComparable<String, Integer> comparable =
       new ValueComparable<String, Integer>(map);

// tree map instantiated with tags, keys and the comparable
Map<String, Integer> sortedMap =
       new TreeMap<String, Integer>(comparable);

//input the hashmap to get sorted
sortedMap.putAll(map);
```

5.4 Boundaries detector

The boundaries detector helps the blind and visually impaired users with more suggestions when they create new node and are located inside a boundary. A boundary can be a building, a park, a football field and more. Thus, the tags of the boundary can be used to generate a recommendation list for more assistance.

The first step toward implementing the boundaries detector was expanding the Bounding radius by 300 meters and get all ways around the user. Each way is checked if it forms a closed polygon by checking the nodes that form it. If the first and last node have identical IDs, then it forms a closed shape, otherwise it gets discarded.

Listing 5.4: Railway platform example showing the identical start and the final nodes.

```
<way id="50060849" uid="35560" changeset="11070866">
 <nd ref="635771597"/>     #start_node
 <nd ref="1686688594"/>
 <nd ref="1686688567"/>
 <nd ref="1686688596"/>
 <nd ref="635771597"/>     #final_node
```

```
  <tag k="area" v="yes"/>
  <tag k="bench" v="yes"/>
  <tag k="railway" v="platform"/>
  <tag k="shelter" v="yes"/>
  <tag k="waste_basket" v="yes"/>
 </way>
```

In order check whether the user's location is inside any of the closed ways, the GeoLib geometry library was used. It is a free to use, efficient, computational geometry library that contains a collection of classes like vectors, points, lines, circles, rectangles, polygons and more.

Each node in the closed polygon way has its ID concatenated in a GET message to get more information regarding the node's *longitude* and *latitude*. The points are then used to create a virtual polygon using the GeoLib. As illustrated in the code snippet below for each closed polygon, it is checked if the user location is inside the polygon. If the user proved to be inside one of these areas, the corresponding tags will be used to generate the recommendation list as mentioned in section 5.3. Otherwise the user will have to enter his own tags manually.

Listing 5.5: Boundaries detector algorithm

```
//instantiate an arraylist to hold the polygons coordinates
ArrayList<Coordinate> arrayofpoints =
       new ArrayList<Coordinate>();

//points associated with ways added to the arraylist
Coordinate wx = new Coordinate(longitude,latitude);

//instantiate the polygon with coordinates
Polygon p = new Polygon(new LinearRing(arrayofpoints));

