

# **Titel der Arbeit**

## **Optionaler Untertitel der Arbeit**

**BACHELORARBEIT**

zur Erlangung des akademischen Grades

**Bachelor of Science**

im Rahmen des Studiums

**Software und Information Engineering**

eingereicht von

**Kevin Haller**

Matrikelnummer 1325694

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Mag.rer.soc.oec Stefan Biffl, Univ.Doz.

Mitwirkung: MSc PhD Reka Marta Sabou, Project Ass.

Wien, 1. August 2016

Kevin Haller

Stefan Biffl



# Title

## Optional Subtitle of the Thesis

### BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

### Bachelor of Science

in

### Software and Information Engineering

by

**Kevin Haller**

Registration Number 1325694

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Mag.rer.soc.oec Stefan Biffl, Univ.Doz.

Assistance: MSc PhD Reka Marta Sabou, Project Ass.

Vienna, 1<sup>st</sup> August, 2016

\_\_\_\_\_  
Kevin Haller

\_\_\_\_\_  
Stefan Biffl



# **Erklärung zur Verfassung der Arbeit**

Kevin Haller  
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. August 2016

---

Kevin Haller



# Danksagung

**TODO:** Ihr Text hier.



# Acknowledgements

**TODO:** Enter your text here.



# Kurzfassung

**TODO:** Ihr Text hier.



# Abstract

**TODO:** Enter your text here.



# Contents

<b>Kurzfassung</b>	<b>xi</b>
<b>Abstract</b>	<b>xiii</b>
<b>Contents</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem description . . . . .	3
1.3 Structure of the thesis . . . . .	4
<b>2 Background</b>	<b>5</b>
2.1 Web of Data . . . . .	5
2.2 RDF . . . . .	5
2.3 Ontology . . . . .	5
2.4 SPARQL . . . . .	5
<b>3 Related Work</b>	<b>7</b>
3.1 Geospatial ontologies . . . . .	7
3.2 Linked Open Data based map applications . . . . .	13
3.3 Indoor modelling . . . . .	14
<b>4 Solution</b>	<b>17</b>
4.1 Data acquisition . . . . .	17
4.2 Prototype of ontology . . . . .	21
4.3 Architectural prototype . . . . .	24
4.4 Map application . . . . .	43
<b>5 Discussion</b>	<b>45</b>
5.1 Evaluation . . . . .	45
5.2 Further work . . . . .	47
5.3 Conclusion . . . . .	48
<b>List of Figures</b>	<b>51</b>

<b>List of Tables</b>	<b>52</b>
<b>List of Algorithms</b>	<b>53</b>
<b>Acronyms</b>	<b>55</b>
<b>Bibliography</b>	<b>57</b>

# Introduction

## 1.1 Motivation

Today's universities usually have to manage a significant amount of information and have to provide systems to handle common services like finding and booking courses for students. As the current time demands it, those services are often provided over web sites participating in the Web of Documents (see figure 1.1). The domain of a university is quite complex and as a consequence it is likely that different isolated information systems handling a particular part of the domain evolve; resulting in an environment, where information is distributed over multiple disconnected data silos that may have different formats and/or data owners. Such a situation prevents universities of fully exploiting their data [38].

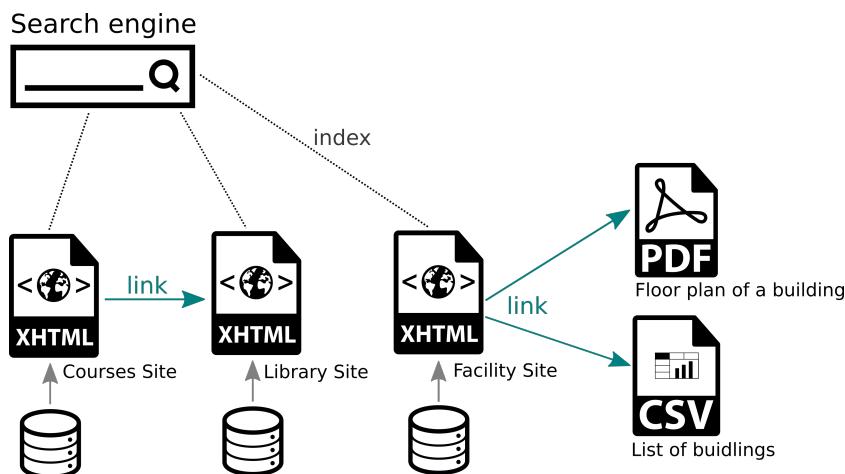


Figure 1.1: Illustration of the Web of Documents

## 1. INTRODUCTION

---

The same applies at the moment to the Vienna University of Technology (TU Vienna). In order to find a special location like a certain lecture hall and how it can be accessed without obstacles like stairways for persons with mobility-impairments, one has to search for information on different web sites, scan floor plans and eventually construct a convenient route to the location based on the gathered knowledge. For humans this procedure does not constitute a problem, but think of machines. The process of information retrieval through data mining or harvesting is quite difficult and/or time consuming. As a consequence application developers that may have innovative ideas, which would be a benefit for the information environment of the university, face a barrier that is hard to overcome. Linked Data (LD) is one way to transform this information published on multiple web sites into a university-wide data space (see figure 1.2).

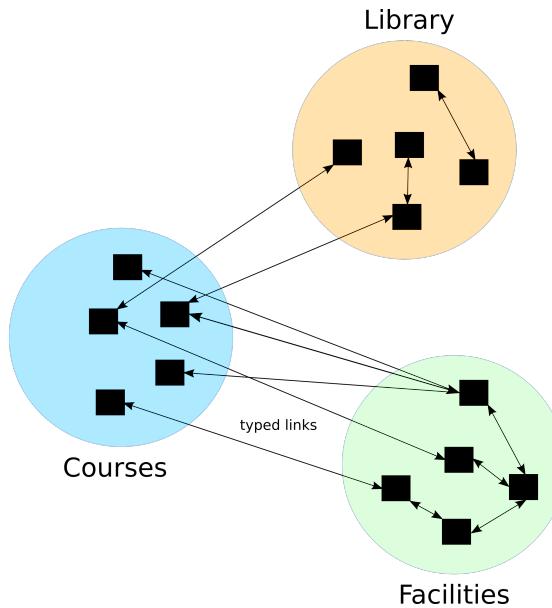


Figure 1.2: Illustration of the Web of Data

LD describes a method of publishing structured data so that it can be interlinked and become more useful. It builds upon standard Web technologies, but rather than using them to serve web pages for human readers, it extends them to share information in a way that can be read automatically by computers. This enables data from different sources to be connected and queried [7].

This thesis aims to show the potential that could evolve, if spatial data about TU Vienna is transformed into a machine-readable form such as LD, by proposing a prototype of a map application based on a subset of spatial data about TU Vienna. This subset is

intended to comply with the principles of LD [4] and to fulfil the submission criteria to the Linking Open Data cloud<sup>1</sup>, which are as follows [11]:

- There must be resolvable `http://` (or `https://`) URIs.
- They must resolve, with or without content negotiation, to RDF data in one of the popular RDF formats (RDFa, RDF/XML, Turtle, N-Triples).
- The dataset must contain at least 1000 triples.
- The dataset must be connected via RDF links to a dataset that is already in the diagram. This means, either your dataset must use URIs from the other dataset, or vice versa. We arbitrarily require at least 50 links.
- Access of the entire dataset must be possible via RDF crawling, via an RDF dump, or via a SPARQL endpoint.

## 1.2 Problem description

As already mentioned does the TU Vienna distribute spatial information over different sources. Few of this spatial data is actually in a machine-readable format, so that it can be transformed into LD automatically. The **1st problem** to solve is therefore the transformation process of the unstructured part of the data into a machine-readable format. Section 4.1 is suggesting a solution for this problem.

The **2nd problem** is how the structured spatial data shall be described so that it can be easily used by application developers as well as integrated into the Web of Data. The developed ontology shall be compliant to the best practises for publishing ontologies [5]. Section 4.2 is suggesting a prototype of an ontology by taking already existing approaches (see section 3.1 and 3.3) into consideration.

After the spatial data was transformed into LD, a system must be designed to expose this data on the Web in order to qualify as Linked Open Data (LOD), representing the **3rd problem** to solve. The solution shall take best practises for publishing LD[18] and best practises for spatial data on the Web[30] into consideration. Section 4.3 is proposing an architectural prototype of such a system.

The **4th problem** is the development of a map application that is based on the resulting LOD. The application shall answer the following question: *"Give me all learning rooms that are nearby that are free in a given time range and are accessible without obstacles for person with mobility-impairments"* to show the potential of the data. Section 4.4 presents a solution to this problem.

---

<sup>1</sup><http://lod-cloud.net/>

### 1.3 Structure of the thesis

This chapter outlined the motivation for writing this thesis and gave a description of the problems for which a solution will be suggested. The rest of this thesis is structured as follows: Chapter 2 discusses the basic concepts, principles and technologies that build the foundation of LD and the Semantic Web. It is intended to be a brief introduction for readers that are not familiar with this topic. Chapter 3 provides a summary of past efforts in research and works related to the focus of this thesis. In the subsequent chapter 4 a solution for the given problems will be suggested and discussed. This includes the prototype of an ontology that models the problem domain as well as an architectural prototype of a system that manages a subset of spatial data about the TU Vienna in form of LD and provides a human- and machine-friendly interface to it. Finally, the implemented solution is evaluated and conclusions are drawn in the last chapter 5. It provides furthermore an outlook to future work and potential improvements.

# CHAPTER 2

## Background

### 2.1 Web of Data

**TODO:** Enter your text here.

### 2.2 RDF

Resource description framework (RDF) **TODO:** Enter your text here.

### 2.3 Ontology

**TODO:** Enter your text here.

#### 2.3.1 RDFS

**TODO:** Enter your text here.

#### 2.3.2 OWL

**TODO:** Enter your text here.

### 2.4 SPARQL

SPARQL protocol and RDF query language (SPARQL) **TODO:** Enter your text here.



# CHAPTER 3

## Related Work

In this chapter important efforts and research related to this thesis are outlined. Section 3.1 describes and evaluates geospatial ontologies that were suggested by [30] or discovered in the dataset of Linked Open Vocabulary (LOV) through a systematic search. Section 3.2 lists already existing map applications that are based on LOD and describes their capabilities. The final section 3.3 describes done research and designed ontologies for modelling indoor environments.

### 3.1 Geospatial ontologies

This section is going to outline common ontologies for describing geospatial data and to evaluate them from the perspective of the given problem description (see 1.2). The ontologies were either suggested by [30] or discovered in the dataset of LOV by common spatial search terms like '*feature*', '*geometry*' or '*location*'. *Feature* and *Geometry* are widely used terms in these ontologies and describe two different concepts of geospatial science. A *feature* is simply a spatial entity that can be everything from a building with fixed position to a movable food truck as long as it has a spatial extent. *Geometry* as the name suggests is a certain geometric shape from points, lines to polygons. Geometric shapes can be used to describe the spatial extent of *features*.

The ontologies are visualized using the VOWL2 notation[21]. Classes are modelled as circles, properties in form of rectangles; data type properties have a green colour and object properties a blue one. Literals are presented in a yellow rectangle.

#### 3.1.1 Basic Geo Vocabulary (WGS84)

Basic Geo (WGS84 long/lat) Vocabulary (GEO)[8] is a lightweight ontology published by W3C to describe the position of a spatial *feature* using the WGS84 geodetic reference

### 3. RELATED WORK

---

datum<sup>1</sup> with the properties longitude, latitude and attitude. The general class SpatialThing is the domain of all these properties. Figure 3.1 shows a visualization of this ontology.

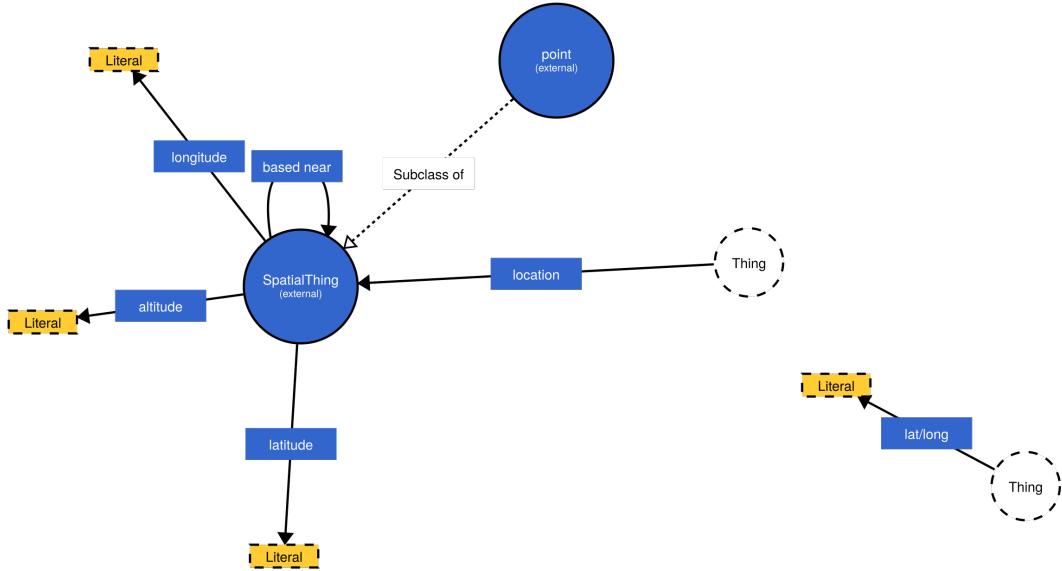


Figure 3.1: Visualization of GEO

Advantages are its simplicity and popularity as an analysis of the *Comprehensive Knowledge Archive* (CKAN, "The Data Hub"<sup>2</sup>) indicates. This analysis shows that GEO is used in **538** different datasets in the archive and is thereby the 4th most used ontology regarding usage in different datasets [26]. It is used by big players like DBpedia<sup>3</sup>, Linked-GeoData<sup>4</sup> and GeoNames<sup>5</sup>. Listing 3.1 shows a data snippet of DBpedia describing the town square 'Karlsplatz' in Vienna with GEO.

