UNIVERSITÄT LEIPZIG
Faculty of Mathematics and Computer Science
Department of Computer Science
Institute of Business Information Systems

Community-Driven Engineering of the DBpedia Infobox Ontology and DBpedia Live Extraction

Diploma Thesis

Leipzig, May 2010

Thesis Supervisors:		Submitted by	Claus Stadler
Prof. Dr. Inf. habil. K. Fähnrich		Born	12.05.1984
Dipl. Inf. Sebastian Hellmann		Course of Studies	Computer Science

Abstract

The DBpedia project aims at extracting information based on semi-structured data present in Wikipedia articles, interlinking it with other knowledge bases, and publishing this information as RDF freely on the Web. So far, the DBpedia project has succeeded in creating one of the largest knowledge bases on the Data Web, which is used in many applications and research prototypes. However, the manual effort required to produce and publish a new version of the dataset – which was already partially outdated the moment it was released – has been a drawback. Additionally, the maintenance of the DBpedia Ontology, an ontology serving as a structural backbone for the extracted data, made the release cycles even more heavyweight. In the course of this thesis, we make two contributions: Firstly, we develop a wiki-based solution for maintaining the DBpedia Ontology. By allowing anyone to edit, we aim to distribute the maintenance work among the DBpedia community. Secondly, we extend DBpedia with a Live Extraction Framework, which is capable of extracting RDF data from articles that have recently been edited on the English Wikipedia. By making this RDF data automatically public in near realtime, namely via SPARQL and Linked Data, we overcome many of the drawbacks of the former release cycles.

Contents

1 Introduction

Wikipedia has become the most popular online encyclopedia and ranks among the top ten visited sites[1]. Most of its contained knowledge is represented as free text and is therefore only of limited use to machines. However plenty of information is also provided in semi-structured and even structured form as for example references to infobox templates and links. The DBpedia project aims at extracting this (semi) structured information and publishing it freely on the Web. So far, there are two teams collaboratively working on DBpedia: One team from the Freie Universität (FU) Berlin, and one from the Universität Leipzig. The DBpedia Ontology is a multi-domain ontology that serves as a structural backbone for this data. This ontology was manually created and has been maintained by the DBpedia team from the FU Berlin.

1.1 Motivation

So far the generation of the DBpedia datasets was based on database dumps from Wikipedia. Each release required manual efforts of downloading the dumps, loading them into a database, configuring and starting the extraction process, and finally publishing the resulting data by making it accessible via file downloads, the Linked Data Interface, and the SPARQL endpoint.

This accounted for a rather heavyweight release cycle and releases were only done every three to six months. Also, due to the fast moving nature of Wikipedia, at the time these datasets were released, they were already partially outdated. The DBpedia Ontology and the mapping rules (rules that relate infoboxes to that ontology) also suffer from actuality problems. But in addition it turned out that maintaining the ontology and the mapping rules is very hard for a small team. After all, these things need to be kept in sync with the actual data on Wikipedia.

1.2 Goals

The goals of this thesis are twofold: The maintenance of the DBpedia Ontology and the mapping rules should be crowd-sourced in order to distribute the burden. For that reason

[1] rank 6 according to Alexa.com (retrieved 17-Feb-2010)

a wiki-based solution that allows modeling both of them is developed in the course of this thesis. The other goal is to extend the original DBpedia Extraction Framework to make it capable of processing the following things in real-time: edits of Wikipedia articles, changes to the DBpedia Ontology, and changes to the mapping rules. This means that a publicly accessible triple store should always contain two things: (1) All RDF data corresponding to Wikipedia articles' latest revisions and (2) the RDF data reflecting the most recent state of the DBpedia ontology.

1.3 Structure of this Thesis

In the first chapter we give an introduction to basic Semantic Web technologies. The following chapter gives an overview over the English Wikipedia and describes the original state of the DBpedia Extraction Framework (i.e. the state when this thesis was started). Chapters four and five explain the main contributions of this thesis: Chapter four describes a wiki-based solution for engineering the DBpedia Ontology and relating Wikipedia infoboxes to it. Chapter five explains how the original DBpedia Extraction Framework has been improved: It was made capable of processing articles from the English Wikipedia, as well as the Ontology definitions introduced in section four, in realtime. Chapter six describes some of the related work. Finally, chapter seven concludes this thesis.

1.4 Conventions

For making reading easier, the following conventions are used:

- The English Wikipedia will be referred to as EnWiki.
- Many URIs used throughout this thesis are abbriviated using the namespace prefixes in Table 1.

Prefix	URI
ex	http://example.org/
dbpedia	http://dbpedia.org/resource/
dbpedia-owl	http://dbpedia.org/ontology/
dbpprop	http://dbpedia.org/property
dbpmeta	http://dbpedia.org/meta/
foaf	http://xmlns.com/foaf/0.1/
opencyc	http://sw.opencyc.org/concept/
owl	http://www.w3.org/2002/07/owl#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
skos	http://www.w3.org/2004/02/skos/core#
umbel-sc	http://umbel.org/umbel/sc/
xsd	http://www.w3.org/2001/XMLSchema#
yago	http://www.mpii.de/yago/resource/