// check intersection
boolean intersects = p.intersects(
       new Point(new Coordinate(userlong,userlat);
```

Limitations

The example below shows a case where intersection between user's current location and the boundary containing him will not be the optimal solutions, where the user is inside two coincided boundaries with different tags that might cancel out each other when the recommendation list is processed.

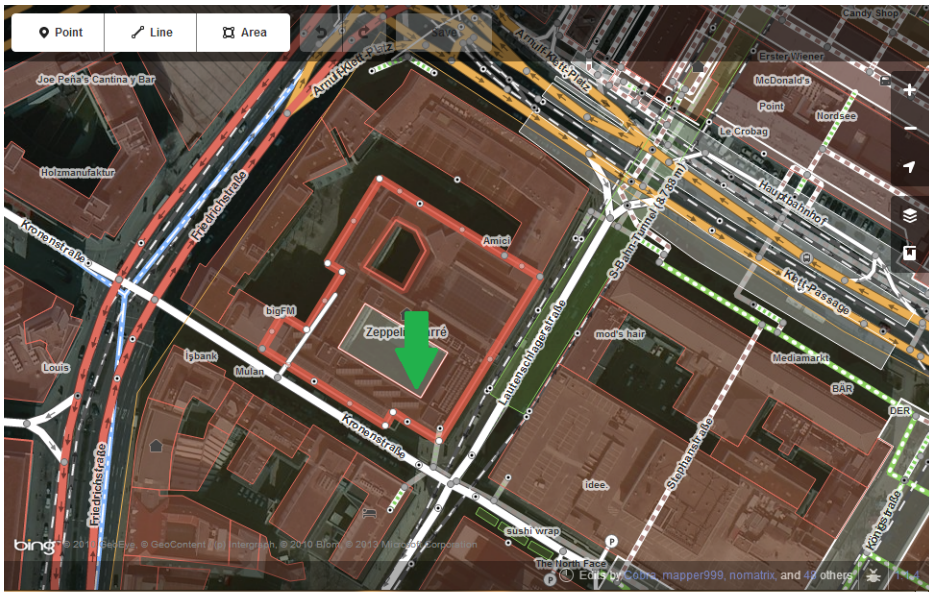


Figure 5.2: Concentric boundaries

One solution proposed was to get the tags of the smaller area of the two polygons. However that wouldn't work if the boundaries detector got three or more boundaries nearby where the user's position coincide with them all. Also, the above solution was based on the assumption that a polygon is containing the other. which will not work in the following case.

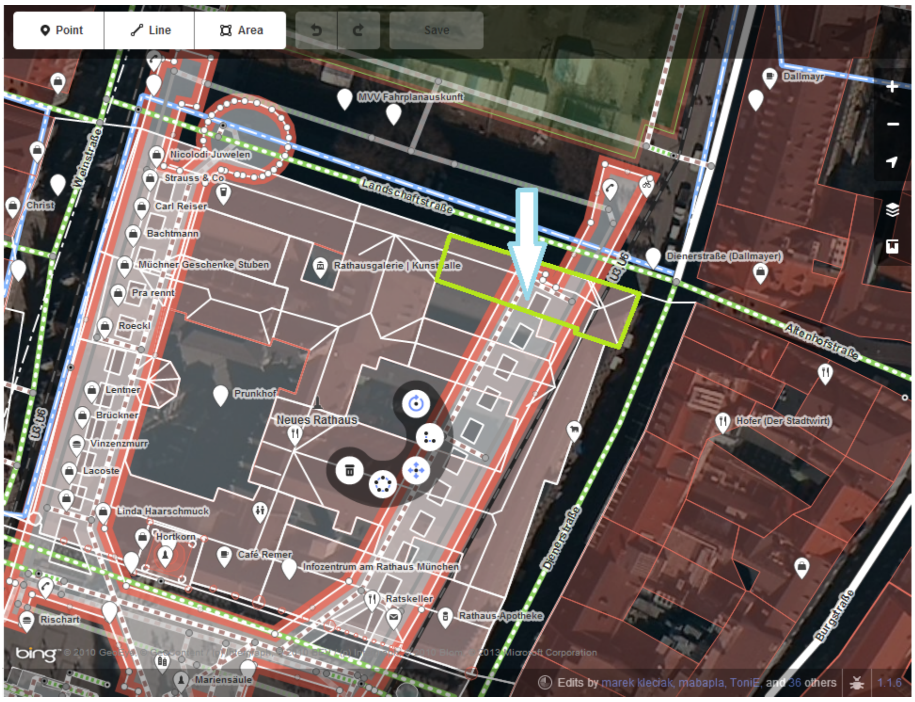


Figure 5.3: Intersecting boundaries

If the smallest area technique is applied, important tags from the larger boundary area will be discarded. Thus, all tags from all boundaries the user might be inside are taken in consideration, processed and used to generate the recommendation system. Future work may include more restrictions and test scenarios to get more specific tags regarding the user's location.

5.5 Uploading

When the user is done with creating and customizing a new node, or satisfied with his current changes, he can click on the upload button to send his changes to the server using his credentials. First, a new changeset has to be created using the following code snippet:

```
//instantiate the Output Stream Writer where
//httpCon variable is an open PUT HTTP connection
OutputStreamWriter out =
       new OutputStreamWriter(httpCon.getOutputStream());