Listing 3.1: Snippet of DBpedia

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .

dbr:Karlsplatz rdf:type geo:SpatialThing ;
    rdfs:label "Karlsplatz"@en , "Karlsplatz (Wien)"@de ;
    geo:lat "48.19916534423828125"^^xsd:float ;
    geo:long "16.370000839233398438"^^xsd:float .
```

<sup>1</sup>[http://en.wikipedia.org/wiki/World\\_Geodetic\\_System](http://en.wikipedia.org/wiki/World_Geodetic_System)

<sup>2</sup><http://thedatahub.org>

<sup>3</sup><http://dbpedia.org>

<sup>4</sup><http://linkedgeodata.org/About>

<sup>5</sup><http://linkedgeodata.org/About>

GEO evolved into a standard for presenting the location of points of interest and as described by [18] in an article about best practises, standardized ontologies shall be used wherever possible to facilitate inclusion into the Web of Data.

However, this ontology is not sufficient for describing spatial data more complex than a single point in a coordinate reference system.

### 3.1.2 NeoGeo Geometry/Spatial Ontology

NeoGeo Geometry Ontology (NGEO)<sup>6</sup> is an ontology that aims to provide a comprehensive descriptive power for modelling geographic regions, whereas NeoGeo Spatial Ontology (SPATIAL)<sup>7</sup> aims to describe topological relationships between *features*. Both ontologies were designed to have a strict distinction between *features* and *geometries* [23].

#### NGEO

NGEO follows the principle of modelling geometries in pure RDF, which leads to a number of classes and properties for covering all shapes suggested by Simple Features Profile<sup>8</sup>. This includes points, lines and polygons. Shapes that go beyond a simple point are represented by a RDF collection of point instances, whereby point is a external class of GEO. Figure 3.2 shows a visualization of this ontology.

One advantage of this approach is that triple stores do not have to meet special requirements in order to enable the querying over geometric data like it is the case for GeoSPARQL. However, this shifts the burden of writing geometric queries to developers. A further problem is the high demand for resource identifiers or blank nodes (at least for each point of a shape), as the example of describing the geometry of Iceland shows<sup>9</sup>; this significantly increases the verbosity of the data without adding particular gains of expressivity to it[3]. NGEO has also a generic property asWKT for pointing to the Well-known text (WKT) serialization of a geometry, which has at the moment of writing the status 'deprecated'.

LinkedGeoData uses the *geometry* property of this ontology to point to the *geometry* of a *feature*, but the *asWKT* property of GeoSPARQL to represent the geometry.

#### SPATIAL

SPATIAL is an ontology that aims to provide properties to make topological relationships of *features* explicit, which else would only exist implicit in the geometric data. Included properties cover all topological relationships described by RCC-8 like does a *feature* overlap with another *feature*. One problem of making the relationships explicit is that these relationships have to be adopted if the *geometry* of *features* changes.

---

<sup>6</sup><http://geovocab.org/geometry>

<sup>7</sup><http://geovocab.org/spatial>

<sup>8</sup><http://www.ogcnetwork.net/gml-sf>

<sup>9</sup>[http://nuts.geovocab.org/id/IS\\_geometry.ttl](http://nuts.geovocab.org/id/IS_geometry.ttl)

### 3. RELATED WORK

---

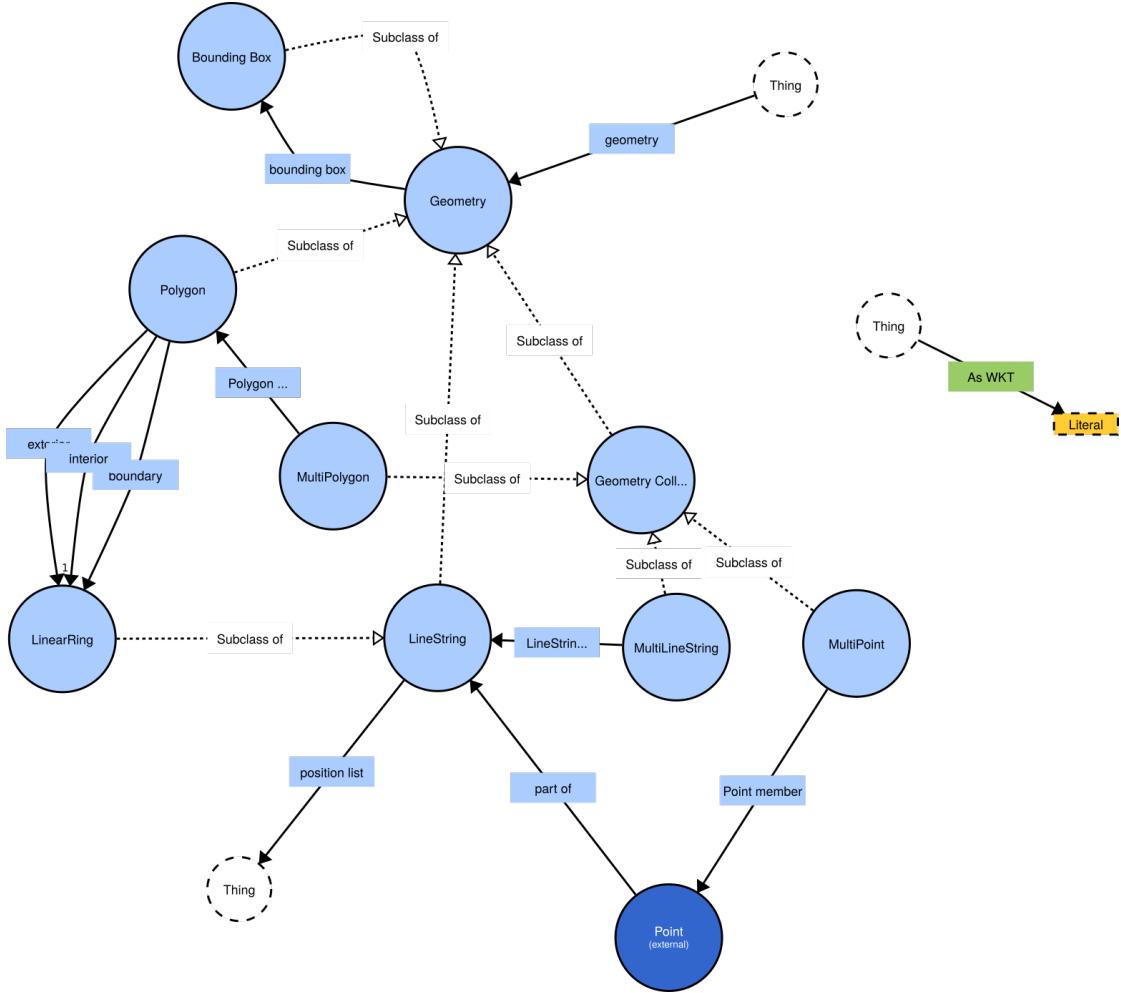


Figure 3.2: Visualization of NGEO

#### 3.1.3 GeoSPARQL

GeoSPARQL defines a set of SPARQL extension functions, a set of Rule Interchange Format (RIF) rules, and a core ontology for geographic information[25]. “GeoSPARQL attempts to unify data access for the geospatial Semantic Web”[3]. Figure 3.3 shows a visualization of this core ontology. It has two major classes namely *Feature* to represent spatial *features* and *Geometry* to represent *geometries*. An instance of *Geometry* can be assigned to an instance of *Feature* with the property *hasGeometry*. It is based on the Simple Features model<sup>10</sup> like NGEO. However, GeoSPARQL defines two properties that can be used to point to the serialization of a geometry; either *asWKT* for WKT or *asGML* for Geography Markup Language (GML); in contrast to NGEO, where *geometries* are represented in pure RDF. The fact that *geometries* are serialized using WKT or

<sup>10</sup><http://www.ogcnetwork.net/gml-sf>

GML simplifies the integration of *geometries*, because these serialization formats are widely supported (e.g. QGIS). GeoSPARQL proposes furthermore a set of extension functions that can be separated into two categories. The first category includes functions that take a number of *geometries* as arguments and produces a new one or returns a value with a certain datatype. The distance function for instance takes two *geometries* as well as a reference to measurement unit like meters and returns a xsd:double value. The second category are functions that are equivalent to the properties describing topological relationships (e.g. contains, coveredBy), only that these functions can be used in filters and are computed on-the-fly; not asserted in the knowledge base. However, the used triple store or LD framework must support those functions and the GeoSPARQL standard in general, which limits the choice. As it will be discussed in section 4.3.1, one of the common frameworks Apache Jena and its native triple store is at the moment not compliant with the GeoSPARQL standard.

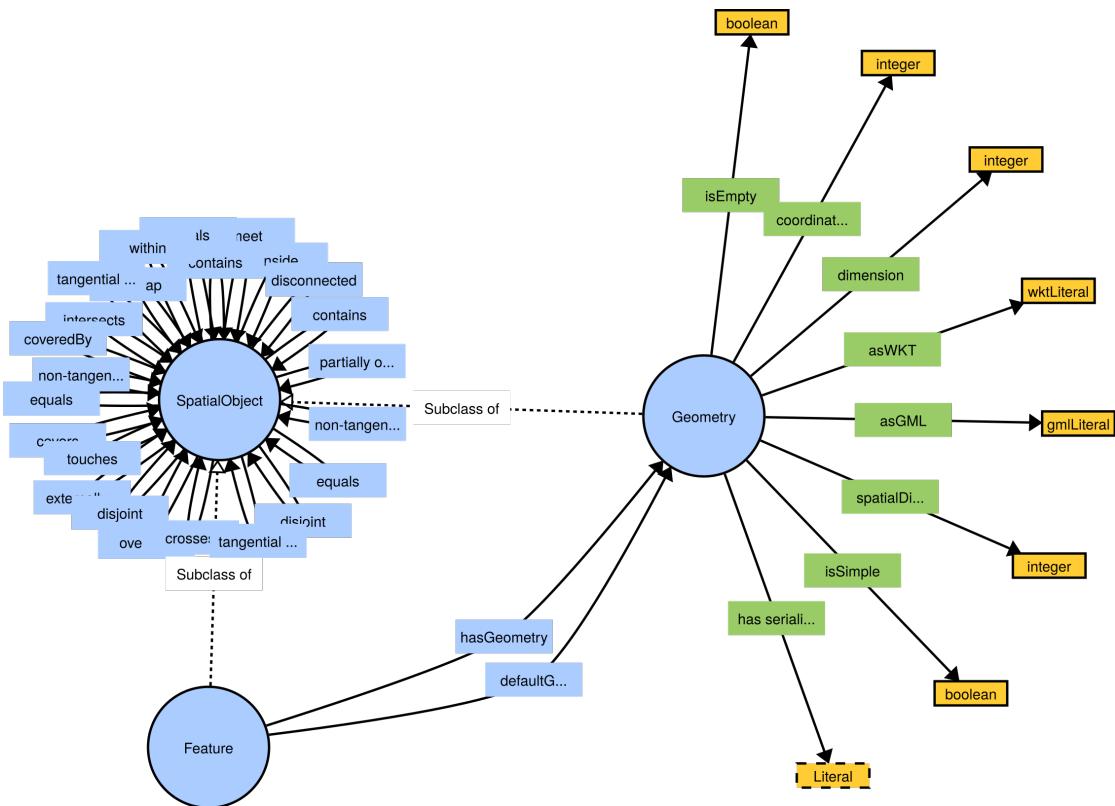


Figure 3.3: Visualization of GeoSPARQL

#### 3.1.4 schema.org

schema.org<sup>11</sup> is an initiative that was started by Google, Bing and Yahoo! (subsequently joined by Yandex) to promote a common ontology for publishing structured data mark-up on web pages [13]. This machine-readable information can then be extracted from web pages by search engines to improve search results.

schema.org is a quite broad ontology covering a set of different domains from events, organizations to places. For this thesis especially the class `Place` is of interest describing entities with a fixed, physical extent; it has a set of more specific subclasses. This class has properties to assign an address to its instances (`address`) as well as to provide geo-information (`geo`). The value of the property `geo` can either be instances of the class `GeoCoordinates` or `GeoShapes`. `GeoCoordinates` represents a point in a georeference system similar to GEO. `GeoShapes` on the other hand provide the ability to describe more complex geometric shapes including circles, boxes, polygons and simple lines. All those shapes are expressed in textual form following an own specific pattern, in contrast to GeoSparql, which declares the use of GML or WKT. However, such literals could be transformed into WKT by string operations (supported in SPARQL) except for the circle, which has no direct representation in WKT.

One advantage of this ontology is the visibility to search engines when exposed on web pages in one of the supported formats (Microdata<sup>12</sup>, RDF in Attributes (RDFa)<sup>13</sup> and JSON-LD<sup>14</sup>). As mentioned earlier, the embedded data can then be extracted and in fact major search engines crawl for it to enhance search results[14]. However, there are odd ascriptions of properties like that instances of the class `Beach` (subclass of `Place`) can have the property `faxNumber`; it must be considered that schema.org is meant for web masters to add structured metadata to their web sites [35] and not to describe a specific domain precisely.

#### 3.1.5 ISA Programme Location Core Vocabulary

ISA Programme Location Core Vocabulary (LOCN) is an ontology that provides a set of properties and classes to describe the geometry, address and location of a *feature*. Figure 3.4 shows a visualization of this ontology. It provides a comprehensive set of properties to describe the address of a *feature*, but a minimal set for geometries and locations. For geometries this ontology suggests the use of external classes or a simple literal representing the serialization of the geometry in formats such as WKT and GML. Those suggested classes are *Geometry* of GeoSPARQL with its more specific subclasses and `GeoCoordinates` as well as `GeoShapes` from schema.org. In case of representing simple points also `point` of GEO is mentioned. [24] However, this quite broad range of possibilities does not make it easy for the consumer of such LD.

---

<sup>11</sup><http://schema.org/>

<sup>12</sup><https://www.w3.org/TR/microdata/>

<sup>13</sup><https://www.w3.org/TR/rdfa-syntax/>

<sup>14</sup><http://json-ld.org/>

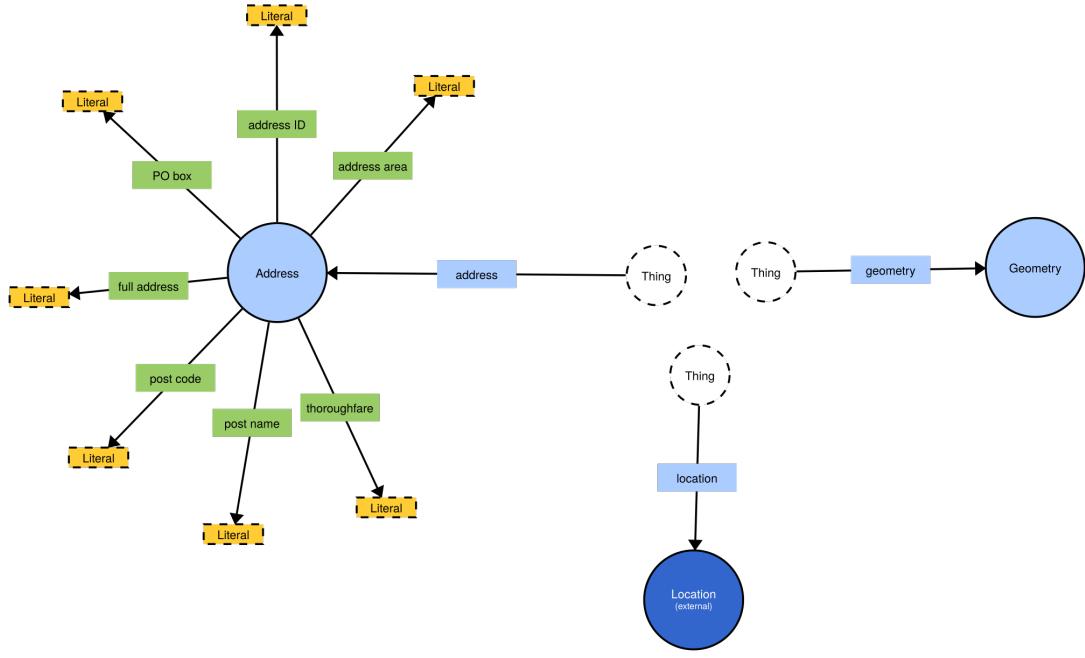


Figure 3.4: Visualization of LOCN

## 3.2 Linked Open Data based map applications

Multiple universities (e.g. Linked Universities<sup>15</sup>) exposed their public university data and map applications played out to be a common use case for this data.

University of Münster started an Open Data initiative named LODUM[19] from which a map application<sup>16</sup> evolved that shows the location of buildings and which departments are located in the selected building, but there is no deeper insight into the buildings. However, there is a publication of the initiative that deals with the issue of indoor navigation [20]. Furthermore an indoor map ontology named LIMAP<sup>17</sup> was designed.

University of Southampton is a further university that exposed spatial information about their campus as LOD. In contrast to the map of the LODUM initiative Southampton's map application<sup>18</sup> (see figure 3.5) gives insight into buildings and shows the location of certain rooms with the ability to switch the floor. It shows also computer rooms with their current estimated capacity and opening hours.

University of Oxford has also exposed spatial information in form of LOD on their Open

<sup>15</sup><http://linkeduniversities.org>

<sup>16</sup><http://app.uni-muenster.de/Karte/>

<sup>17</sup><http://lodum.de/results/>

<sup>18</sup><http://maps.southampton.ac.uk/>

### 3. RELATED WORK

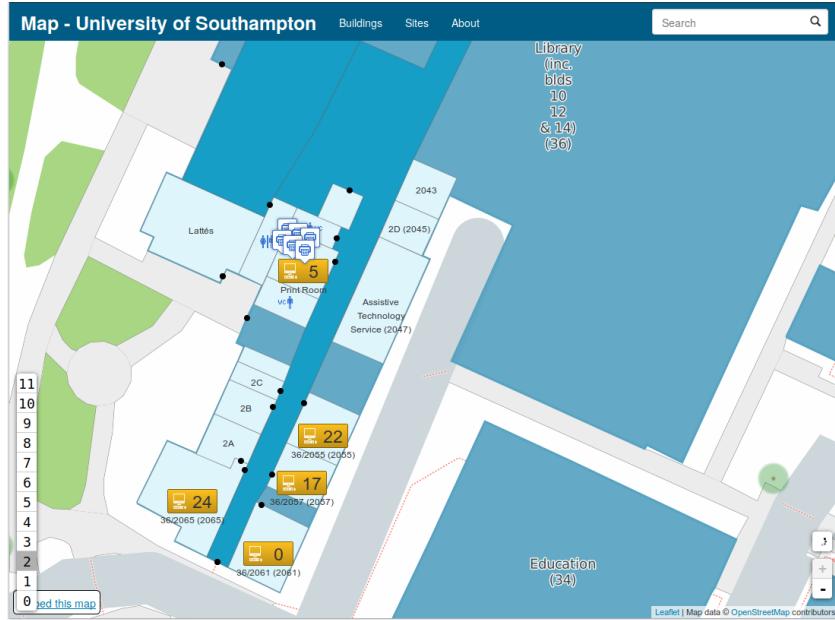


Figure 3.5: Map application of University of Southampton

Data portal and developed an application named *University Science Area Map*<sup>19</sup>. It highlights buildings, where departments of a certain field are located, on a map, but it gives no insight into the buildings.

### 3.3 Indoor modelling

Section 3.1 outlined ontologies to describe the spatial extent of *features* and topological relationships between them, but those ontologies are not intended for expressing the characteristics of *features* in an indoor environment; like for example that a *feature* is a learning room with certain opening hours or an elevator to get from one floor to another. This section discusses previous efforts and ontologies to model indoor environments and also how to navigate in them.

IndoorGML<sup>20</sup> is a Extensible Markup Language (XML) schema of OGC that aims to provide a framework for representing and exchanging indoor spatial information, but no efforts has yet been made to transform its model to the LD domain. OntoNav presents a semantic indoor navigation system that aims to follow a user-centric paradigm in which the capabilities, limitations and preferences of the user are taken into consideration for computing the best suiting path. A path is according to OntoNav a sequence of connected corridors, passages (stairways, ramps, elevators, etc.) and exits from one location to another[1]. ONALIN is a ontology that models the indoor environment similar

<sup>19</sup><https://data.ox.ac.uk/explore/science-area/>

<sup>20</sup><http://indoorgml.net/>

to OntoNav, but it follows the approach of modeling buildings as network of hallways consisting of different points (start, decision and corner points); resulting in a finer granularity. ONALIN also takes the American Disability Act into consideration, which leads to properties for describing the height and number of steps of stairways or the height of sinks in a restroom [12]. Both ontologies are at the moment not locatable over the Web. iLoc is an ontology that has a minimal set of classes to describe the internal structure of a building from rooms, floors to vertical passages like elevators and stairways. Points of interest like entrances and landmarks play a special role in the navigation. A route section connects then two points and asserts that there is a walk-able path between them. This route section can have certain constraints like an access control measurement is required or steps have to be climbed [29].

#### 3.3.1 Other ontologies

The ontologies mentioned in this section have been discovered in the repository of LOV by searching for terms such as '*room*', '*building*' and '*toilets*' or explored in datasets of map applications based on LOD (see section 3.2).