Table 1: Namespace prefixes used throughout this thesis

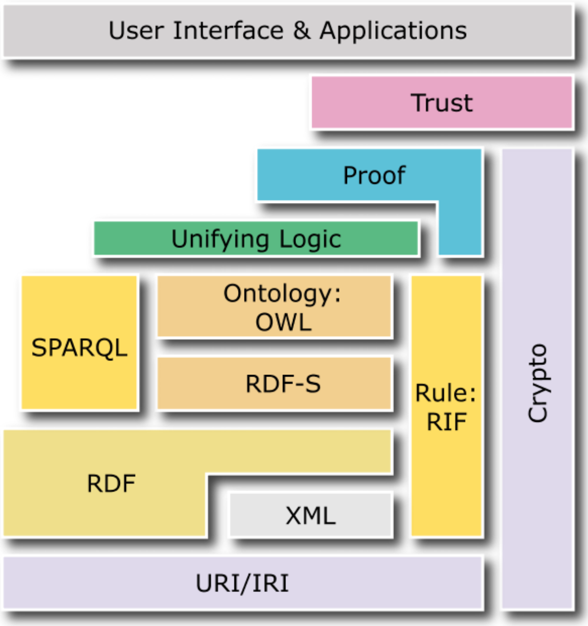
2 Preliminaries

2.1 The Semantic Web

The vision of the Semantic Web, first described in [6], is about computer agents being able to understand the "*meaning*" of some of the content on the Web. By that we do not mean that agents magically become as intelligent as humans, but rather that content is published in a form that enables machines to easily gather information about things, and eventually allows them to apply formal logic reasoning to that content. Consider for example the query of finding inexpensive hotels in the vicinity of a given location which offer certain services such as cable TV and a swimming pool. Traditional agents would have a hard time determining the answer to such task. They would have to rely on data scraping or would require specialized code in order to deal with the many non-standard APIs sites nowadays provide. Once they manage to obtain the necessary data for answering the query, there is still the problem of how to account for information that may become available in the future, such as what TV channels are actually offered. But there is even another problem: Agents supposedly solving the same problem, given the exact same query, and the exact same data to operate on, may come to different conclusions. This happens due to the data lacking a defined meaning.[2] For example one agent may interpret "hotels" as to include "motels" as well, while another agent may follow different semantics. These are the basic problems addressed by the Semantic Web. One of its fundamental properties is the implication of a *Web of Data* - a web made up of links between individual pieces of data as opposed to the traditional links between documents[25]. The main reason is that data is a prerequisite for reasoning. Traditional (e.g. HTML and XML) documents with their sheer endless amounts of different schemas are unsuitable for uniform data access.

In order to realize the Semantic Web, an architecture of technologies has been proposed, which is known as the *Semantic Web Stack*. It is depicted in Figure 1. In the following sections we will discuss the components which were relevant to this thesis.

[2] Or agents not adhering to it. But that is beside the point.



Source: `http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24)`

Figure 1: Components of the *Semantic Web Stack*

2.2 Uniform Resource Identifiers

A Uniform Resource Identifier (URI) is a “compact sequence of characters that identifies an abstract or physical resource”[5]. It is important to understand that the sole purpose of URIs is to identify resources, but not to interact with them. As such they can be used to identify *anything* - so any concept one can think of, such as websites, books, people or numbers. The dereferencable subset of URIs - that is the set of URIs identifying machine accessible resources such as websites - is known as Uniform Resource Locators (URLs). Identifying things with URLs has the advantage that more information about the thing may be retrieved upon dereferencing them. As a side note, a URL may both identify a thing and point to a description of that thing, which makes its meaning ambiguous. For example the URL `http://en.wikipedia.org/wiki/European_Union` could refer to both the “European Union” as an organization and the article on Wikipedia. Solutions to disambiguate the meanings are given in [17]. For convenience URIs are often abbreviated using *namespace prefixes*. In regard to the example above, if we somewhere stated that `enwiki` is an abbreviation of `http://en.wikipedia.org/wiki/` we could have written `enwiki:European_Union` instead.

2.3 The Resource Description Framework

The Resource Description Framework (RDF) is used to represent information on the Web[13]. The framework defines two things: A *data model* for representing arbitrary statements about resources, and a basic *vocabulary* which can be used to give statements a defined meaning. Since RDF is intended for the Web, it is natural that things are identified with URIs. Furthermore RDF supports anonymous resources and literals such as strings and integers. A *statement* consists of three parts namely *subject*, *predicate* and *object* where the set of values they can take is shown below:

Part	URI	Anonymous	Literal
Subject	✓	✓	
Predicate	✓		
Object	✓	✓	✓

Table 2: Valid values within an RDF statement

Since the object of one triple may appear as the subject in another triple it is easy to see that a triple can be seen as an edge in a directed labeled graph. Therefore a set of triples is also called a *graph*. Anonymous resources are usually referred to as *blank nodes*.

It is worth pointing out that literals may be either plain or typed. A plain literal is a combination of a string with an optional language tag. A typed literal is a combination of string and a URI denoting its data type. This data type constrains the set of values that may be assigned to the string.

A couple of RDF statements is shown in Listing 1: In this example this resource `ex:London` shall denote the city of London. The first line states that its label in English is “London”. The second line states that the population is an integer of value 8278251.

```
ex:London ex:label "London"@en .
ex:London ex:population 8278251^^xsd:integer
```

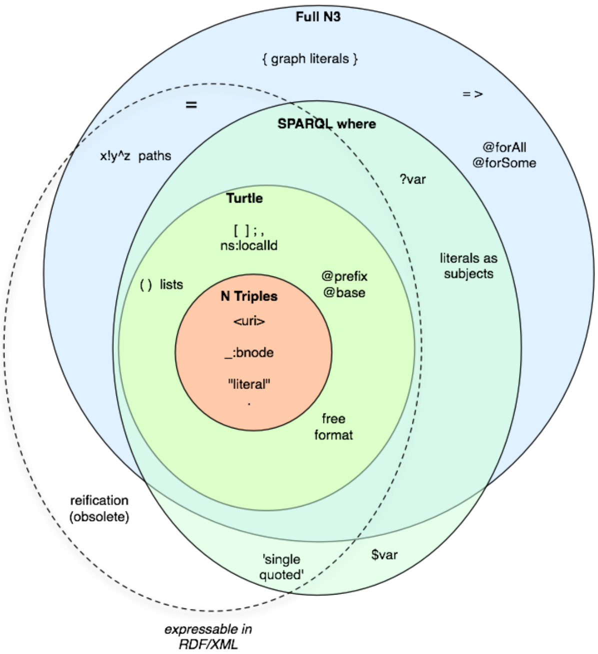
Listing 1: Examples of RDF statements

2.3.1 Syntaxes

The RDF data model itself is an abstract syntax which means that it is independent of any particular representation or encoding. Several concrete syntaxes exist which vary in readability, expressiveness and tool support.