//send the XML string and wait for response
out.write("<osm> <changeset> <tag k="created_by"
       v="Caescus Vagus 1.0"/><tag k="comment" v="testing."/>
       </changeset></osm>);
```

The above code snippet returns the newly created changeset's ID, Changesets are used by the OpenstreetMap API for backups and possibility to revert back to a recent revised version. Thus, every change to the database has to be stamped with an ID of an open changeset. Due to the fact that changesets are only open for 24 hours and some limited data restrictions, a changeset is created and opened every time the user wants to edit or create a node.

Every existing node has a version tag, which shows how many time this node has been edited. When the user wants to edit a node, the version tag is parsed, implemented and attached to the new XML output tag.

The example below shows the output string to the API, where the newly created changeset is attached, along with the ID of the node being edited, the incremented version and the tags the user wants to commit. The new version number is returned by the API on successful updating. Finally, the open changeset ID can be used to call the API and close the changeset to avoid blacklisting the user for redundantly opening new changesets in short amount of time.

```
//httpCon variable is an open PUT HTTP connection
OutputStreamWriter out =
       new OutputStreamWriter(httpCon.getOutputStream());

//send the XML string and wait for response
out.write("<osm> <node id="12323" changeset="12" lon=".."
lat=".." version="2"> <tag k="building"
       v="yes"/><tag k="name" v=" Stuttgart hauptbahnhof"/>
       </node></osm>);
```

Chapter 6

Conclusion

The purpose of this chapter is to summarize the thesis research, summing up all goals and tasks, as well as offering suggestions for future work and how to improve and develop the proposed application.

6.1 Summary

The main goal of this thesis was to implement an assistive OpenStreetMap editing application for blind and visual impaired users on mobile devices, in order to introduce blind contributors to the OSM community. The application includes a reasonable recommendation system that helps predicting amends and changes users might wish to perform.

Throughout the course of the bachelor thesis, three major tasks were realized and completed. The first task was to create a fully editable interface for OSM using its API. The application allows the user to create new Points of Interest at his location. Also, it is capable of getting all nearby Points of Interest for editing and deletion. The second task was to create an intuitive and logical model of expectation to help building a simple recommendation system for the application. The model offers the user a dynamic list of recommendations the user might like to add or amend using SPARQL query language. The third and final task was to implement a blind-friendly user interface for the application. The UI consists of introducing a text-to-speech engine, manipulating vibrations in mobile devices and exploiting the layout for the proper distribution of buttons to avoid misclicking and triggering unintended events.

6.2 Future Work

To further optimize the implemented application and overcome the problems faced, future work may include:

- Voice commands that can be used and compared to a built-in voice library of tags.
- Before uploading changes, the altered data can be matched with surrounding POIs around the user in a specific radius, alerting him if the same set of tags are found to reduce redundancies.
- Nearby Points of interest, ways and relations in a radius around the user can be used as an input of the recommendation system when creating new nodes.
- The use of crowdsourcing to enhancing the model of expectation based on users' activities and modifications .
- Adding special prefix to the ID of the users, to alert other blind and visually impaired contributors with special tags changes that might be of interest, e.g, tactile paving. Thus, using the application to help blind users and give them something back in return for their contribution.
- Using full SPARQL capabilities, i.e, more prioritization of recommendation using second, third and Nth level of matchmaking using the model of expectation.
- The thesis lacks the evaluation phase on real users which classifies it as a non experimental research. Thus, feedbacks from blind and visually impaired users can be helpful to upgrade the user interface and add more features.