Buildings and Rooms Vocabulary (ROOMS) provides a minimal set of classes and properties to describe the basic structure of a building. It has 6 classes Building, Floor, FloorSection, which is a named (identifiable) section of a floor, Room and Desk as well as Site, which describes an area of land like a campus.



# CHAPTER 4

## Solution

This chapter suggests solutions for the given problem description (see 1.2). Section 4.1 is dealing with the problem of transforming unstructured spatial data of the TU Vienna into a machine-readable format. The question of how this structured data shall be described is answered in section 4.2, where a prototype of an ontology is discussed. The subsequent section 4.3 is proposing an architectural prototype of a system for integrating and publishing LD consistent with the LD principles [4]. The final section 4.4 is presenting a map application to show the potential of the generated spatial LOD.

### 4.1 Data acquisition

Spatial data of the TU Vienna is distributed over multiple sources with different formats and data owners. In order to be eventually transformed into LD, this data must be extracted and prepared for the transformation. Figure 4.1 visualizes the different data sources.

The major source for building information is the web site of the facility management unit of the TU Vienna (GUT<sup>1</sup>). This site lists all buildings and an enumeration of their floors in form of a (X)HTML table. It also contains links to the floor plans that are provided as Portable Document Format (PDF) files as well as to a PDF file for each building consisting of a table of all contained rooms with their intended function (e.g. office room, sanitary room). The central information system TISS<sup>2</sup> provides a RESTful API that gives access to organizational information and the public address book. For this thesis, this API is useful for linking persons and organizations to their offices. TISS has also a web page dedicated to present the event schedule of certain rooms like lecture halls, but this data is not available over the mentioned RESTful API; only in a human-readable form.

---

<sup>1</sup><http://www.gut.tuwien.ac.at/>

<sup>2</sup><https://tiss.tuwien.ac.at/>

Section 4.1.1 deals with the problem of extracting and transforming tables from PDFs and web pages. The approach for processing floor plans is discussed in section 4.1.2. Tools for transforming XML documents into LD are outlined in section 4.1.3.

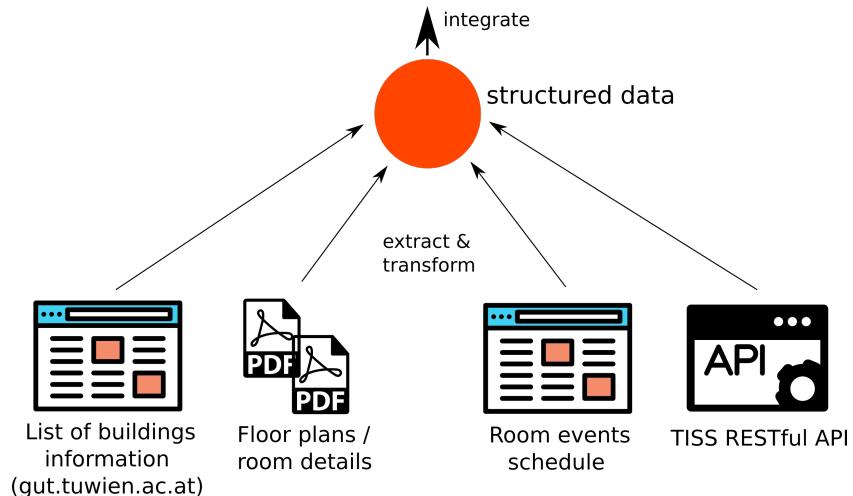


Figure 4.1: Visualization of the data sources

### 4.1.1 (X)HTML and PDF tables

As mentioned earlier, some of the data is presented to users only in form of (X)HTML tables and must therefore be extracted in order to be transformed into LD. Web scrapers/Web harvester are an useful tool to extract such tables from web pages manually or by using XPATH expressions and to store them in an intermediate representation. For this thesis a chrome extension named 'Web Scraper'<sup>3</sup> was used. This intermediate representation was then exported into a Comma-separated values (CSV) file.

For tables in PDF files a different approach is required and the tool named 'Tabula'<sup>4</sup> is qualified for this kind of problem. It can detect tables automatically and transforms them into an intermediate representation, but this does not always work appropriately. In this case, the user can support the program by selecting the relevant parts manually. This intermediate representation was then also exported into a CSV file.

After the information was extracted from web pages and PDF files and transformed into a structured format, it still was messy data that is not convenient for a further transformation into LD. In this case the tool OpenRefine<sup>5</sup> is quite useful. It provides a set of functions to manipulate messy, tabular data including an own expression language named GREL. LODRefine and RDF extension are extensions for OpenRefine, which enable the transformation of tabular data into RDF by mapping columns to classes and

<sup>3</sup><http://webscraper.io/>

<sup>4</sup><http://tabula.technology/>

<sup>5</sup><http://openrefine.org/>

relationships between columns to properties. This mapping is then computed for each row.

#### 4.1.2 Floor plans

Floor plans are ideal for describing the internal structure of a building, but they are mostly available in form of images and vector graphics. This representation is fine for humans, but not for machines; because the floor plan should not only be available as single image, but rather be the knowledge base for applications to find paths and locate rooms in the corresponding building.

Google Maps is a map application that also gives insight into certain buildings. It provides a tool<sup>6</sup>, where any user can submit floor plans and this floor plan then will be automatically transformed into an internal representation that can thenceforward be looked at in the application (if accepted). The problem is that the user cannot demand access to this internal representation of the plan, although the user may be the owner. Figure 4.3 shows the indoor plan of TU Vienna's main library in Google Maps.

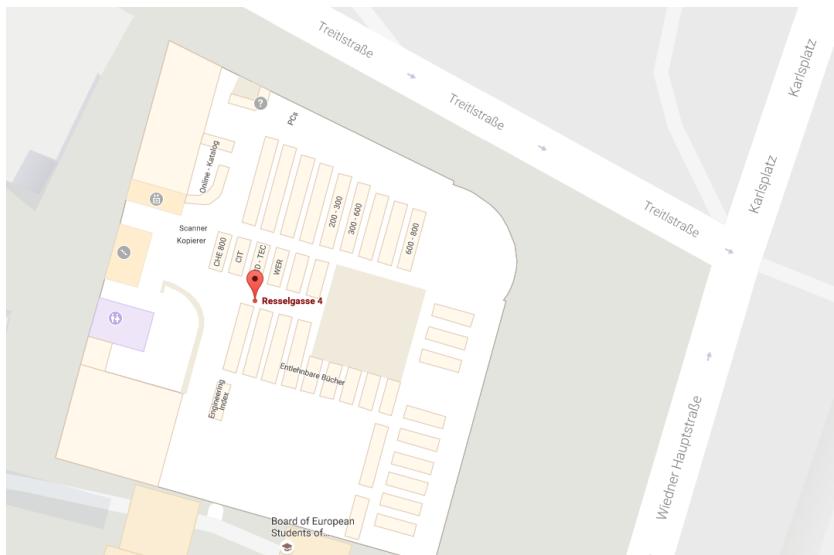


Figure 4.2: Google Maps: Indoor plan of TU Vienna's main library

Due to the lack of alternatives, the approach used for this thesis was to manually transform one floor plan of one building into shape files. The tool used for this approach was the Open Source software QGIS<sup>7</sup>. It has two plugins that are useful for the transformation, the 'Open Layers' plug-in to enable the use of OpenStreetMaps among others and the GDAL georeferencing plug-in to position a raster image on a map. Especially the last plug-in is important for positioning the floor plan to the correct position on the map, whereby

---

<sup>6</sup><https://maps.google.com/floorplans/find>

<sup>7</sup><http://www.qgis.org/>

the map should already contain the boundary of the building. After georeferencing the floor plan, a shape file of the rooms and other *features* of interest can be created. Each of the created shapes has a table of attributes with an unique ID that can be set by the user. The unique ID of each shape is an Internationalized Resource Identifier (IRI), so that the shape is ready to be transformed into LD. The resulting shape file can then be exported in multiple formats including GeoJSON<sup>8</sup>. For this thesis, a CSV file with the WKT serialization of each shape of the file and the corresponding ID per row showed to be the easiest way to integrate this data. Figure 4.3 shows a shape file of rooms on the basement floor of the informatics institute's building at the TU Vienna.



Figure 4.3: Room shapes file of the basement floor of the informatics institute's building

#### 4.1.3 XML

The RESTful API of the central information system TISS returns results of its services either in form of an XML document or JavaScript Object Notation (JSON). Thus the responses are already in a (semi-)structured format; the question is how to transform this data into LD. Extensible Stylesheet Language Transformations (XSLT) offers an useful language for transforming a source XML document into a target document that could be likewise an XML document or other representations like plain text or HTML. The data-management layer (see section 4.3.2) of the proposed system uses this language therefor to transform a XML document with a known schema into a RDF/XML document, which expresses RDF data as an XML document. GRDDL[10] can be used to mark parts of XML documents with pointers to XSLT stylesheets, but this has to be implemented by the API designers.

---

<sup>8</sup><http://geojson.org/>

## 4.2 Prototype of ontology

In this section two ontologies will be proposed that are intended to cover the specific domain of spatial university data. Section 4.2.1 presents an ontology for describing the campus from buildings to indoor environments, whereas section 4.2.2 proposes an ontology for modelling indoor navigation. The design process took best practises for publishing ontologies[5] into consideration. In order to enhance the interchangeability of the described data it is commonly understood to reuse terms of common ontologies, which is why chapter 3 outlined common geospatial ontologies as well as ontologies and approaches for modelling indoor environments. The conclusion of the evaluation was to use the GeoSPARQL ontology for describing the geometry of indoor features and buildings. Hence, the system architecture has to consider an additional prerequisite; the used triple store should support the GeoSPARQL extensions and for the purpose of better scalability also spatial indexing.

### 4.2.1 Spatial ontology

**TODO:** describe ...

### 4.2.2 Navigation ontology

The spatial ontology (see section 4.2.1) provides the necessary expressiveness to describe the indoor environment of buildings from rooms, corridors to entrances, but no assertions can be made about how these *features* are actually connected. Some conclusion can be drawn by looking at the *geometries* of *features*. In order to get for example the two rooms a door is connecting, the distance to the *geometry* of all rooms must be computed and the two rooms with the lowest distance by considering a reasonable threshold (in case one or both rooms are unknown) could be selected. Furthermore, would it be possible to compute the path to a room by considering the *geometries*, but this would still need information about how certain *features* are connected. A door for example could require an access control measure to pass it in one direction, but in the other direction it could be passed without constraints. A further problem to consider, if the path should be computed based on the *geometries*, is that not all paths may be accessible for all persons. A space could contain steps that are hard to overcome for persons with mobility impairments, or the space could be occupied by other obstacles. As an example, directly in front of or behind a door could be some steps that would force such persons to use another entrance.

For this thesis the approach of pre-defining routes based on *features* and points of interest was chosen for the purpose of simplification. However, the previous mentioned techniques could be used to compute certain parts of the route and humans can take over, when it became more complex. Another important question to address is the granularity. Should a route consists of a collection of connected rooms, horizontal passageways like corridors and vertical passageways like elevators or be more detailed with different paths inside of rooms and corridors. The problem of the first approach is that there could be some

#### 4. SOLUTION

---

obstacles inside of rooms and corridors (e.g. steps in the middle of the corridor) that would be overseen. In case of the more detailed routes consisting of a collection of points of interest such obstacles can be taken into consideration, but consequently leads to more verbose data than with the first approach.

Figure 4.4 shows the designed ontology for the indoor navigation. It has two major classes named `NavigationEntity` and `Constraint`. The class `NavigationEntity` has two subclasses namely `Route` and `Access`.

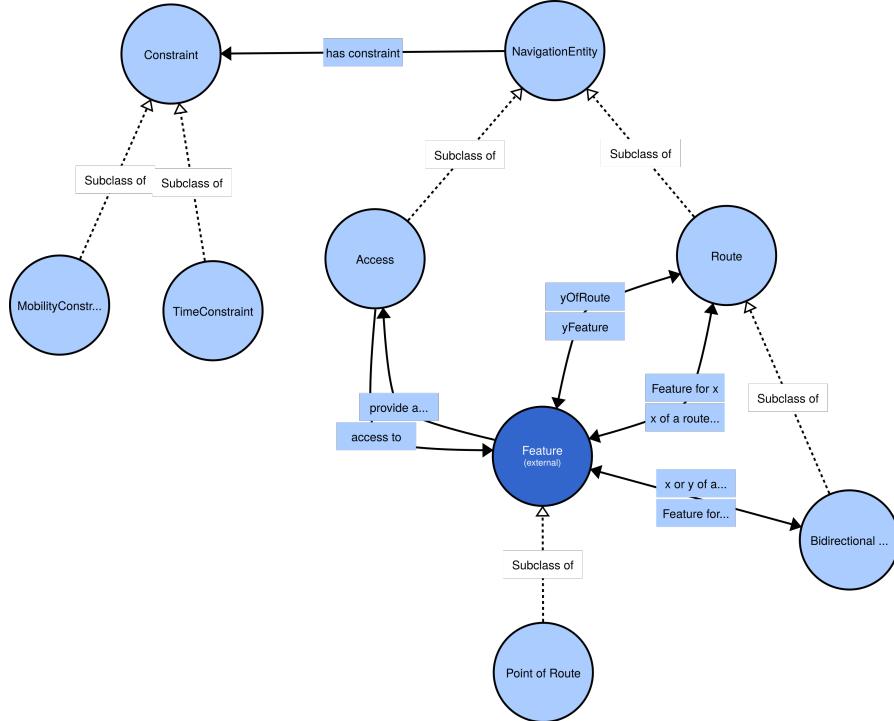


Figure 4.4: Visualization of the navigation ontology

**Route** is a path  $R(x,y)$  between two spatial *features*  $x$  and  $y$ , whereby  $x$  and  $y$  are instances of the class `Feature` of GeoSPARQL. The path is walkable from  $x$  to  $y$  by considering the given constraints of this route. No assumptions about the other direction  $R(y,x)$  are made. The  $x$  and  $y$  values of a route are assigned to it by the properties `xOfRoute` (inverse property `xFeature`) and `yOfRoute` (inverse property `yFeature`). A door that requires an access control measurement in one direction and not in the other is one example of a `Route`.

**BiRoute** is a subclass of `Route` and represents a path  $R(x,y)$  between two spatial *features*, whereas the path between  $x$  and  $y$  is walkable in both directions considering the given constraints. The values  $x$  and  $y$  can be assigned to an instance of `BiRoute` by using the property `xyOfRoute` (inverse property `xyFeature`). This property is

a subproperty of `xOfRoute` and `yOfRoute`; means that each value of `xyOfRoute` is automatically `x` and `y` of a `Route`. Consequently also  $R(x,x)$  and  $R(y,y)$  are valid paths, which must be filtered. A simple path inside of a corridor is an example for a `BiRoute`.

It would be nice to query for the path between two spatial *features* in pure SPARQL. The most recent version of the language (v1.1) introduced a new feature called property paths. These property paths can be used to recursively follow property links[15]. However, property paths lack expressiveness and queries that count the number of possible paths between two nodes or compute the shortest path are not expressible with SPARQL v1.1 [27][2]. Nonetheless, a SPARQL query could prepare the input for a path finding algorithm.

Listing 4.1: Prepare for path finding