RDF/XML is an XML based syntax and therefore has the same advantages and disadvantages as basically any other XML based syntax: On the one hand there exists a wide range of tools which make parsing and processing of such data relatively easy for machines. On the other hand XML documents contain a lot of syntactic noise which makes human reading and writing of actual data harder than it ought to be. However, RDF/XML is a mandatory exchange syntax[8] for some RDF-based languages, such as OWL 2[3]. This means that the syntax must be supported by tools dealing with that language.

N-Triples, Turtle and Notation3 are plain text based concrete syntaxes which are more human-friendly as the XML variant. N-Triples is a subset of Turtle and Turtle is a subset of Notation3 which means that they are not completely different syntaxes but merely syntaxes with varying degrees of expressivity, as shown in Figure 2.



Source: `http://www.w3.org/DesignIssues/Notation3`

Figure 2: Set relations between the various syntaxes

Besides the syntaxes listed here, others came into existence like TriX, TriplesML, or

[3] Discussed in 2.6.1

Regular XML RDF (RXR). However none of them seems to play an important role.

2.4 SPARQL and SPARUL

SPARQL[16] is a recursive acronym for “SPARQL Protocol and RDF Query Language”. It is the query language for the Semantic Web and is therefore a language adapted to the RDF data model. Since RDF data forms a graph, the central part of the SPARQL query language are graph patterns. These graph patterns look similar to graphs written in Turtle with the exception that variables may appear anywhere as a subject, predicate, or object. Another important part of a SPARQL query is the *query form* which controls the result being returned based on the data matched by the graph pattern of the query. Four query forms are given, namely SELECT, CONSTRUCT, DESCRIBE and ASK. The first two return the result as a table and a graph, respectively. DESCRIBE is used to fetch data about individual resources, and ASK serves to determine whether a query yields any result at all.

SPARUL[18] is the corresponding data manipulation language also known as *SPARQL/Update*. It introduces the INSERT, DELETE and the more general MODIFY statement. The latter is a combination of the first two. In essence it works like a SPARQL CONSTRUCT query that inserts and/or removes the result set from the store. Table 3 shows some example queries.

Select	Construct	Insert

`SELECT ?name`
`FROM <http://ex.org> {`
`  ?p ex:type ex:Person .`
`  ?p ex:name ?name .`
`}`
	`CONSTRUCT {`
	`  ?s rdfs:label ?o .`
	`} FROM <http://ex.org> {`
	`  ?s ex:name ?o .`
	`}`
		`INSERT INTO <http://ex.org> {`
		`  ex:Anne ex:knows ex:Bob .`
		`}`

Table 3: Examples of simple SPARQL queries

2.5 Triple Stores and Query Engines

Triple stores are database systems capable of storing and retrieving RDF data. Some implementations are based on relational database technology which means that they internally rewrite SPARQL and SPARUL queries to SQL statements. There also exist native implementations of triple stores such as *Sesame*[4], *Virtuoso*[5] and *AllegroGraph*[6]. Engines like *Triplify*[7] and *D2R*[8] allow the definition of mappings from legacy relational data to RDF. The latter engine even supports querying with SPARQL. The Berlin Benchmark[7] compared the performance of native and rewriting-based triple stores. At that time rewriters outperformed native stores with increasing dataset size.

2.6 Ontologies and Ontology Languages

"An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon. Explicit means that the types of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine- readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group."[21] Ontologies are represented using *ontology languages*. Many of such languages exist which vary greatly in expressivity, computational complexity and decidability. Some of them are presented in the following section.

2.6.1 RDFS, OWL and OWL 2

RDF schema (RDFS), the Web Ontology Language (OWL[9]), and its successor OWL 2 are ontology languages which have gained significant importance in the context of the Semantic Web. OWL and OWL 2 can be broken down into sub languages which trade expressive power for efficiency of reasoning. OWL's sublanguages are OWL Lite, OWL DL and OWL Full. The sublanguages of OWL 2, better known as *profiles*, are OWL 2 EL, OWL 2 QL and OWL 2 RL. A major difference between the two versions of OWL is, that the profiles were designed to be especially suitable for certain types of applications.

These languages have the following basic concepts in common:

[4] `http://www.aduna-software.com/technology/sesame`

[5] `http://virtuoso.openlinksw.com/`

[6] `http://www.franz.com/agraph/allegrograph/`

[7] `http://triplify.org`

[8] `http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/`

[9] This is not a typo: `http://lists.w3.org/Archives/Public/www-webont-wg/2001Dec/0169.html`

- *Individuals* are primitive entities. As such they are no sets which distinguishes them from classes.
- *Classes* are sets of individuals. The members of a class are called *instances*. An individual may belong to multiple classes which are referred to as its *types*.
- *Literals* are data values such as strings or integers.
- *Properties* are relations between individuals, classes and literals.

Although the aforementioned languages are all based on these concepts, their expressivity varies. For example while it is legal in RDFS to model classes of classes, this is considered illegal in OWL DL.

Ontologies form the backbone of the Semantic Web as they enable us to give meanings to resources. For example, assume we are given the following statements:

```
Leipzig a City.
Leipzig locatedIn Saxony.
Saxony locatedIn Germany.
```

If a machine wanted to find all German cities based on these statements, it would have to infer the fact `Leipzig locatedIn Germany`. Although a special treatment of the `locatedIn` property could be hard coded into an application, the point of ontologies is that this treatment can be explicitly stated: In our example we would have to add the triple `locatedIn a owl:TransitiveProperty` in order to define `locatedIn` as a transitive property. As a consequence, any application understanding the ontology language is then able to draw the correct conclusions.