To sum up, Canedroid is the first OpenstreetMap editing application that offers directive assistance to blind users. A recommendation system was proposed and implemented to act as an assistive tool, where the user is offered a list of suggested changes he might want to commit. Also, Canedroid is designed to match blind and visually impaired users' needs by exploiting text-to-speech engines, vibrations and alert dialogues to allow them to commit their intended changes with as little confusion as possible.

Appendix

List of Figures

Listings

Bibliography

[1] Sören Auer, Jens Lehmann, and Sebastian Hellmann. Linkedgeodata: Adding a spatial dimension to the web of data. In *The Semantic Web-ISWC 2009*, pages 731–746. Springer, 2009.

[2] Andrea Ballatore, Michela Bertolotto, and David C Wilson. Geographic knowledge extraction and semantic similarity in openstreetmap. *Knowledge and Information Systems*, pages 1–21, 2012.

[3] Christian Becker and Christian Bizer. Exploring the geospatial semantic web with dbpedia mobile. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4), 2009.

[4] Tim Bray. Namespaces in xml 1.0 (third edition), dec 2009.

[5] Irene Celino and Francesco Corcoglioniti. Towards the formalization of interaction semantics. In *Proceedings of the 6th International Conference on Semantic Systems*, page 10. ACM, 2010.

[6] Andreas Schultz Chris Bizer. Berlin sparql benchmark results, mar 2009.

[7] Steve Coast. Yahoo! aerial imagery in osm, Dec 2006.

[8] D Coomans and DL Massart. Alternative¡ i¿ k¡/i¿-nearest neighbour rules in supervised pattern recognition: Part 1. k-nearest neighbour classification by using alternative voting rules. *Analytica Chimica Acta*, 136:15–27, 1982.

[9] Marin Dimitrov. Xml standards for ontology exchange. In *Proceedings of OntoLex*, pages 8–10. Citeseer, 2000.

[10] ANTONIO Estepa and FRANCISCO T Sanchez-Cobo. Evaluation of the comprehension of correlative and regression through problems. *Statistics Education Research Journal*, 2(1):54–68, 2003.

[11] Tom Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[12] Elliote Rusty Harold. *Processing Xml with Java.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[13] Ivan Herman. W3c semantic web frequently asked questions @ONLINE, November 2009.

[14] Jiewen Huang, Daniel J Abadi, and Kun Ren. Scalable sparql querying of large rdf graphs. *Proceedings of the VLDB Endowment*, 4(11):1123–1134, 2011.

[15] David Hunter, Jon Rafter, Andrew Watt, and Linda McKinnon. *beginning XML*. John Wiley & Sons, 2011.

[16] Jkpj. Sharenav wiki page, nov 2011.

[17] Prasad Kulkarni. Distributed sparql query engine using mapreduce. *Master of Science, Computer Science, School of Informatics, University of Edinburgh*, 2010.

[18] Ioan Alfred Letia and Anca Nicoleta Marginean. Service monitoring with ontology based expectations. In *Intelligent Computer Communication and Processing (ICCP), 2011 IEEE International Conference on*, pages 111–114. IEEE, 2011.

[19] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, pages 73–105. Springer, 2011.

[20] M.Rajasekhara Babu M V Uttam Tej, Dhanaraj Cheelu and P Venkata Krishna. Analyzing xml parsers performance for android platform. *VIT University, Tamil Nadu*, 2011.

[21] Matt Lacchiato Marcus Wolschon. Vespucci wiki page, apr 2011.

[22] Boris Motik. Reasoning in kaon2, nov 2006.

[23] Olaf Noppens, Marko Luther, Thorsten Liebig, Matthias Wagner, and Massimo Paolucci. Ontology-supported preference handling for mobile music selection. In *Proceedings of the Multidisciplinary Workshop on Advances in Preference Handling, Riva del Garda, Italy (August 2006)*. Citeseer, 2006.

[24] OSM. Openstreetmap stats, jul 2013.

[25] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *The Semantic Web-ISWC 2006*, pages 30–43. Springer, 2006.

[26] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer, 2011.

[27] Andy Seaborne. Sparql query language for rdf, oct 2006.

[28] Andy Seaborne. Jena documentation, 2011.

[29] Akbar S Shaik, Gahangir Hossain, and Mohammed Yeasin. Design, development and performance evaluation of reconfigured mobile android phone for people who are blind or visually impaired. In *Proceedings of the 28th ACM International Conference on Design of Communication*, pages 159–166. ACM, 2010.

[30] Mark Needleman Norman Walsh Tony Coates, Michael Mealling. Uris, urls, and urns: Clarifications and recommendations 1.0, jul 2009.

[31] Fangju Wang, Jing Li, and Hooman Homayounfar. A space efficient xml dom parser. *Data & Knowledge Engineering*, 60(1):185–207, 2007.

[32] WHO. Visual impairment and blindness, jun 2012.