```
SELECT ?xRouteFeature ?route ?yRouteFeature ?length WHERE
{
    values (?startFeature ?endFeature) {
        <http://finder.tuwien.ac.at/spatial/accessunit/id/HEEGM1>
        <http://finder.tuwien.ac.at/spatial/elevator/id/HFEG02C>
    }
    # Fetch the features of possible paths
    ?startFeature (navi:xOfRoute/navi:yFeature)+ ?routeFeature .
    ?xRouteFeature navi:xOfRoute ?route ;
        gsp:hasGeometry [ gsp:asWKT ?xRouteFeatureGeom ] .
    ?route navi:yFeature ?yRouteFeature .
    ?yRouteFeature (navi:xOfRoute/navi:yFeature)* ?endFeature ;
        gsp:hasGeometry [ gsp:asWKT ?yRouteFeatureGeom ] .
    FILTER(?xRouteFeature != ?yRouteFeature).
    # Computes the length of the path in meters.
    BIND(gspf:distance(?xRouteFeatureGeom , ?yRouteFeatureGeom , uom:metre)
        as ?length) .
    # Remove all routes that have certain constraints.
    FILTER NOT EXISTS {
        ?route navi:hasConstraint ex:building-id-H-opening-hours .
    }
}
```

Listing 4.1 shows a query returning a result set of all *features* (`xRouteFeature`) that are part of a possible path between two given *features* (`startFeature`, `endFeature`) and all the adjacent, noteworthy *features* (`yRouteFeature`) for each of them. An adjacent *feature* (`yRouteFeature`) is noteworthy, if there is a possible path to the target *feature* (`endFeature`) and the route  $R(xRouteFeature, yRouteFeature)$  has no undesirable constraints. Additionally, the length will be determined for each route in order to have a weight for the path finding algorithm. The length is simply the distance between the `x`- and `y-feature` of a route.

**Access ...**

**TODO: finish ...**

### 4.3 Architectural prototype

This section proposes a architectural prototype of a software system that manages spatial LOD from integration to publication. Figure 4.5 shows an overview of this architecture, which is separated into three layers; the data management, service and presentation layer. The shown data sources (described in section 4.1) build the knowledge base of the system. Section 4.3.2 discusses the data management layer, which deals with the integration of those data sources and the maintenance of it. The linking of the local spatial LD with external sources like LinkedGeoData and GeoNames is described in section 4.3.3. After the local spatial data has been integrated into the system, the resulting LD can be published. Section 4.3.4 discusses different approaches for the publication with the goal to make the underlying LD easily accessible by machine and human agents. The prototype of a map application that make use of the provided services of the system is presented in the subsequent section 4.4.

In order to implement the suggested prototype the programming language Java was used and section 4.3.1 outlines the common frameworks for handling LD in this language. It also describes the reasoning, why RDF4J<sup>9</sup> was chosen.

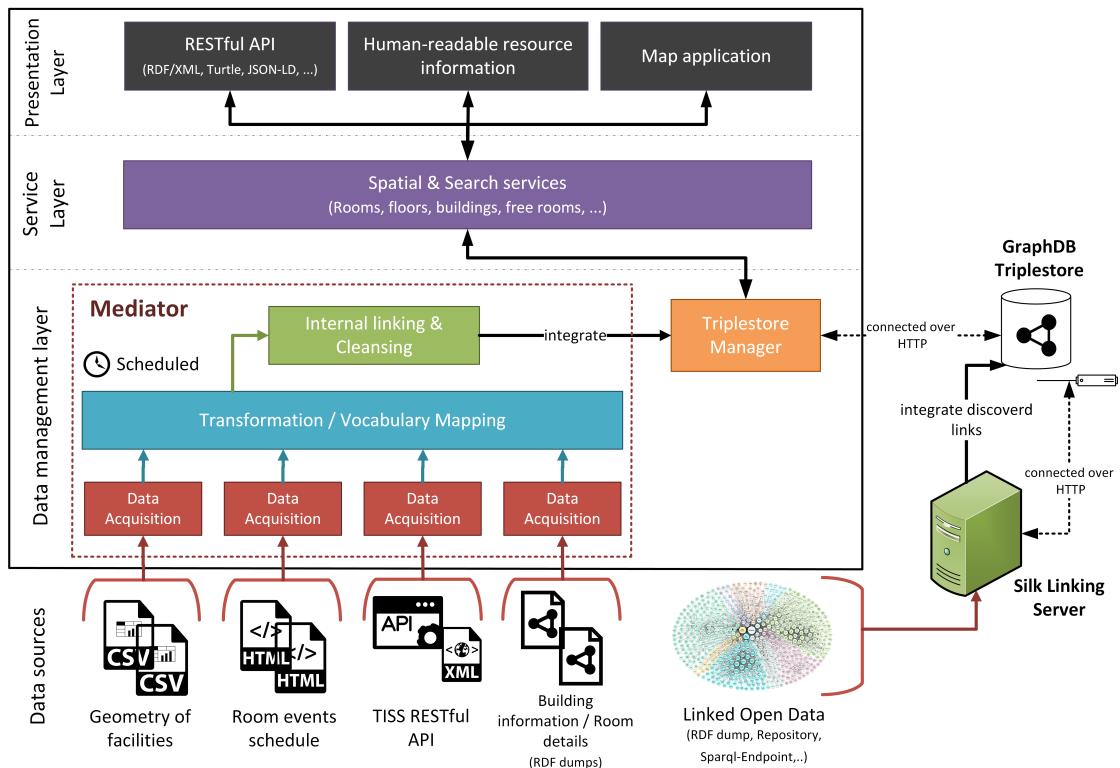


Figure 4.5: Overview of the system architecture

<sup>9</sup><http://rdf4j.org/>

### 4.3.1 RDF Framework

Handling RDF data (as representation of LD) is an essential part of the proposed software system and a reliable framework for fulfilling this task is needed. Besides reliability the framework should also support most of the following requirements:

1. it should be freely available, favourably Open Source.
2. it should be possible to store data into either an embedded or external triple store.
3. it should support SPARQL to query data.
4. it should support common serialization formats like RDF/XML, Turtle, LD-JSON and N-Triples.
5. it should support spatial functionalities in SPARQL queries compliant to GeoSPARQL (see section 3.1). At least the minimal distance between two geometries must be computable.
6. it should be possible to reason over stored LD given a list of ontologies.

There are currently two common frameworks for handling RDF data in the Java environment namely Apache Jena<sup>10</sup> and Sesame, which has recently been migrated to the project RDF4J<sup>11</sup>. Both are Open Source, but none fulfils all of the mentioned requirements, why it is also necessary to take plug-ins and external triple stores into consideration.

**Apache Jena** is a framework that can be used to store, manipulate and query LD. Its architecture consists basically of three layers, the store API at the bottom, the inference API in the middle, the RDF API and SPARQL API respectively as interface to the application code [31]. In contrary to RDF4J, RDF data can be parsed into a model instance and thenceforward this model can be used to query and reason over it. RDF4J requires this RDF data to be at least stored into a repository embedded in memory in order to be queried using SPARQL. One advantage of Jena over RDF4J is also the built-in reasoner that supports simple RDF Schema (RDFS) reasoning as well as multiple profiles of Web ontology language (OWL) [32]. Jena supports different types of storage from native triple store to a memory based one; also custom triple stores can be used, but in this case an adopter is required. In case of spatial functionalities Jena does support spatial relationships like near by, within box, intersect box or north/south/west/east of [33], but it is not compliant to the GeoSPARQL standard . Functions like computing the minimal distance between two geometries are not covered. There is also no support for GML serializations of geometries. As a consequence requirement **5** is not fulfilled. A way to circumvent this problem is to use an external triple store that provides these spatial functionalities, but as mentioned above this triple store must offer an adaptor for Jena, which limits the number of considerable triple stores.

---

<sup>10</sup><https://jena.apache.org/>

<sup>11</sup><http://rdf4j.org/>

**RDF4J** is as well a framework to store, manipulate and query LD, but there are conceptual differences. As mentioned above in RDF4J actions are carried out against repositories and a model has less functionality than in Jena; it is simply a collection of statements (triples). RDF4J provides RDFS reasoning, but nothing more expressive than that. However, custom reasoners can be added like in Apache Jena too. An advantage of RDF4J over Jena is the simplicity of adding other storages besides the built-in ones (from native triple stores to in-memory repositories). It provides for example classes to use a SPARQL endpoint as repository. In fact there are multiple external triple stores that can be used with RDF4J over HTTP like GraphDB<sup>12</sup>, Stardog<sup>13</sup> or Blazegraph<sup>14</sup>. RDF4J does have a package named `rdf4j-queryalgebra-geosparql` to add spatial functionalities compliant with GeoSPARQL, but during testing the minimal distance function always returned null. It was not clear, if there was a bug in the framework or it was used incorrectly. This leads to a major drawback of RDF4J in contrast to Jena, it has a sparse documentation at the moment of writing.

### Conclusion

The support of spatial functionalities in SPARQL queries like computing the minimal distance between two geometries is crucial for the proposed software system, not only for the navigation task, but also for spatial linking of resources to external sources (see section 4.3.3). In this regard both frameworks would be insufficient, but custom/external triple stores can help here. GraphDB is a triple store that is able to take care of reasoning (with support of RDFS as well as profiles of OWL). Additionally, it offers a GeoSPARQL plug-in that not only adds the corresponding spatial functionalities, but also spatial indexing resulting in a performance gain. This is the reason why RDF4J was chosen as framework, because it is easier to use such an external triple store, which then can also mitigate some of the drawbacks of RDF4J.

#### 4.3.2 LD Management

At the bottom of the proposed, layered software system is the data management layer, which is responsible for integrating and maintaining LD. The first step is called mediation and it deals with the problem of integrating data from different data sources and updating them in a frequency that fits the characteristics of the dataset. The triple store manager controls the access to the used triple store. For this thesis, GraphDB was chosen as explained in the previous section 4.3.1, but the idea of the triple store manager is to hide this fact; also other storage solution that are compatible with RDF4J and fulfil the mentioned requirements can be used. All these aspects are outlined in the following in more detail, starting with the designed naming scheme for resources.

---

<sup>12</sup><http://ontotext.com/products/graphdb/>

<sup>13</sup><http://docs.stardog.com/>

<sup>14</sup><https://www.blazegraph.com/product/>

Name of graph	Description
<base>/spatial	Dataset containing statements about resources of the spatial domain.
<base>/organizational	Dataset containing statements about resources concerned with organizational information like persons, institutes and faculties.
<base>/event	Dataset containing statements about resources concerned with events like lectures.
<base>/catalog	Dataset containing statements describing the mentioned datasets (named graphs) like f.e. last update or frequency of changes. This dataset can be used as starting point to explore the datasets mentioned above and to inform yourself about available distributions.

Table 4.1: Named graphs used in the proposed software system

### Naming scheme

In order to handle a larger amount of LD a consistent naming scheme for resources is inevitable. This also helps to simplify the design of a RESTful API (see section 4.3.4). As suggested by the first two principles of LD brought forward by Tim-Berners-Lee [4], unique IRIs should be used to name resources and be based on Hypertext Transfer Protocol (HTTP) so that people can look up those resources. A driving factor for exposing data as LD is the idea of lowering bars for innovative, motivated developers for coming up with useful applications using the exposed resources. Furthermore other datasets may link to those resources, which is why it must be considered that the designed IRIs are as stable and persistent as possible. However, this issue goes beyond the designed software system and has to include administrators of the infrastructure, where the system is going to be deployed. For this thesis the base IRI was chosen in advance to be `http://finder.tuwien.ac.at/`, but it is not hard-coded and can be changed before deployment.

The data that will be integrated into the system is predominately spatial data, but also organizational data and information about events are of interest for certain services. Information about events for instance can be used to determine which lecture room is available at a given time. All of this different types of data will be stored into different named graphs. Table 4.1 shows the chosen naming scheme for the named graphs. The catalog graph contains metadata about the different named graphs as suggested by best practises for data on the Web[37]. This metadata includes general information (like title, keywords, publisher, frequency of changes), license information, quality details and available forms of distribution. The data catalog vocabulary[22] was used to provide this information. This data set can then be used by both humans and machines to gather information about the provided data and distribution.

The suggested naming scheme of resources is then considering to which named graph the resource belongs. Table 4.2 shows the scheme for spatial resources. The IRIs are intended to be self-descriptive and it is assumed that each resource has an unique identifier. In case of rooms, buildings and floors it is easy, because the TU Vienna has already specified an unique identifier for those spatial entities and this is also reflected in the gathered data. Assigning an identifier to resources like addresses is more complicated, due to the lack of a predefined global identifier; this leads to the question how addresses that are considered to be equal, but gathered from different data sources, should be identified? On one hand a string consisting of a part that identifies the data source uniquely and a second part that identifies the resource inside of this data source uniquely can be used, whereby internal linking must be computed afterwards. The transformation process becomes easier with this approach, but consequently `owl:sameAs` statements will be produced during the linking, which could have an impact on the querying performance. The approach used for address resources in this thesis was a combination of concatenating key properties and hashing. This makes the transformation process more complicated, but prevents unnecessary `owl:sameAs` statements. The address identifier consists of the country code, the postal code, a MD5 hash of the street name and eventually the locator. The address '*Favoritenstraße 9-11, 1040 Vienna, Austria*' for example has the identifier AT1040-3ebd6229134d2d9c36a91657af166825-9-11. Although, a collision may occur due to hashing, it is unlikely and easily detectable by checking if the affected address resource has two different street names. Internal linking is still necessary due to addresses that may have multiple spellings, but the amount of `owl:sameAs` will be reduced.

The idea of the naming scheme was not only to be self-descriptive, but also to be easily extensible for the RESTful API. The IRI '`<base>/spatial/room/id/<id>/address`' calls for instance the service that returns the default address of the room with the given identifier in a requested format (e.g. Turtle). The triple store does not store this information for each room in a building, it can be computed on the fly by using already persisted information. Section 4.3.4 goes into more detail.

### Mediator

As mentioned above, the mediator is responsible for the data acquisition and thenceforward for processing and transforming the gathered data into the desired LD representation. Finally, the resulting LD will be stored into the local triple store. More precisely, it consists of the steps *data acquisition, transformation, cleansing, internal linking* and *persisting*. The idea is not to have a single mediator for all data, but to have specialized mediators for different types of data and/or data sources. An important characteristic of data is the accrual periodicity (can be found in the `catalog` dataset), which determines how often the data in question changes. Spatial data concerning buildings does not change as often as data about events. The approach for updating could furthermore differ from data type to data type and source to source respectively. If events are gathered by requesting a XML dump for a given time range, this dump might not contain explicit