2.6.2 Manchester OWL Syntax (MOS)

This syntax's main goal is to simplify the reading and writing of class expressions[10], especially for people who do not have Description Logic (DL) background[11]. It achieves that goal by hiding the formal symbols typically used in this field behind English keywords and introducing a grammar where these keywords appear in locations that make them look more natural. An example is shown in Figure 3.

[10] Also known as class descriptions.

```
Pizza ⊓ (
¬(∃hasTopping.MeatTopping) ⊔
¬(∃hasTopping.FishTopping))
```

```
Pizza THAT
NOT hasTopping SOME
(MeatTopping OR FishTopping)
```

Figure 3: DL Syntax vs MOS syntax
Definition of the class “Vegetarian Pizza” as a Pizza without Meat and Fish toppings in traditional DL syntax (left) and MOS syntax (right)

2.6.3 Reification and Annotations

Reification, sometimes called “nounification” or “thingification”, refers to making a statement about a statement. If we want to represent the statement “Anne knows Bob claims Charlie” in RDF we first have to introduce a new resource - usually a blank node - which represents the base statement “Anne knows Bob”. Once such a resource exists, it can be used in further statements. RDF provides a reification vocabulary. However it was declared deprecated because of unclear semantics. Eventually *annotations* were introduced in OWL 2. Instead of making statements about statements, the view point has changed to annotating statements. Annotations are logically irrelevant, which means that they do not affect the truth value of the statement being annotated, and are therefore ignored by a reasoner. An example of the RDF-reification and OWL 2 annotation approaches is given in Table 4.

RDF-Reification	OWL 2 Axiom Annotations

`_:b a rdf:Statement`
`_:b rdf:subject :s`
`_:b rdf:predicate :p`
`_:b rdf:object :o`
	`_:b a owl:Axiom`
	`_:b owl:annotatedSource :s`
	`_:b rdf:annotatedProperty :p`
	`_:b rdf:annotatedTarget :o`

Table 4: RDF-Reification vs OWL 2 Axiom Annotations

2.7 Linked Data

The fundamental feature which contributed to the success of the World Wide Web was the hyperlink. In fact without links, there would not be any web at all. However most links are untyped relations between documents (e.g. HTML or XHTML). While this is of no problem for humans, it greatly complicates the tasks for machine agents to explore and find relevant information. On the one hand an agent has to deal with the various schemas of these documents and on the other hand it must somehow decide which links to follow. In contrast, Linked Data is about making typed links between pieces of (RDF) data. This makes it easier for machine agents to follow links and retrieve data that seems interesting to them just like humans. Note that already existing technologies for naming resources (URIs), resolving these names to physical addresses (DNS), and transferring data from them (HTTP) can be reused directly. The following principles for publishing Linked Data are adapted from [4] and are summarized as:

- Name things using HTTP[11] URIs[12]. Even better, use URLs.
- Upon dereferencing such URL, return an RDF dataset describing that resource. Additionally that dataset should...
- ...contain URLs that can be further resolved to more RDF data.

Content negotiation is a feature of HTTP which enables user agents to choose from different representations for a given resource identified by a URI. A browser could for example attempt to request websites in preferred languages or images in preferred formats. Of course the server must support the requested representation. By using content negotiation, it is possible that for the same URI a browser would obtain a human readable HTML representation while another agent would receive an RDF representation.

The Linking Open Data cloud (LOD cloud), depicted in Figure 4, is the result of the *Linking Open Data* W3C Community project[19]: Within this project various open datasets are (converted to and) published as RDF and interlinked with each other. As a consequence, information about entities residing in multiple, partly even heterogeneous datasets can be accessed in a uniform way.

Notably, DBpedia plays the role of a *linking hub* within the LOD cloud. The main reasons that led to this development are:

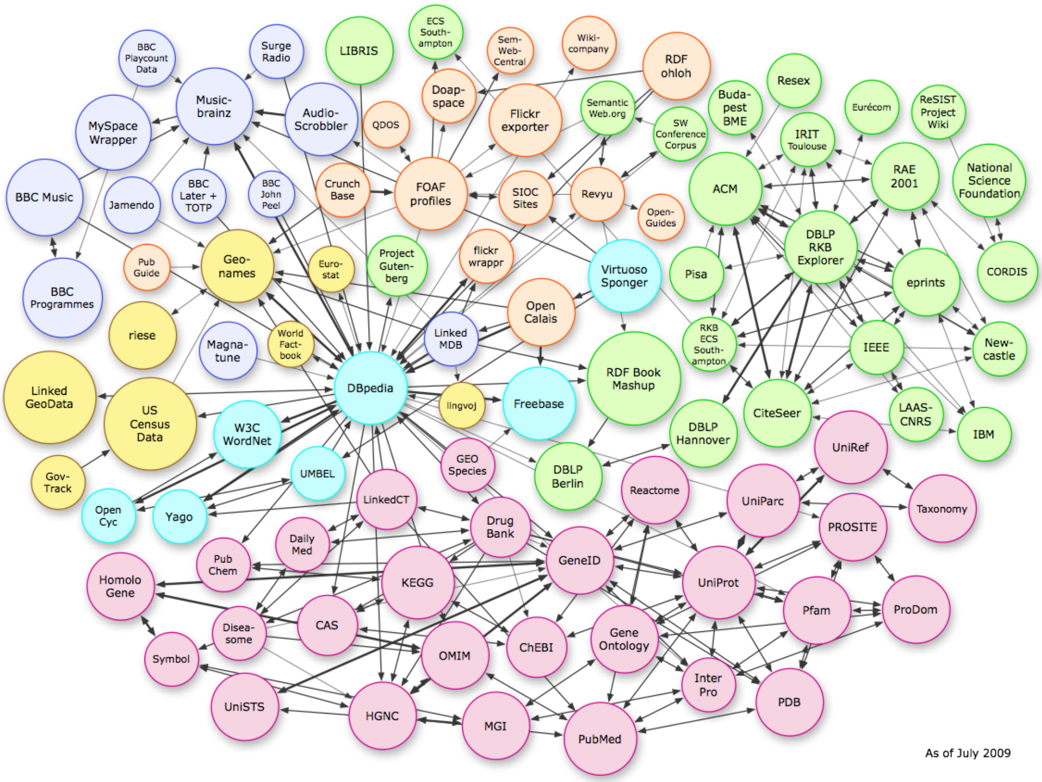
1. The meaning of (most) EnWiki article names does not change (significantly) over time as shown in [10], making them suitable for knowledge representation. The same is valid for DBpedia URIs as they are based on these names.