Naming of spatial resource	Description
<base>/spatial/building/id/<id>	Describes building with the given id.
<base>/spatial/building/id/<id>/address	Describes the default address of the building with the given id.
<base>/spatial/floor/id/<id>	Describes floor with the given id.
<base>/spatial/room/id/<id>	Describes room with the given id.
<base>/spatial/room/id/<id>/address	Describes the default address of the room with the given id.
...	

Table 4.2: Naming scheme for spatial resources

information about changes since the last request. Thus the changes must be determined by the mediator through comparison of the local triple store and the dump (e.g. an event was cancelled or moved to a different date/time). In order to consider all these differences, the mediator is designed to be composable. Figure 4.6 shows the UML class diagram of the mediator. The `mediate()` method simply executes the given number of `DataAcquirer` instances concurrently using the `CompletionService` introduced in Java 7, transforms there result into a LD representation by using the specified `DataTransformer` instance of the corresponding `DataAcquirer` (see figure 4.8) and combines all those resulting LD models to a single one. Afterwards this single model will be integrated into the local triple store using the given `DataIntegrator`. The *internal linking* and *cleansing* is part of the `DataIntegrator`.

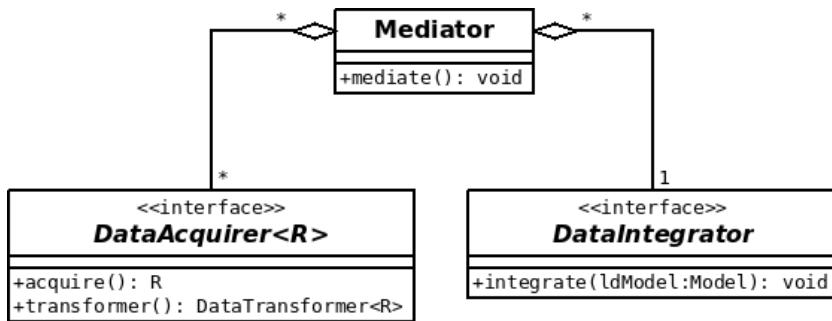


Figure 4.6: UML class diagram of the mediator

## 4. SOLUTION

---

**Data acquisition** is the first step of the mediation process. According to Simperl et.al.[28] [16] there are in general three architectural patterns that are used by LD applications; *crawling pattern*, *on-the-fly dereferencing pattern* and *federated query pattern*. The choice depends on a number of factors like the desired response time of the data management layer or how up-to-date the data must be. Applications using the *crawling pattern* are like Web crawlers designed to harvest RDF data on an open, growing set of resources; also new resources can be discovered at run-time. This leads to the disadvantage that the data may not be up-to-date, when it is accessed. The *on-the-fly dereferencing pattern* on the other side is acquiring data and follows links at the moment when it is requested. As consequence, the result is up-to-date, but the response time may be slow in dependence of the data sources that must be accessed. The *federated query pattern* describes the approach of sending complex queries to a fix set of data sources that expose their data over a SPARQL endpoint. However, the data sources do not provide a SPARQL endpoint, which is why this pattern is not applicable. In case of the proposed software system the response time is crucial, because it is intended to provide access to spatial data of the TU Vienna for other applications (like the map application proposed in section 4.4). The approach followed by the proposed software system is therefore similar to the *crawling pattern*, but the number of data sources is fixed and no links outside of this sources will be followed. The disadvantage of stale data is a minor issue in case of spatial data, which does not change frequently, but a greater one for event and organizational data. The fact that the *crawling pattern* requires replicated data is a gladly accepted side effect, if this leads to a lower response time. Figure 4.8 shows a UML class diagram of the DataAcquirer, which can have multiple implementations. Such an implementation might for example be a automated Web scrapper, which is extracting (X)HTML tables from a Web page and stores them into a XML document. Each DataAcquirer has exactly one DataTransformer capable of transforming the acquired data into LD.

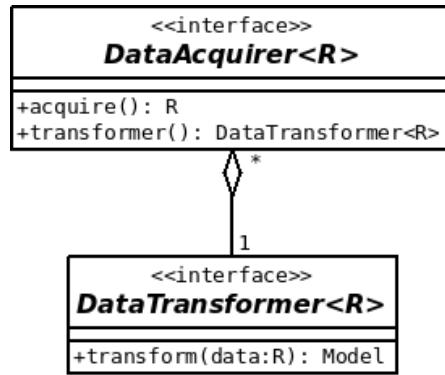


Figure 4.7: UML class diagram of the data acquire

**Transformation/Vocabulary Mapping** is the next step in the mediation process and it deals with the issue of transforming the intermediate representation of the data

into the system's target schema (see proposed ontologies in section 4.2). This is the task of the DataTransformer of which multiple implementations can exist. In case of the TISS RESTful API, the intermediate representation of the acquired data is XML and a corresponding DataTransformer implementation could use XSLT to transform it into RDF data.

**Internal linking** is the third step of the mediation process. In this step specific relationships between resources of the local triple store and the resulting LD of the previous steps will be discovered. In case of spatial data of the TU Vienna a room can be linked to the floor section or building tract of which it is a part by taking a look at the room number. The first two symbols are an indicator for the building tract and the two following ones for the floor section; HHEG01 is a room located in building tract HH and floor section HHEG on the basement floor (indicated by EG). Linking to external sources like GeoNames and LinkedGeoData is not part of the process; it is computed by an external application (see section 4.3.3).

**Cleansing** is the fourth step of the mediation process and is responsible for cleaning up the data resulting from the previous steps before or after it is stored; for example to remove ambiguities.

**Persisting** is the final step in which the data is eventually persisted. Figure shows the UML class diagram of the DataIntegrator, which is in charge of carrying out the three tasks *internal linking*, *cleansing* and *persisting*. The DataIntegrator has a reference to the TripleStoreManager, which is a singleton class that provides a getConnection() method, but is hiding all other information about the used storage method. The DataIntegrator implements the storing of certain LD and consists of DataLinker as well as DataCleanser, whereby a DataIntegrator instance can also have none of them and only store given data. The integrate() method simply stores the given data and executes the given instances of DataLinker as well as DataCleanser.

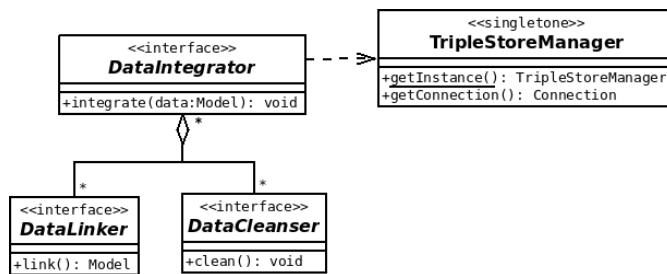


Figure 4.8: UML class diagram of the data integrator

## Result

DataAcquirer and DataTransformer aim to provide a unique view of the different data sources with diverging formats mentioned in section 4.1. The DataTransformer transforms the acquired data into RDF data compliant to the proposed naming scheme and ontologies. After the transformation, the resulting LD can then be used for internal linking and eventually be persisted by using a DataIntegrator. After the integration the overlying service layer can query this data and provide certain services like where is the lecture hall with a certain name located.

### 4.3.3 Linking

The first two principles of LD were mentioned in the previous section, when the naming scheme was discussed. A further important principle is interlinking the local resources with other local resources and external ones, so that more things can be discovered [4]. There are basically two ways to generate links between resources; setting them manually or auto-generating them. However, setting them manually is not feasible, due to large amount of spatial resources that could be part of this software system. Hence, the links must be generated automatically. There are in general two approaches; a *key-based approach* or a *similarity-based approach*[16]. The *key-based approach* is applicable for resources that share identifiers with a well-known naming schemata; for books this could be ISBN. In case of spatial data of the TU Vienna the identifier of the building units (rooms, floors, building tracts) can be used internally for linking for instance rooms to the building tract of which they are a part. This section aims to deal with interlinking local resources with external LD like GeoNames or LinkedGeoData, which leads to the issue that there is no such common naming schemata for places, addresses or other spatial entities, so that the *similarity-based approach* must be applied. Using this approach linkage heuristics that compare one or more properties of resources are employed and links are generated, if the heuristic results in a confidence value for two resources that exceeds a given threshold.

In order to accomplish this task the Silk discovery framework[36] was used and as the visualization of the software system shows (see figure 4.5) its standalone-machine is directly communicating with the GraphDB triple store over HTTP; making use of the SPARQL endpoint of the triple store. The framework provides a set of numeric, string as well as geographical matchers that can be combined to a heuristic. A heuristic basically consists of a transformation step, a comparison step and eventually an aggregation step for combining the similarity scores of comparisons. In the following the linking of spatial *features* of the proposed system and GeoNames as well as LinkedGeoData is discussed.

#### GeoNames

GeoNames is a geographical database with over 10 million geographical names and over 9 million unique *features*; it is available free of charge under a creative commons license [36]. DBPedia among others has links pointing to GeoNames, which makes it

an attractive target for linking spatial *features* of the proposed system to *features* in GeoNames database.

GeoNames does not expose its LOD over a SPARQL endpoint, but instead provides a RDF dump for download<sup>15</sup>. However, the dump is not fully compliant with RDF, it is rather a text file with a list of RDF snippets (see appendix for a UNIX command to transform it into a single RDF file). The dump is quite large and therefore not suitable to be used directly with the Silk framework, wherefore the dump was stored into the local triple store in a separate repository; this repository provides then a SPARQL endpoint.

**Equivalent buildings** of the proposed system and GeoNames shall be detected. In case of a match a `owl:sameAs` link shall be generated to indicate that both resources reference to the same thing — same building. Each feature in GeoNames has a `featureCode` and `featureClass`. The `featureClass` of a feature describes the category it belongs to like spot (buildings, places), parks, or forest, whereas `featureCode` describes the more precise subcategory it belongs to like university or hotel. The `featureCode` is the concatenation of the category and subcategory separated by a point. University buildings for example have the `featureCode` `S.UNIV`, where `S` stands for spot and `UNIV` for university. This is important because the designed heuristic has to ensure that both resources reference to the same thing — a building of a university. Figure 4.9 shows the designed heuristic. The proposed system uses GeoSPARQL properties and WKT literals to express the geographical location of a building, whereas GeoNames simply uses the (long, lat) properties of GEO. The geotransformer of the Silk framework transforms this different representations into an intermediate one that can be compared by the provided geographic matchers. To find equivalent *features* the minimum distance between the two observed *features* will be computed and if the distance is lower than 15 meters than it is considered to be a candidate; the two other comparison must also hold true.

**Features that are nearby** buildings of the proposed system shall be detected and for each of them a `foaf:based_near` link shall be generated. This property of the FOAF ontology describes “*A location that something is based near, for some broadly human notion of near.*”[9]. Figure 4.10 shows the designed heuristic for finding spatial *features* nearby. The heuristic defines that near is less than 100 meters away. The `featureClass` and `featureCode` are ignored, because it is not important of which type the detected *features* are; they can be super markets, parks or restaurants.

## LinkedGeoData

LinkedGeoData uses the spatial data collected collaboratively by the OpenStreetMap<sup>16</sup> community to create a spatial knowledge base; it consists of 3 billion nodes and 300 million ways. The data is interlinked with DBPedia and GeoNames [34]. LinkedGeoData provide more interesting *features* to interlink with than GeoNames from the perspective

---

<sup>15</sup><http://download.geonames.org/all-geonames-rdf.zip>

<sup>16</sup><https://www.openstreetmap.org>

## 4. SOLUTION

---

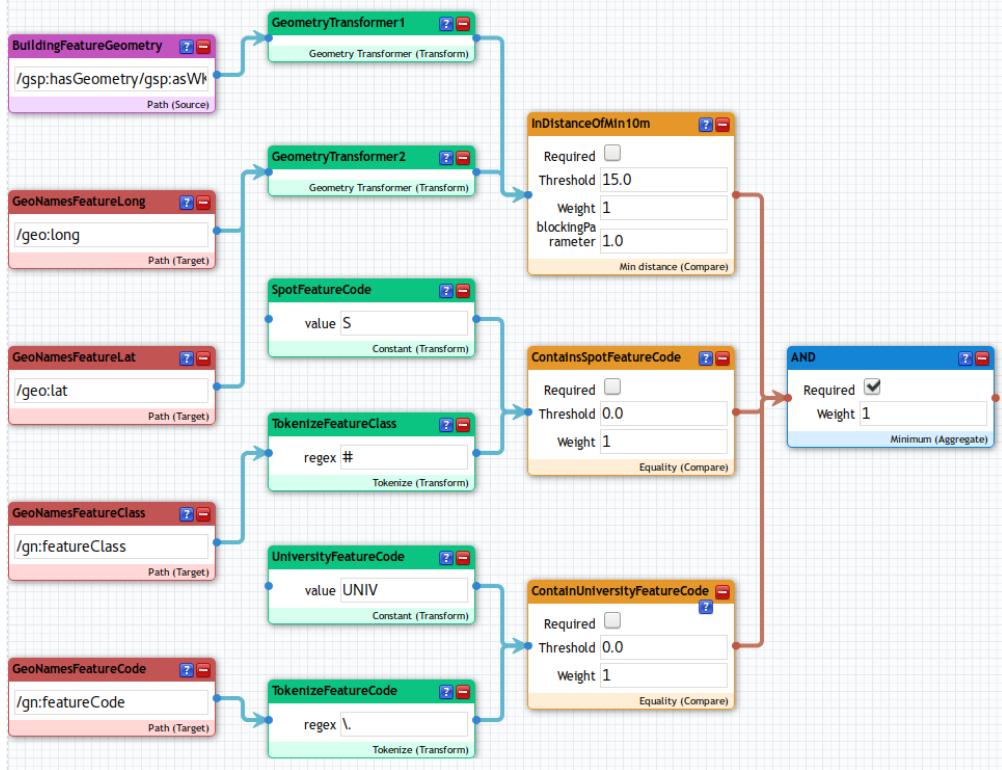


Figure 4.9: Heuristic for detecting equivalent buildings (GeoNames).

of an end user. GeoNames *features* near TU Vienna's buildings are mostly hotels, which are likely not of interest for students, lecturers and university employees in general. LinkedGeoData on the other side provides *features* like restaurants, cafés, super markets and bars.

LinkedGeoData does provide a SPARQL endpoint, which was in the time range of this thesis not reachable from time to time. Alternatively there is also a RDF dump available, which is quite large; the data overall consists of 20 billion triples. The huge amount of *features* (approximately 3 billions) leads to the challenge of designing queries that only gather the relevant *features* by not causing a time out or unreasonable run times. LinkedGeoData has in contrast to GeoNames, where each *feature* has a *featureCode* that describes the category it belongs to, a number of classes of which a *feature* can be a member. For the following heuristics only *features* that are members of the class *Amenity*<sup>17</sup> will be considered. A spatial *feature* in LinkedGeoData can furthermore follow two approaches to express its location and *geometry* respectively. Its *geometry*, if available, is represented in form of a WKT literal using the *Geometry* classes of Geosparql and the *asWKT* property. The *geometry* property of NGEO is used to point to this instance. A location point, if available, is expressed using the (*long*, *lat*)

<sup>17</sup><https://en.wikipedia.org/wiki/Amenity>

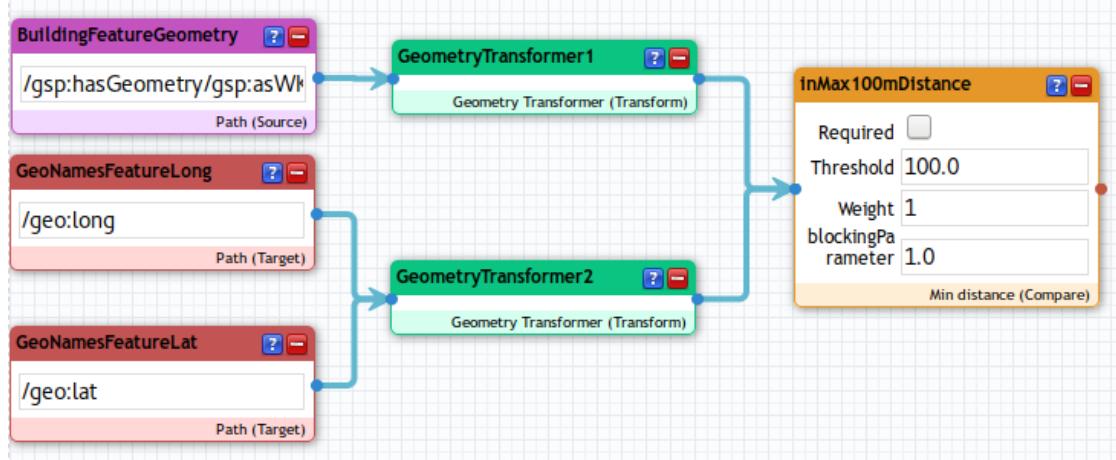


Figure 4.10: GeoNames features that are nearby (max. 100m)

property of GEO. Listing 4.2 shows a SPARQL query to check, if there is *feature* that has a *geometry* instance, but no location point; the result is **true**, so yes. Listing 4.3 shows a SPARQL query to check the contrary, if there is *feature* that has a location point, but no *geometry* instance; the result is **false**, so no. Thus there is no need for designing a query that considers both approaches in order to guarantee that no *feature* will be overlooked; it is sufficient to look at the *geometry* instances.

Listing 4.2: Ask if there is a feature with a geometry instance that has no location point.