[11] As HTTP is widely supported.

[12] As any URI may already be or eventually become a URL, making it machine accessible.

2. EnWiki covers multiple domains such as movies, places, organizations. The corresponding identifiers are attractive for reuse in these domains due to their stable meaning. As in the previous point, DBpedia benefits directly from EnWiki.

3. In contrast to Wikipedia, DBpedia URIs can be resolved to RDF via the Linked Data interface. As DBpedia itself contains links to various other knowledge bases, linking to DBpedia indirectly interlinks with them.



As of July 2009

Source: `http://www4.wiwiss.fu-berlin.de/bizer/pub/lod-datasets_2009-07-14_colored.png`

Figure 4: The *Linking Open Data cloud*

3 Status Analysis of DBpedia

The *DBpedia project* aims at extracting information based on semi-structured data present in Wikipedia articles, interlinking it with other knowledge bases, and publishing this information as RDF freely on the Web. In a sense DBpedia can be seen as the “Semantic Web mirror of Wikipedia”: The data generated within the project are RDF representations of a subset of the information contained in Wikipedia articles. This data is then made public in accordance with Semantic Web principles, concretely via Linked Data and SPARQL interfaces. DBpedia’s prime showcase is its capability of answering complex queries Wikipedia cannot answer, as for example the SPARQL query corresponding to “find all soccer players, who played as the goalkeeper for a club that has a stadium with more than 40.000 seats and who are born in a country with more than 10 million inhabitants”[13].

The three major parts of the DBpedia project are the website, the datasets and the Extraction Framework. The *DBpedia datasets* are the results of extraction processes from Wikipedia. The dataset based on EnWiki’s infoboxes is probably the most interesting one. The extraction work is performed by the *DBpedia Extraction Framework*, which can be freely downloaded from source forge[14]. The *DBpedia ontology* has been created for the purpose of classifying this extracted data. In the course of the Linking Open Data community project, DBpedia became interlinked with other knowledge bases. For instance, YAGO, UMBEL and OpenCyc serve as additional classification schemas. More information about these knowledge bases is presented in Section 3.5.

The *DBpedia website* provides access to the datasets via Downloads, SPARQL and the Linked Data interface. Notably the latter two are powered by OpenLink’s *Virtuoso Universal Server*. DBpedia is a community effort in the sense that anyone is invited to share their ideas and criticism on the mailing list.

In this chapter we first give an introduction to Wikipedia, followed by an overview of the structures present in its articles. Then we describe how those structures are translated to RDF by the DBpedia Extraction Framework, which is also explained. Finally we briefly describe some of the datasets DBpedia is interlinked with, and the triple store

[13] `http://wiki.dbpedia.org/OnlineAccess`

[14] `http://sourceforge.net/projects/dbpedia`

hosting these datasets.

3.1 An Introduction to Wikipedia

DBpedia uses Wikipedia as its primary data source. We will first give an overview of Wikipedia in general before moving into the technical details like the structures present in its articles.

3.1.1 The Wiki Methodology

Wikis are a proven technology for enabling massive amounts of users to collaboratively contribute content to a system for gathering knowledge. The breakthrough came with Wikipedia. Despite early concerns that the concept is doomed to end in pure chaos, Wikipedia's community succeeded in the development of mechanisms which prevent that. Ward Cunningham, the founder of the first wiki, assembled a set of general design principles[15] for wikis. These include: easy to use, freely editable, and tolerant to errors. Although these principles largely apply to Wikipedia as well, Wikipedia defines more of them which ensure its status as an encyclopedia. These are known as the *Five Pillars*[16]. Probably the best known of these principles is the *Neutral Point of View (NPOV)*.

The basic concepts of a wiki are relatively simple: Users may view or edit pages. All edits are tracked in a history and can be undone as needed as for example because of vandalism. For each article in Wikipedia there is a talk page which allows discussion prior to making changes to the article.

The social structures of Wikipedia are far more complex than that. Although there exist roles such as bureaucrats and administrators which grant some users more power and privileges than the average user (e.g. deleting articles and blocking users), their actions must conform to certain rules. These rules are not dictated by an individual person but are rather worked out and agreed upon through a community process. In general, conflicts can be resolved on different levels. In the simplest case issues can be resolved on talk pages. More complex cases are dealt with on the administrators' notice board. Editors may also request comments or arbitration for their problem to get

[15] http://c2.com/cgi/wiki?WikiDesignPrinciples

[16] http://en.wikipedia.org/w/index.php?title=Wikipedia:Five_pillars&oldid=363811590

the community's opinion on how to reach consensus. It is important to understand that finding consensus on a matter does not aim for seeking universal agreement, but merely the best solution that can be found at that time.[17]

3.1.2 The Wiki-Software MediaWiki

MediaWiki[18] is Wikipedia's underlying wiki software and was originally developed for the needs of Wikipedia. Because it is free software, anyone can download it and set up their own instance. Nowadays the WikiMedia foundation uses it for many other projects besides Wikipedia, among them Wiktionary[19] and Wikiquote[20].

The software allows the creation of *pages* in which information is written in a lightweight markup language known as *wikitext*. This markup is eventually rendered to HTML for viewing with browsers. As a small clarification: We use the term *article* to refer to encyclopedic articles, as EnWiki defines them[21]. Every article on Wikipedia is technically realized with a MediaWiki page. However not every page corresponds to an article. For instance, talk pages are clearly not articles.

In the remainder of this section we will give an overview over the page structure and explain some of the most relevant markup for DBpedia.

Page structure A *page title* is composed of a *namespace* and a *page name*, which are usually separated by the first colon. However if there is no colon or the part before the first colon is not a namespace known to the wiki, the page will be considered to be in the *main namespace*. MediaWiki defines a set of default namespaces such as *Talk*, *User*, and *Help*. Therefore a page named `Help:Namespace` will be in the *Help* namespace, whereas a page named `Mission:Impossible` will be located in the main namespace (unless *Mission* is declared as a namespace). In general the MediaWiki software allows every page to have *sub pages*, where the sub page name is separated from its parent page by a slash. An example is given in Figure 5.

The case sensitivity rules are as follows: Namespaces are case insensitive, page

[17] `http://en.wikipedia.org/w/index.php?title=Wikipedia:Consensus&oldid=359380073`

[18] `http://www.mediawiki.org`

[19] `http://wiktionary.org`

[20] `http://wikiquote.org`

[21] `http://en.wikipedia.org/w/index.php?title=Wikipedia:What_is_an_article\%3F&oldid=357461931`

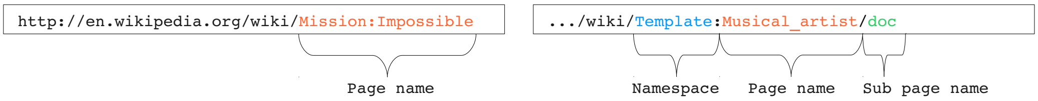
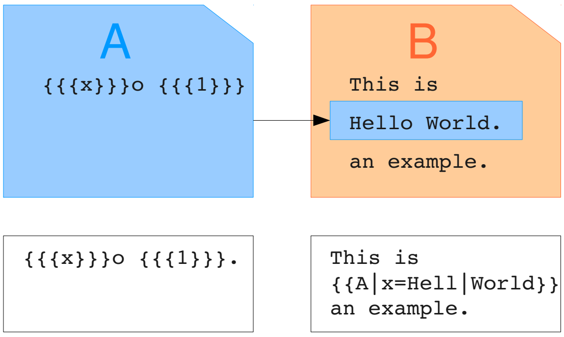

http://en.wikipedia.org/wiki/Mission:Impossible
Page name
.../wiki/Template:Musical_artist/doc
Namespace
Page name
Sub page name

Figure 5: Examples of MediaWiki page names

names are case sensitive except for the first letter, and sub page names are fully case sensitive.

Templates and Transclusion MediaWiki supports *transclusion*, which means that a page can be embedded into another page by reference. Within the reference it is possible to specify *arguments*, consisting of an optional *argument-key* and an *argument value*. This causes every *parameter* in the referenced page to become replaced with the value of the argument whose key matches the name of that parameter. The name of a parameter corresponding to the nth key-less argument is “n”, starting with 1. An example of a reference is given in Figure 6. Pages specifically intended for transclusion are called *Templates*. Templates are often used for defining layouts, text snippets and even calculations common to multiple pages. Naturally templates should reside in the `Template` namespace.



```
{{{x}}}o {{{1}}}.
```

```
This is
{{A|x=Hell|World}}
an example.
```

Figure 6: Example of a MediaWiki transclusion
In this example page *B* transcludes page *A*. The parameter *x* becomes replaced with *Hell*. As the argument with the value *World* is the first one without a key, the value is assigned to the parameter *1*.

Links MediaWiki supports four types of links. The following list is taken from the MediaWiki documentation[22],

- *Internal links* to other pages in the wiki
- *Interwiki links* to other websites registered to the wiki in advance
- *Interlanguage links* to other websites registered as other language versions of the wiki
- *External links* to other websites

As a full description would be beyond the scope of this thesis, we only give a brief overview. In general, the first three link types are syntactically equivalent: They consist of the name of a link target enclosed with double square brackets. The software disambiguates the link types by matching the names of the link target against a set of registered prefixes.

A special type of internal link is the category link: A page linking to another page in the category namespace automatically becomes a member of that category, and therefore appears in the member list of that category.

Absolute URIs within the text are automatically recognized as external links, however it is also possible to enclose them with single square brackets. An alternate piece of text that should be displayed in the rendered page instead of the raw link can only be specified for links that are enclosed with single or double brackets. Figure 7 gives examples for the different link types.

Link type	Example
Internal	`[[Munich]]`
Language	`[[de:München]]`
Category	`[[Category:German state capitals]]`
External	`http://www.muenchen.de/`
	`[http://www.muenchen.de/]`

Figure 7: Examples of MediaWiki links

[22] `http://www.mediawiki.org/w/index.php?title=Help:Links&oldid=324505`

3.1.3 Wikipedia-Specific Structures

While MediaWiki provides essential markup for the creation of articles, the design of Wikipedia articles must also follow conventions in order to present information in a coherent way. The structures interesting to DBpedia are presented below.

Abstracts Wikipedia articles should start with an abstract of the topic they are about.