```
ASK {
    ?a a <http://linkedgeodata.org/ontology/Amenity> ;
        ngeo:geometry [ gsp:asWKT ?amenityGeometry ] .
    FILTER NOT EXISTS {
        ?a geo:long ?long ;
            geo:lat ?lat .
    }
}
```

Listing 4.3: Ask if there is a feature with a location point that has no geometry instance.

```
ASK {
    ?a a <http://linkedgeodata.org/ontology/Amenity> ;
        geo:long ?long ;
        geo:lat ?lat .
    FILTER NOT EXISTS {
        ?a ngeo:geometry [ gsp:asWKT ?amenityGeometry ] .
    }
}
```

As a result the Silk framework is configured to take the *features* returned by the SPARQL query shown in listing 4.4 as input for the following heuristics. The problem is that the SPARQL endpoint only returns 10,000 results, due to an error “*Virtuoso 22023*

*Error SR353: Sorted TOP clause specifies more then 11000 rows to sort. Only 10000 are allowed. Either decrease the offset and/or row count or use a scrollable cursor*<sup>18</sup>. The issue has been closed, but it is not part of the last stable release of Silk (2.7.1) at the moment of writing.

Listing 4.4: Query for preparing the input for linking.

```
SELECT DISTINCT ?a ?v0 WHERE {
    ?a ngeo:geometry > ?t1 .
    ?t1 gsp:asWKT ?amenityGeometry .
    FILTER (<bif:st_intersects>(?amenityGeometry,
        <bif:st_point>(16.37001, 48.198891), 10.0)) .
}
```

**Equivalent buildings** of the proposed system and LinkedGeoData shall be detected. In case of a match a `owl:sameAs` link shall be generated to indicate that both resources reference to the same building. The heuristic is similar to GeoNames solution regarding this issue, whereas they differ in the way they ensure that links are only generated for resources referencing to the same thing — a university (building). In case of LinkedGeoData it will be checked if a resource is a member of either `BuildingUniversity` or `University`. If this holds true and the resource is in near distance (max. 15m), then a match has been detected. Figure 4.11 shows the designed heuristic.

**Features that are nearby** buildings of the proposed system shall be detected and for each of them a `foaf:based_near` link shall be generated. The heuristic is similar to the heuristic for GeoNames regarding this issue; it is shown in figure 4.12.

## Result

After the computation of the mentioned heuristics buildings of the proposed systems are linked to equivalent buildings of GeoNames and LinkedGeoData. In case of GeoNames, there was only one `owl:sameAs` link generated between the main building of the TU Vienna and the equivalent in the GeoNames dataset, but several `foaf:based_near` links, which mostly points to hotels next to university buildings. The detected spots in case of LinkedGeoData on the other side were much more diverse; from restaurants to bars and cafés. These spots could be actually valuable for the end user to discover. See section 5.1 for a visualization of those links.

### 4.3.4 Services & Linked Data Publishing

In the last sections three principles of LD as they were proposed by Tim Berners-Lee have already been discussed. The final one will be the matter of this section, “*when someone looks up a IRI, provide useful information using the standards (RDF, SPARQL)*”[4]. In the proposed software system resources are exposed over a RESTful API that either

<sup>18</sup><https://github.com/silk-framework/silk/issues/31#event-797295779>

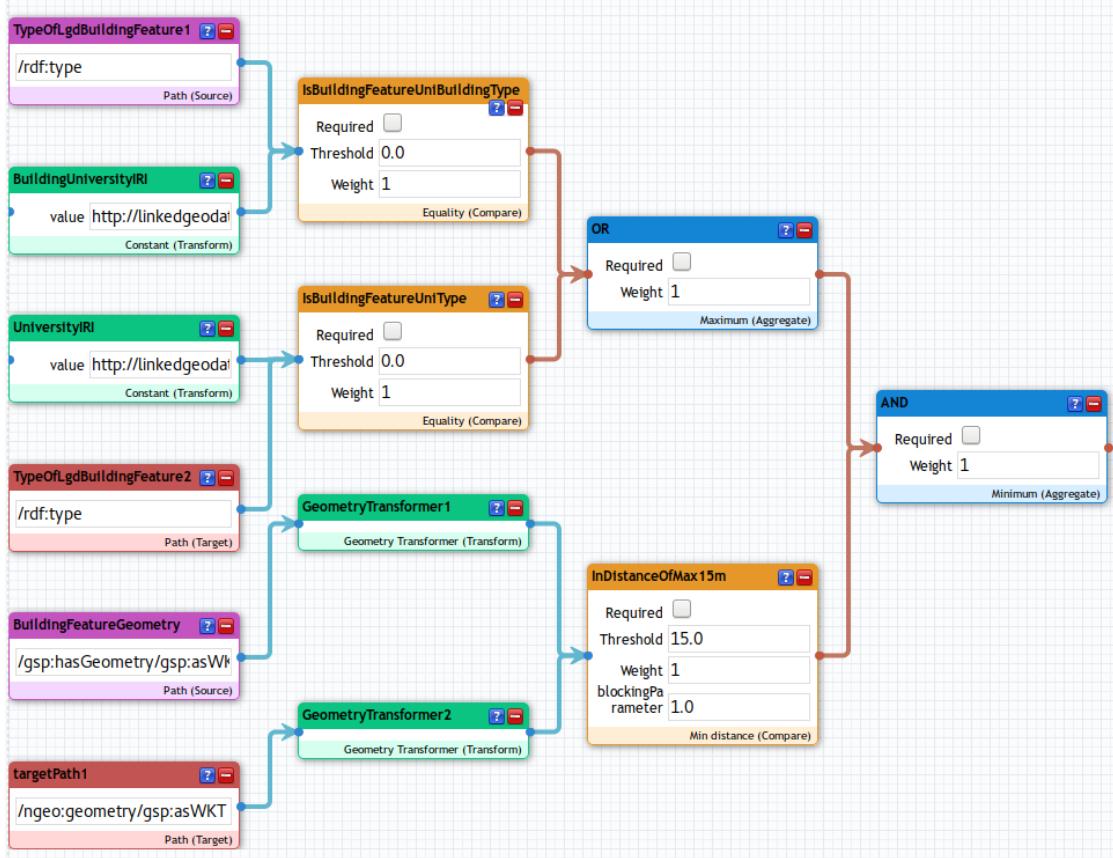


Figure 4.11: Heuristic for detecting equivalent buildings (LinkedGeoData).

returns a human-readable description of the requested resource or a machine-readable one in the requested RDF format. The desired format is detected by analysing the header of the HTTP request; a mechanism called content negotiation. But the API does not only provide access to the description of resources, but also to services like give me all rooms of a certain building tract. Before the resources of the proposed system can be exposed over such an API, the service layer needs to be discussed and data transfer objects that hide the persistent strategy from the overlying presentation layer must be designed in order to be compliant with the applied multi-tiered architectural pattern.

### Service layer

The service layer is responsible for handling all the services invoked by the RESTful API or map application (see section 4.4). Figure 4.13 shows the approach used for the service layer. Services are requested by passing the path of the corresponding IRI followed by the base (in case of this thesis <http://finder.tuwien.ac.at/>) to the root `IServiceFactory`. A `IServiceFactory` is assigned to each segment of a valid path except segments representing a parameter. These service factories then have a map

#### 4. SOLUTION

---

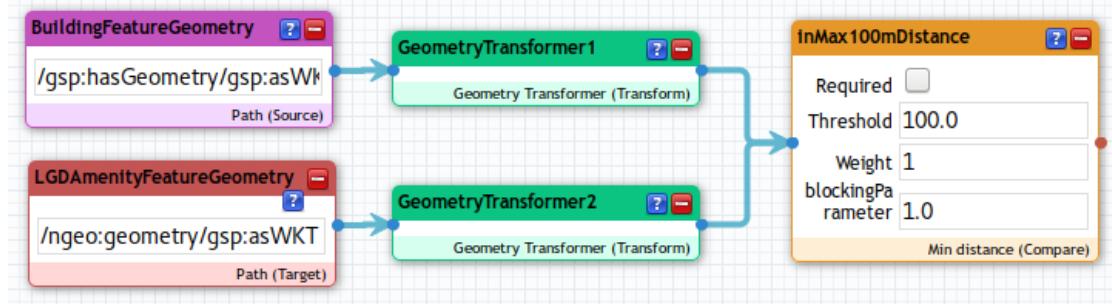


Figure 4.12: LinkedGeoData features that are nearby (max. 100m)

of all `IServiceFactories` representing path segments that can follow the current path segment. This leads to the advantage of helpful messages for users of the API that are easy to implement. An invalid request like `http://finder.tuwien.ac.at/spatial/r` invokes following error message.

The given IRI '`http://finder.tuwien.ac.at/spatial/r`' is not valid. Possible continuations of '`http://finder.tuwien.ac.at/spatial`' are `..buildingtract`, `..rooms`, `..address`, `..building`, `..room`, `..por`, `..route`, `..elevator`, `..accessunit`, `..buildings`, `..geometry`, `..stairway`, `..floor`.

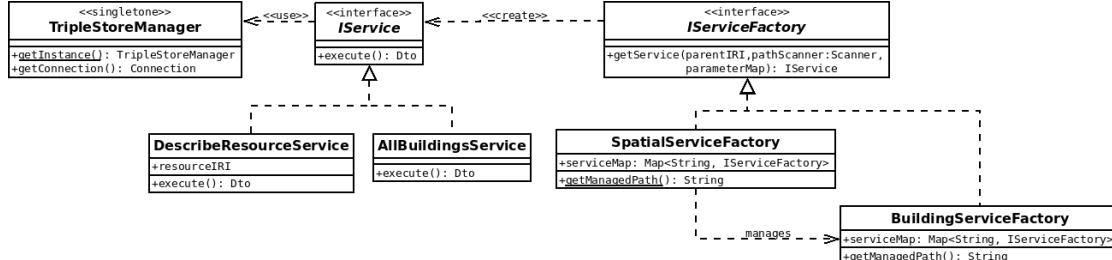


Figure 4.13: UML class diagram of the service layer

The disadvantage of this approach is that it requires a lot of factories that are only assigned to a single path segment. Shared functionalities can be inherited from an abstract class and may not be implemented by each factory, but it still limits the maintainability. Dependency injection would be an alternative approach that could be used. The Play framework, which is used in the presentation layer, follows this principle and provides a route file in which the path and the corresponding function that is able to process the service expressed by the given path can be specified; the dependency injection is then handled by the framework. This increases the maintainability and in general the testability. Expressive help messages are not as easy to implement in comparison to the chosen approach.

**As example** the request `http://finder.tuwien.ac.at/spatial/building/id/H` consists of the base and the path `spatial/building/id/H`, which is passed to the root `IServiceFactory` using the method `getService(...)`. This root service factory manages a set of `IServiceFactories` including an instance of `SpatialServiceFactory`, which is responsible for the path segment `spatial` and on its hand manages a set of `IServiceFactories` representing possible continuations of the path (`building`, `room`, `floor`, etc.). The root service factory calls the `getService(...)` method of the spatial factory and this is done recursively until a service factory detects that there is no further path segment or there is no managed `IServiceFactory` for the given continuation; parameter segments like in this case `H` are simply consumed and stored into a parameter map. In the second case (invalid path) an error message including possible continuations will be thrown as shown above. In the first case the corresponding `IService` will be returned; considering the given example, the service implementation would simply be a `DescribeResourceService`.

A `IService` has a reference to the `TripleStoreManager` and requests a connection to the triple store, when needed. Most of the services are implemented using SPARQL queries; the mentioned `DescribeResourceService` uses the `describe` operator of SPARQL. The result of services is then mapped into the corresponding `Dto` in order to be compliant with the multi-tiered architecture and to simplify the presentation, but there is like with relational databases an impedance mismatch between LD and Java that must be considered.

**D<sub>to</sub> & Impedance mismatch:** The service layer directly queries the given LD, which is why no domain objects are required to interact with the underlying data-management layer. However, the resulting LD describing entities like rooms and buildings must be mapped into corresponding objects that can be passed to the presentation layer; data-transfer objects are a common pattern for this issue and are used in this proposed system as well. `Dtos` allow additionally to minimize the amount of required requests; a `Dto` describing a certain building could contain information about its rooms and floors including *geometries*, so that not several requests are required to gather all the information. Figure 4.14 shows the UML class diagram of a portion of the used `Dtos`. There is still an issue that has not been solved yet and this is the question of how the `Dtos` are assembled. One possibility is to implement the assembler manually, which is prone to failures and repeating, tedious work to do. It is also not necessary, due to available frameworks that facilitate the assembling. Empire<sup>19</sup> as well as Alibaba<sup>20</sup> are frameworks enhancing Sesame (now RDF4J) with functionalities that are required for designing more complex applications based on LD. Empire aims to be the equivalent of JPA<sup>21</sup> only for RDF data stores. While both frameworks provide the ability to map RDF data into Java objects, they are not yet compatible with RDF4J and would be

---

<sup>19</sup><https://github.com/mhgrove/Empire>

<sup>20</sup><https://bitbucket.org/openrdf/alibaba>

<sup>21</sup><http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>

## 4. SOLUTION

---

an overload for the intended use case. For this thesis the framework Pinto<sup>22</sup> was used. It is designed to transform RDF data into Java beans and is therefore lightweight in comparison to the previous mentioned ones. It provides annotations to assign RDF types to classes and properties in RDF to getter/setter methods in the corresponding class.

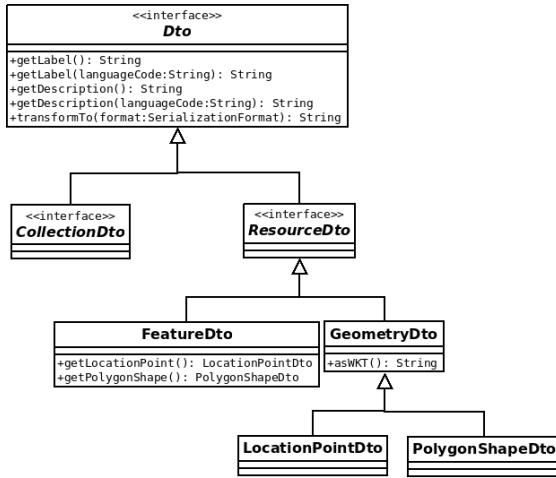


Figure 4.14: UML class diagram of a portion of the implemented Dtos

For this thesis the mapping of RDF data into Java objects causes no major problems, but there is a impedance mismatch between these both worlds that must be considered. Java and OWL/RDF are conceptionally different; Java focuses on classes, whereas OWL/RDF focuses on properties. An issue could be that Java only supports simple inheritance of classes, but multiple inheritance of interfaces. A resource in RDF could also have two types (classes) and no class is the subclass of the other, which cannot be resolved; a room could be for instance a seminar and a learning room. Due to these mismatches and the fact that Dtos shall not only be objects for the presentation layer, but also be transformed into a certain RDF format (see LD principles), each Dto contains the original RDF model, which was the knowledge base for the mapping.

### Presentation layer

The presentation layer is the uppermost layer of the proposed software system and is responsible for presenting the results of the provided services in a human-readable as well as in a machine-readable form (in multiple formats) over the Web. “*The primary means of publishing LD on the Web is by making IRIs dereferenceable, thereby enabling the follow-your-nose style of data discovery. This should be considered the minimal requirements for Linked Data publishing*”[6]. This is also one of the criteria for the submission to the LOD cloud and is evaluated in section 5.1.3. Beside the assurance that all IRIs identifying resources are dereferenceable, there are other ways to publish LD. One possibility is the provision of a ***SPARQL endpoint***, which is a Web service that

---

<sup>22</sup><https://github.com/stardog-union/pinto>

can process SPARQL requests and returns the corresponding results. The advantage of ***SPARQL endpoints*** is that the end user can flexibly request a particular subset of the data stored in the local triple store. In this thesis no ***SPARQL endpoint*** will be accessible on the designed Web application, but the used triple store GraphDB offers one. Another approach are ***RESTful APIs*** that potentially go beyond making resources dereferencable. “***RESTful APIs*** are a programming interface implemented using *HTTP* and the principles of *REST* (*Representational State Transfer*) to allow actions on Web resources”[17]. This is one of the approaches implemented by the proposed system (see next section). Another easy way to publish LD, which is also implemented, is a simple ***RDF dump*** of the whole stored data.

The previous mentioned approaches are mainly focusing on machines and are not necessarily convenient for the average human reader, but ***RESTful APIs*** can for instance be adopted so that results will be returned in a human-readable representation like (X)HTML, if it is demanded by the client (e.g. by a browser) — content negotiations. Due to the ability of nowadays search engines to extract knowledge provided by RDFa, microdata or JSON-LD on regular Web pages, could it be useful to not only provide a human-friendly Web page, but to also include the presented data in a format that is understood by those crawlers into the rendered (X)HTML page. As mentioned in section 3.1 in context of schema.org do search engines use this gathered LD to enhance the search experience, but it only pays off, if the provided LD is understood.

The role model for the designed Web application was DBPedia<sup>23</sup>, which implements all of the mentioned approaches. All the resources are referenceable and provide different formats on demand from Web pages to RDF data in multiple formats. The human-friendly Web pages describing a resource also embeds machine-readable data using the format RDFa. DBPedia also offers an ***SPARQL endpoint*** and ***RDF dumps***.

**To summarize** the Web application that constitutes the presentation layer of the proposed system implements a ***RESTful API*** that not only returns descriptions of resources in a human- and machine-readable format, but also adds additional services like listing all known buildings or search services that are useful for the map application. The human-friendly Web pages also include the presented data in form of JSON-LD. Figure 4.15 shows the tree of services concerning buildings. The blue node with the light-blue border is the start node, the blue nodes represent path segments except segments that are arguments, which are represented by green nodes. The red nodes are the end nodes. Besides the ***RESTful API*** also a ***RDF dump*** is available.

The Play Framework<sup>24</sup> was chosen to realize this layer. It is a Web application framework that is actually written in Scala, but which also provides a Java wrapper. It follows the architectural pattern MVC (Model View Controller). The controllers act as a proxy for the root service factory of the service layer and the models are the Dtos returned by

---

<sup>23</sup>[http://dbpedia.org/page/Vienna\\_University\\_of\\_Technology](http://dbpedia.org/page/Vienna_University_of_Technology)

<sup>24</sup><https://www.playframework.com/>

#### 4. SOLUTION

---

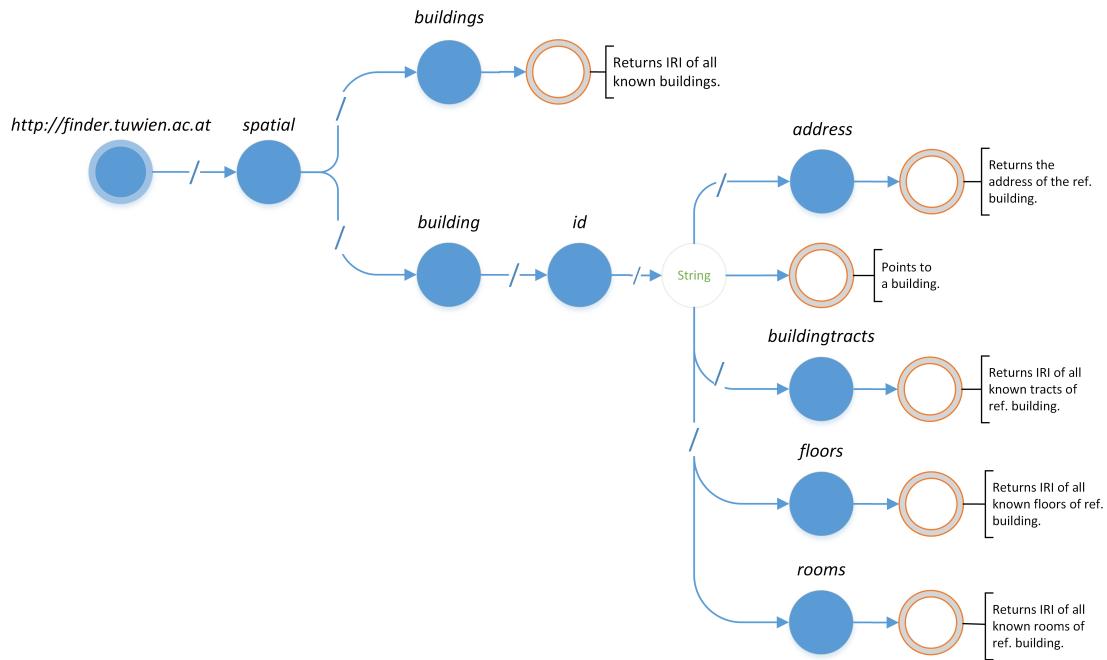


Figure 4.15: IRI design for services concerning buildings

services. These DTOs build the foundation of the rendered view or are simple transformed in one of the supported RDF formats (RDF/XML, Turtle, N-Triples, JSON-LD, TRIG).

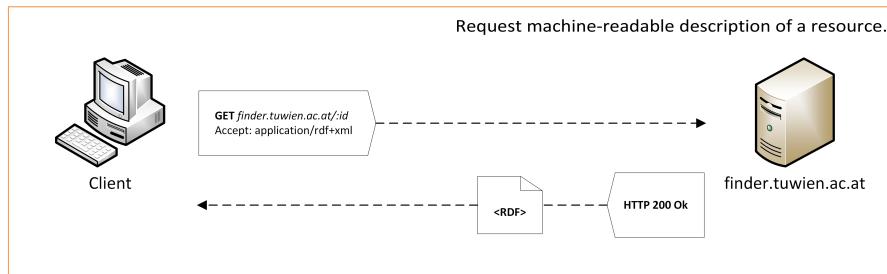


Figure 4.16: Requesting machine-readable representation of a given resource.

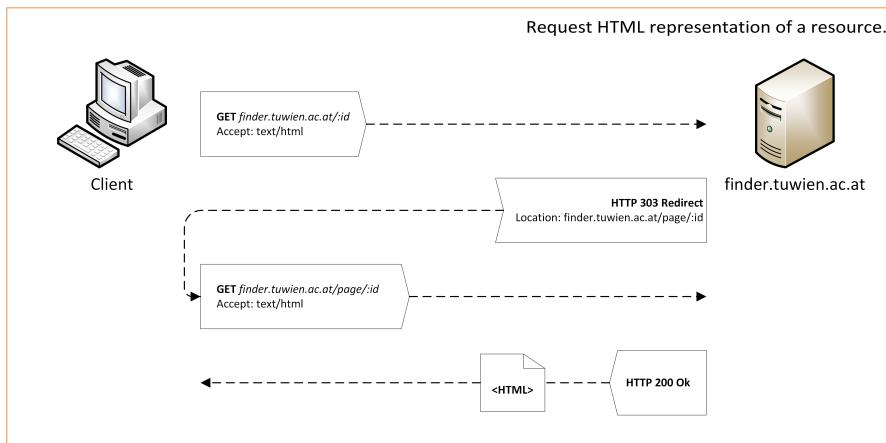


Figure 4.17: Requesting human-readable representation of a given resource.

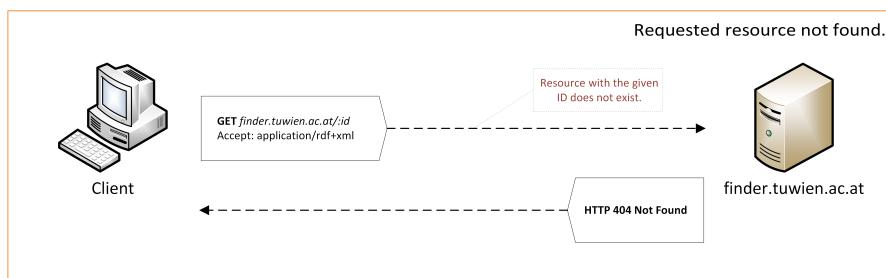


Figure 4.18: Requesting an unknown resource.

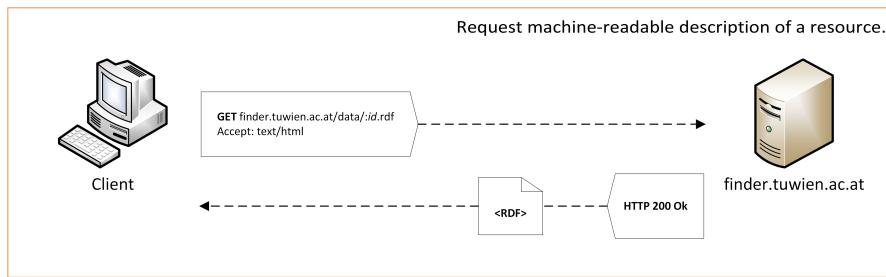


Figure 4.19: Forcing the return of a specific machine-readable representation.

### Content negotiation

## 4.4 Map application

**TODO: A more comprehensive presentation of the user interface for humans to show the targeted use case.**

## 4. SOLUTION

Finder Map Data Catalog Formats ▾ RDF dump

### About: Building of the informatics institute

Entity of type: [tuvs:Building](#) ,

The map displays a building complex with various floors and rooms. The surrounding area includes streets like Favoritenstraße, VAI, and Taubstummengasse. A legend on the left indicates floor levels: EG, 01, 02, 03, 04, 05. Points of interest nearby include Guerilla Bakery, Hotel Johann Strauss, and a copyboxx.

Property	Value
<a href="#">tuvs:buildingCode</a>	"H"
<a href="#">foaf:based_near</a>	<ul style="list-style-type: none"><li>• <a href="http://sws.geonames.org/6506639/">http://sws.geonames.org/6506639/</a></li><li>• <a href="http://linkedgeodata.org/triplify/node/250980183">http://linkedgeodata.org/triplify/node/250980183</a></li><li>• <a href="http://linkedgeodata.org/triplify/node/249718900">http://linkedgeodata.org/triplify/node/249718900</a></li><li>• <a href="http://linkedgeodata.org/triplify/node/1942986502">http://linkedgeodata.org/triplify/node/1942986502</a></li><li>• <a href="http://linkedgeodata.org/triplify/node/249711653">http://linkedgeodata.org/triplify/node/249711653</a></li><li>• <a href="http://linkedgeodata.org/triplify/node/2973957026">http://linkedgeodata.org/triplify/node/2973957026</a></li><li>• <a href="http://linkedgeodata.org/triplify/node/297478688">http://linkedgeodata.org/triplify/node/297478688</a></li><li>• <a href="http://linkedgeodata.org/triplify/node/2984579855">http://linkedgeodata.org/triplify/node/2984579855</a></li><li>• <a href="http://linkedgeodata.org/triplify/node/625303168">http://linkedgeodata.org/triplify/node/625303168</a></li><li>• <a href="http://linkedgeodata.org/triplify/node/625303176">http://linkedgeodata.org/triplify/node/625303176</a></li><li>• <a href="http://linkedgeodata.org/triplify/way/49249315">http://linkedgeodata.org/triplify/way/49249315</a></li><li>• <a href="http://linkedgeodata.org/triplify/way/159915669">http://linkedgeodata.org/triplify/way/159915669</a></li><li>• <a href="http://linkedgeodata.org/triplify/way/182413123">http://linkedgeodata.org/triplify/way/182413123</a></li></ul>

Figure 4.20: Forcing the return of a specific machine-readable representation.

# CHAPTER 5

## Discussion

This chapter is dealing with the evaluation of the proposed solution (see section 5.1) considering the quality measurements outlined in the introduction (see chapter 1). Section 5.2 discusses short comings of the proposed solution and potential enhancements. Conclusion are drawn in the final section 5.3.

### 5.1 Evaluation

In chapter 4 a solution to the given problem description was presented. The goal of this thesis was to satisfy the problem description as well as the quality criteria for the submission to the LOD cloud. As a reminder, those are as follows[11]:

1. There must be resolvable `http://` (or `https://`) URIs.
2. They must resolve, with or without content negotiation, to RDF data in one of the popular RDF formats (RDFa, RDF/XML, Turtle, N-Triples).
3. The dataset must contain at least 1000 triples.
4. The dataset must be connected via RDF links to a dataset that is already in the diagram. This means, either your dataset must use URIs from the other dataset, or vice versa. We arbitrarily require at least 50 links.
5. Access of the entire dataset must be possible via RDF crawling, via an RDF dump, or via a SPARQL endpoint.

The dataset itself is evaluated in section 5.1.1 — point 3. The evaluation of the interlinking is discussed in section 5.1.2 – point 4. Point 1 to 3 and 5 are concerned with the access of the resources and the whole dataset respectively. This is discussed in section 5.1.3.

### 5.1.1 Dataset

The spatial dataset has in total **16295** triples describing spatial entities of the TU Vienna; there are **3280** unique resources in the dataset. Thus this criteria is fulfilled. Figure 5.1 shows a snippet of this dataset describing the building of the informatics institute. The building itself is represented by the dark red node at the top. The big pink nodes reference to building tracts (eight in total), orange nodes to rooms and the blue nodes to *geometries* (points or polygons).

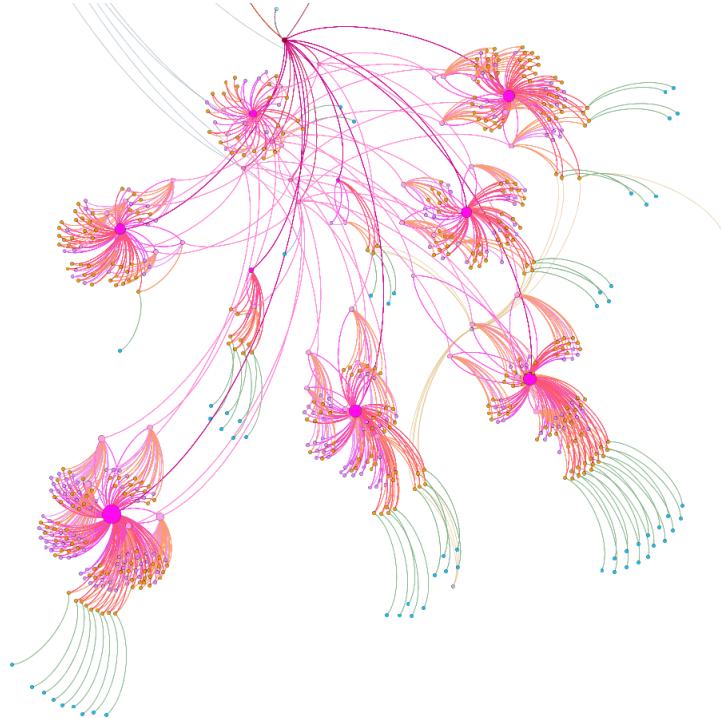


Figure 5.1: Visualization of the spatial data concerning building H (informatics institute).

Overall the majority of the resources describe rooms followed by floor sections and floors. Table 5.1 shows this distribution (for each resource the most specific types were chosen; which have no known subclasses). For this thesis only a little subset of *geometries* and navigation routes were actually gathered and integrated; only for the basement of the informatics building. Hence, these types of resources are under-represented; especially, the navigation routes. If applied to all buildings of the university, points of routes and routes will be leading the chart by a lot, due to the chosen approach for modelling navigation (see section 4.2.2).

### 5.1.2 Linking

Figure 5.2 shows a visualization of `foaf:based_near` links from buildings of TU Vienna that are located in the 4th district of Vienna; the buildings are represented by

Type	Percentage
<a href="http://finder.tuwien.ac.at/vocab/spatial#Room">http://finder.tuwien.ac.at/vocab/spatial#Room</a>	60,7%
<a href="http://finder.tuwien.ac.at/vocab/spatial#FloorSection">http://finder.tuwien.ac.at/vocab/spatial#FloorSection</a>	13,75%
<a href="http://finder.tuwien.ac.at/vocab/spatial#Floor">http://finder.tuwien.ac.at/vocab/spatial#Floor</a>	7,84%