Disambiguation pages Sometimes article names are discovered to be *homonyms*, which means that they have multiple meanings. For instance *Apple* could refer to the fruit or the company. In such cases the EnWiki community extends these names with context information (e.g. "Apple_Inc.") in order to distinguish the different meanings. Additionally, a page with the name *<homonym>_(disambiguation)* is created which links to all the articles that provide information about the different meanings of the homonym. This page is called a *disambiguation page*.

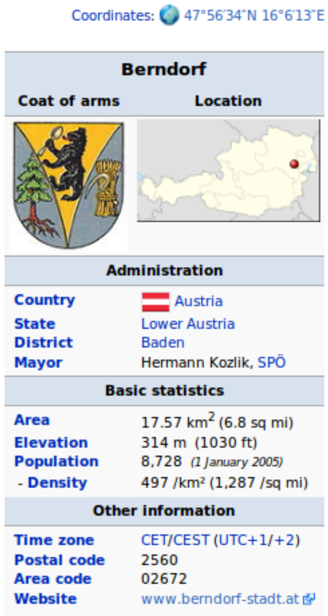
Infoboxes EnWiki defines an infobox as a "fixed-format table designed to be added to the top right-hand corner of articles to consistently present a summary of some unifying aspect that the articles share"[23]. Infoboxes that are implemented using templates are known as *infobox templates*. The obvious advantage of such templates is, that the work required to set up an infobox on a page is reduced to a minimum. These templates form the basis for DBpedia's most specific data about articles. An example of the wikitext of an infobox and its presentation is shown in Figure 8. By convention, the names of infobox templates should start with *Infobox_*.

[23] `http://en.wikipedia.org/w/index.php?title=Help:Infobox&oldid=362463060`

Wikitext

```
{{Infobox Town AT
| name = Berndorf
| image_coa = AUT Berndorf COA.jpg
| state = [[Lower Austria]]
| district = [[Baden (district of Austria)|Baden]]
| population = 8728
| population_as_of = 01.01.2005
| pop_dens = 497
| area = 17.57
| elevation = 314
| lat_deg = 47
| lat_min = 56
| lat_sec = 34
| lat_hem = N
| lon_deg = 16
| lon_min = 6
| lon_sec = 13
| lon_hem = E
| postal_code = 2560
| area_code = 02672
| mayor = Hermann Kozlik, [[SPÖ]]
| website = [http://www.berndorf-stadt.at
            www.berndorf-stadt.at]
}}
```

Rendered



Coordinates: 47°56′34″N 16°6′13″E

Berndorf	
Coat of arms	Location
Administration	
Country	Austria
State	Lower Austria
District	Baden
Mayor	Hermann Kozlik, SPÖ
Basic statistics	
Area	17.57 km^2 (6.8 sq mi)
Elevation	314 m (1030 ft)
Population	8,728 (1 January 2005)
- Density	497 /km² (1,287 /sq mi)
Other information	
Time zone	CET/CEST (UTC+1/+2)
Postal code	2560
Area code	02672
Website	www.berndorf-stadt.at

Figure 8: Example of an EnWiki infobox

3.2 Data Extraction from Wikipedia Articles

Here we first explain the purpose of the DBpedia resources, and how they are created. Afterwards we describe how the wikitext markup, introduced in the previous sections, is converted to RDF.

DBpedia resources As a general rule, for each page a corresponding *DBpedia resource* (also referred to as *DBpedia URI*) of the form *http://dbpedia.org/resource/<page name>* is introduced. By convention, a DBpedia resource should represent the topic of its corresponding Wikipedia article and not the page itself. This transformation is necessary for Linked Data: In contrast to the DBpedia resources, the original Wikipedia article URIs cannot be resolved to RDF representations because Wikipedia does not provide them. Therefore, extractors will in general replace references to Wikipedia pages with their corresponding DBpedia resources. As a side note, as most information extracted from an article is (assumed to be) related to the topic of the article, a page's DBpedia resource will be the subject of most triples generated from it.

The *conceptual stability* of URIs is a major concern in the Semantic Web. URIs are only conceptually stable if their meaning does not change[24]. Fortunately, in [10] it was shown, that more than 90% of EnWiki's article URIs are stable in that sense. Since DBpedia mirrors these URIs, the same statistics apply to DBpedia URIs.

Extractors The original DBpedia framework consists of 11 extractors. The following list, taken from [9], gives an overview how articles and their wikitext are transformed into RDF.

- *Labels*. All Wikipedia articles have a title, which is used as an *rdfs:label* for the corresponding DBpedia resource.
- *Abstracts*. We extract a short abstract (first paragraph, represented using *rdfs:comment*) and a long abstract (text before a table of contents, at most 500 words, using the property *dbpprop:abstract*) from each article.
- *Interlanguage links*. We extract links that connect articles about the same topic in different language editions of Wikipedia and use them for assigning labels and abstracts in different languages to DBpedia resources.
- *Images*. Links pointing at Wikimedia Commons images depicting a resource are extracted and represented using the *foaf:depiction* property.
- *Redirects*. In order to identify synonymous terms, Wikipedia articles can redirect to other articles. We extract these redirects and use them to resolve references between DBpedia resources.
- *Disambiguation*. Wikipedia disambiguation pages explain the different meanings of homonyms. We extract and represent disambiguation links using the predicate *dbpprop:disambiguates*.
- *External links*. Articles contain references to external Web resources which we represent using the DBpedia property *dbpprop:reference*.
- *Pagelinks*. We extract all links between Wikipedia articles and represent them using the *dbpprop:wikilink* property.

[24] At least not significantly, whereas the definition of significance is left open.

- *Homepages*. This extractor obtains links to the homepages of entities such as companies and organisations by looking for the terms homepage or website within article links (represented using *foaf:homepage*).

- *Categories*. Wikipedia articles are arranged in categories, which we represent using the SKOS vocabulary[14]. Categories become *skos:concepts*; category relations are represented using *skos:broader*.