<a href="http://finder.tuwien.ac.at/vocab/navigation#PoR">http://finder.tuwien.ac.at/vocab/navigation#PoR</a>	6,82%
<a href="http://www.opengis.net/ont/sf#Polygon">http://www.opengis.net/ont/sf#Polygon</a>	3,1%
<a href="http://finder.tuwien.ac.at/vocab/spatial#BuildingTract">http://finder.tuwien.ac.at/vocab/spatial#BuildingTract</a>	2,51%

Table 5.1: Common types of the spatial dataset.

Type	Percentage
<a href="http://linkedgeodata.org/ontology/BicycleParking">http://linkedgeodata.org/ontology/BicycleParking</a>	22,65%
<a href="http://linkedgeodata.org/ontology/Restaurant">http://linkedgeodata.org/ontology/Restaurant</a>	14,92%
<a href="http://linkedgeodata.org/ontology/Bench">http://linkedgeodata.org/ontology/Bench</a>	8,28%
<a href="http://linkedgeodata.org/ontology/Cafe">http://linkedgeodata.org/ontology/Cafe</a>	5,52%
<a href="http://www.geonames.org/ontology#Feature">http://www.geonames.org/ontology#Feature</a>	3,57%

Table 5.2: Common types of spatial *features* to which the buildings link.

yellow circles and each outgoing edge is a `foaf:based_near` link. The green circles describe local amenities from markets, caf  s, restaurants to fast food. The dark red circles describes entertainment spots from bars, pubs to cinemas. Orange circles represent *features* of GeoNames that are mostly hotels. Parking lots for bicycles are represented by the blue circles. Grey circles reference to other spots like ATMs, benches and car sharing that do not fit in the previous mentioned categories. As mentioned in section 4.3.3 `foaf:based_near` links are established for spatial *features* that are in distance of maximal 100 meters. This leads to clusters, where buildings that are close to each other share links to the same spatial *features*. The cluster in the upper-right corner consists of the main building at Vienna's Karlsplatz<sup>1</sup> and other close university buildings (max. 200 m). Overall **647** `foaf:based_near` links have been generated, **624** to LinkedGeoData and **23** to GeoNames. Table 5.2 shows the top 10 types of spatial *features* to which the buildings link. This fulfills point 4 of the submission criteria for the LOD cloud.

### 5.1.3 Access

**TODO: Access**

## 5.2 Further work

**TODO: Enter your text here.**

<sup>1</sup><http://www.geonames.org/6944029/technische-universitaet-wien.html>

## 5. DISCUSSION

### **5.3 Conclusion**

**TODO: Enter your text here.**

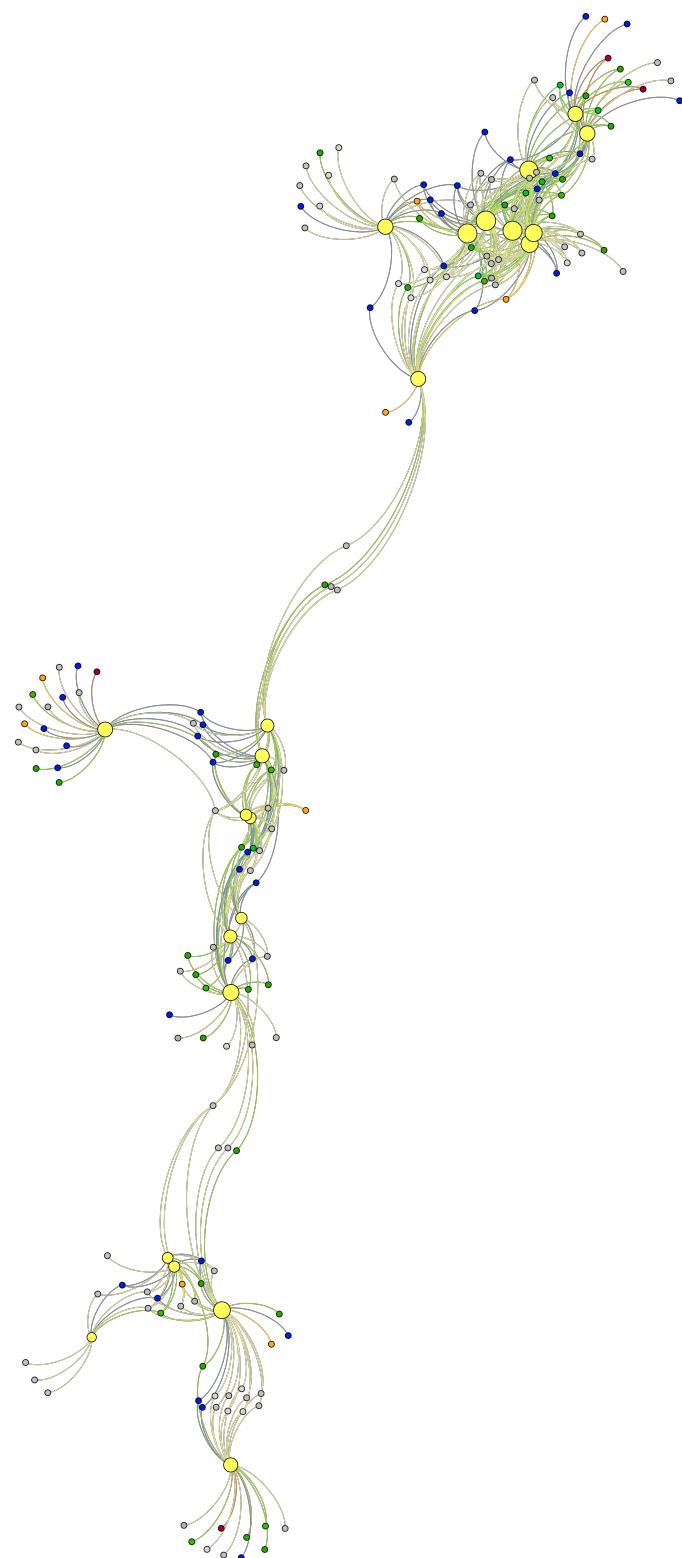


Figure 5.2: Visualization of `foaf:based_near` links.



# List of Figures

1.1	Illustration of the Web of Documents . . . . .	1
1.2	Illustration of the Web of Data . . . . .	2
3.1	Visualization of GEO . . . . .	8
3.2	Visualization of NGEO . . . . .	10
3.3	Visualization of GeoSPARQL . . . . .	11
3.4	Visualization of LOCN . . . . .	13
3.5	Map application of University of Southampton . . . . .	14
4.1	Visualization of the data sources . . . . .	18
4.2	Google Maps: Indoor plan of TU Vienna's main library . . . . .	19
4.3	Room shapes file of the basement floor of the informatics institute's building	20
4.4	Visualization of the navigation ontology . . . . .	22
4.5	Overview of the system architecture . . . . .	24
4.6	UML class diagram of the mediator . . . . .	29
4.7	UML class diagram of the data acquire . . . . .	30
4.8	UML class diagram of the data integrator . . . . .	31
4.9	Heuristic for detecting equivalent buildings (GeoNames) . . . . .	34
4.10	GeoNames features that are nearby (max. 100m) . . . . .	35
4.11	Heuristic for detecting equivalent buildings (LinkedGeoData) . . . . .	37
4.12	LinkedGeoData features that are nearby (max. 100m) . . . . .	38
4.13	UML class diagram of the service layer . . . . .	38
4.14	UML class diagram of a portion of the implemented Dtos . . . . .	40
4.15	IRI design for services concerning buildings . . . . .	42
4.16	Requesting machine-readable representation of a given resource . . . . .	42
4.17	Requesting human-readable representation of a given resource . . . . .	43
4.18	Requesting an unknown resource . . . . .	43
4.19	Forcing the return of a specific machine-readable representation . . . . .	43
4.20	Forcing the return of a specific machine-readable representation . . . . .	44
5.1	Visualization of the spatial data concerning building H (informatics institute) .	46

5.2	Visualization of foaf:based_near links.	49
-----	---	----

## List of Tables

4.1	Named graphs used in the proposed software system	27
4.2	Naming scheme for spatial resources	29
5.1	Common types of the spatial dataset.	47
5.2	Common types of spatial <i>features</i> to which the buildings link.	47

# List of Algorithms



# Acronyms

**CSV** Comma-separated values. 18, 20

**GEO** Basic Geo (WGS84 long/lat) Vocabulary. 7–9, 11, 12, 27

**GML** Geography Markup Language. 10, 12

**IRI** Internationalized Resource Identifier. 20, 24

**LD** Linked Data. 2–4, 12, 14, 17, 18, 20

**LOCN** ISA Programme Location Core Vocabulary. 12, 13, 27

**LOD** Linked Open Data. 3, 7, 12, 13, 15, 17

**LOV** Linked Open Vocabulary. 7, 14

**NGEO** NeoGeo Geometry Ontology. 9, 10, 27

**PDF** Portable Document Format. 17, 18

**RDF** Resource description framework. 5, 9, 10, 12, 18, 23, 31

**RDFa** RDF in Attributes. 12

**RIF** Rule Interchange Format. 10

**ROOMS** Buildings and Rooms Vocabulary. 15

**SPARQL** SPARQL protocol and RDF query language. 5, 12, 22

**SPATIAL** NeoGeo Spatial Ontology. 9

**TU Vienna** Vienna University of Technology. 2–4, 17, 19, 20, 27

**WKT** Well-known text. 9, 10, 12, 20

**XML** Extensible Markup Language. 13, 20



# Bibliography

- [1] Christos Anagnostopoulos, Vassileios Tsetsos, Panayotis Kikiras, and others. On-toNav: A semantic indoor navigation system. In *1st Workshop on Semantics in Mobile Environments (SME05)*, Ayia. Citeseer, 2005.
- [2] Maurizio Atzori. Computing recursive SPARQL queries. In *Semantic Computing (ICSC), 2014 IEEE International Conference on*, pages 258–259. IEEE, 2014.
- [3] Robert Battle and Dave Kolas. Geosparql: enabling a geospatial semantic web. *Semantic Web Journal*, 3(4):355–370, 2011.
- [4] Tim Berners-Lee. Linked data-design issues (2006). URL <http://www.w3.org/DesignIssues/LinkedData.html>, 2009.
- [5] Diego Berraeta, Jon Phipps, Alistair Miles, Thomas Baker, and Ralph Swick. Best practice recipes for publishing RDF vocabularies. *Working draft, W3C*, 2008.
- [6] Christian Bizer. The emerging web of linked data. *IEEE intelligent systems*, 24(5):87–92, 2009.
- [7] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 2009.
- [8] D Brickley. Basic geo vocabulary. *W3C Semantic Web Interest Group*. Available at: <http://www.w3.org/2003/01/geo>, 2003. Accessed: 2016-09-26.
- [9] Dan Brickley and Libby Miller. FOAF vocabulary specification 0.98. *Namespace document*, 9, 2012.
- [10] Dan Connolly and others. Gleaning resource descriptions from dialects of languages (GRDDL). *W3C Recommendation*, 11, 2007.
- [11] Richard Cyganiak and Anja Jentzsch. Linking open data cloud diagram. *LOD Community* (<http://lod-cloud.net/>), 12, 2011.
- [12] Patrick M Dudas, Mahsa Ghafourian, and Hassan A Karimi. ONALIN: Ontology and algorithm for indoor routing. In *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, pages 720–725. IEEE, 2009.

- [13] Ramanathan Guha. Introducing schema.org: Search engines come together for a richer web. *Google Official Blog*, 2011.
- [14] Kevin Haas, Peter Mika, Paul Tarjan, and Roi Blanco. Enhanced results for web search. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 725–734. ACM, 2011.
- [15] Steve Harris, Andy Seaborne, and Eric Prud’hommeaux. SPARQL 1.1 query language. *W3C Recommendation*, 21, 2013.
- [16] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space (1st edition)*, volume Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, 2011.
- [17] Bernadette Hyland, Ghislain Atemezing, Michael Pendleton, and Biplav Srivastava. Linked Data Glossary.
- [18] Bernadette Hyland, Ghislain Atemezing, and Boris Villazón-Terrazas. Best practices for publishing linked data. *W3C Working Group Note*, 2014.
- [19] Carsten Keßler and Tomi Kauppinen. Linked Open Data University of Münster—Infrastructure and Applications. In *Extended Semantic Web Conference*, pages 447–451. Springer, 2012.
- [20] Nemanja Kostic and Simon Scheider. Automated generation of indoor accessibility information for mobility-impaired individuals. In *AGILE 2015*, pages 235–252. Springer, 2015.
- [21] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. VOWL 2: user-oriented visualization of ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 266–281. Springer, 2014.
- [22] Fadi Maali, John Erickson, and Phil Archer. Data catalog vocabulary (DCAT). *W3C Recommendation*, 2014.
- [23] Barry Norton, Luis M. Vilches, Alexander De León, John Goodwin, Claus Stadler, Suchith Anand, and Dominic Harries. NeoGeo Vocabulary Specification - Madrid Edition. 2012.
- [24] Andrea Perego and Michael Lutz. ISA Programme Location Core Vocabulary. 2015.
- [25] Matthew Perry and John Herring. OGC GeoSPARQL-A geographic query language for RDF data. *OGC Implementation Standard. Sept*, 2012.
- [26] Research Group: AKWS. LODStats - Basic Statistics of the LOD cloud. Accessed: 2016-09-26.
- [27] Juan L Reutter, Adrián Soto, and Domagoj Vrgoč. Recursion in SPARQL. In *International Semantic Web Conference*, pages 19–35. Springer, 2015.

- [28] E. Simperl, B. Norton, M. Maleshkova, J. Domingue, A. Mikroyannidis, P. Mulholland, and R. Power. *Using Linked Data Effectively*. The Open University, 2013.
- [29] Barnabás Szász, Rita Fleiner, and András Micsik. iLOC–Building In-door Navigation Services using Linked Data. 2010.
- [30] Jeremy Tandy, Payam Barnaghi, and Linda van den Brink. Spatial Data on the Web Best Practices, January 2016. Accessed: 2016-09-26.
- [31] The Apache Software Foundation. Jena architecture overview. Accessed: 2016-11-12.
- [32] The Apache Software Foundation. Jena Ontology API. Accessed: 2016-11-12.
- [33] The Apache Software Foundation. Spatial searches with SPARQL. Accessed: 2016-11-13.
- [34] Universität Leipzig: Agile Knowledge Engineering and Semantic Web. linkedgeodata.org : About. Accessed: 2016-11-22.
- [35] Csaba Veres and Eivind Elseth. Schema.org for the Semantic Web with MaDaME. In *I-SEMANTICS (Posters & Demos)*, pages 11–15. Citeseer, 2013.
- [36] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and maintaining links on the web of data. In *International Semantic Web Conference*, pages 650–665. Springer, 2009.
- [37] W3C. Data on the Web Best Practices, August 2016. Accessed at: 2016-11-16.
- [38] Fouad Zablith, Mathieu d’Aquin, Stuart Brown, and Liam Green-Hughes. Consuming linked data within a large educational organization. In *Proceedings of the Second International Conference on Consuming Linked Data-Volume 782*, pages 85–96. CEUR-WS. org, 2011.