- *Geo-coordinates*. The geo-extractor expresses coordinates using the Basic Geo (WGS84 lat/long) Vocabulary[25] and the GeoRSS Simple encoding of the W3C Geospatial Vocabulary[26] . The former expresses latitude and longitude components as separate facts, which allows for simple areal filtering in SPARQL queries.

3.3 The DBpedia Infobox Ontology

The DBpedia Infobox Ontology is a cross-domain ontology based on infobox templates in Wikipedia articles. Its creation resulted from the attempt to solve the problems faced with an early DBpedia extraction approach, described in [3]. We will first summarize this early extraction method and explain the Infobox Ontology from there.

Originally a generic but also naive extraction method was used for generating RDF data from arbitrary templates in articles. It proceeded as follows: Each article is scanned for significant template references i.e. references to templates having a certain usage-count or references with more than a certain minimum number of arguments provided. For each argument of such a template a triple is generated according to the following procedure: The article's corresponding DBpedia resource becomes the subject, the predicate is built by the concatenation of the prefix *http://dbpedia.org/property/* and the argument-key. The argument-value becomes the object. After that some post-processing is applied to the object, such as detecting and typing numbers, or replacing links to other articles with their corresponding DBpedia resources. Additionally the DBpedia resource is classified based on the categories present in the article. Figure 9 shows an example of this data generation.

[25] `http://www.w3.org/2003/01/geo/`

[26] `http://www.w3.org/2005/Incubator/geo/XGR-geo/`

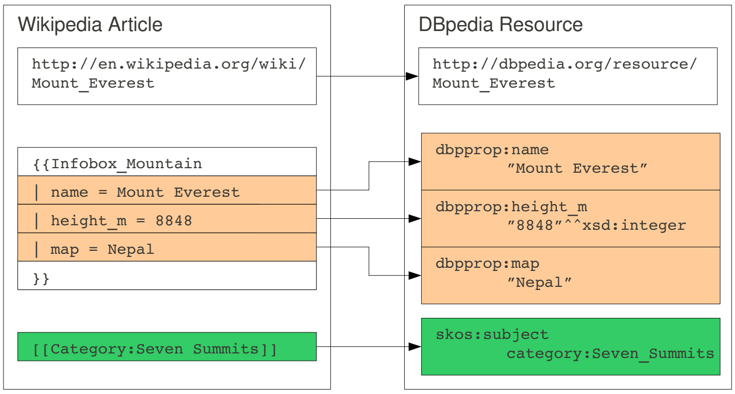

Wikipedia Article
http://en.wikipedia.org/wiki/Mount_Everest
{{Infobox_Mountain
| name = Mount Everest
| height_m = 8848
| map = Nepal
}}
[[Category:Seven Summits]]
DBpedia Resource
http://dbpedia.org/resource/Mount_Everest
dbpprop:name "Mount Everest"
dbpprop:height_m "8848"^^xsd:integer
dbpprop:map "Nepal"
skos:subject category:Seven_Summits

Figure 9: RDF data generation from articles using the generic approach

With this approach some problems soon became apparent: Wikipedia categories are not very suitable for simple classification approaches for two reasons: Firstly the meaning of a relation between an article and a category is often not the desired “is-a” but rather a “somehow related to”. Secondly, occasionally the category graph contains cycles. Therefore it was decided to use the names of infobox templates for the purpose of relating articles to classes. Further issues resulting in poor quality data being extracted by the naive method are related to the following observations:

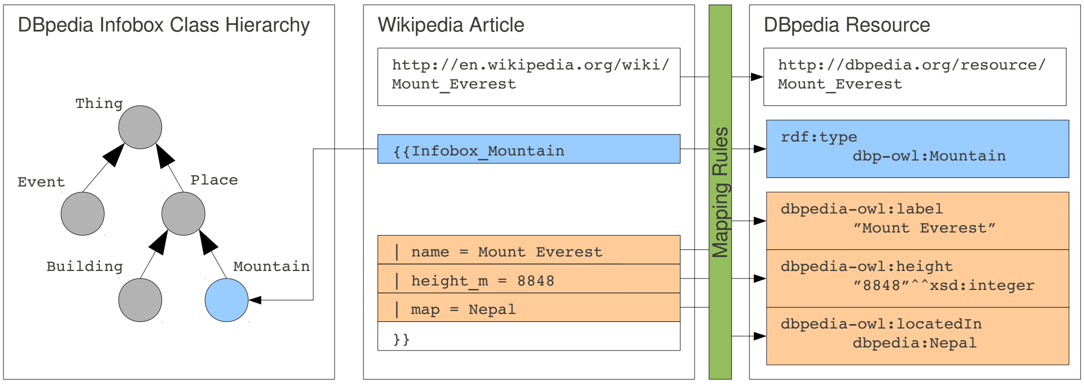
- Parameters with identical meaning may have different spellings such as *birthplace* and *place_of_birth*.
- Multiple pieces of information may be assigned to a single parameter. For example an argument may provide both a date and place for the parameter *born*. (e.g. *born = 1982, Leipzig*)
- A single piece of relevant information may be spread across multiple parameters. For instance the infobox *Infobox_NFLactive*[27] defines separate parameters for the height of a person in feet and the remainder in inches instead of providing a single parameter.
- Infoboxes with different spellings could be related to a common class.

As a solution to these problems, the *DBpedia Infobox Ontology* and the *mapping-based extraction*[1] were introduced. This ontology is an OWL ontology and can therefore

[27] http://en.wikipedia.org/w/index.php?title=Brian_Brohm&oldid=338078082

be separated into schema and instance data. The mapping-based extraction produces the instance data by mapping articles containing infoboxes to instances. It is guided by metadata about infoboxes which defines how infoboxes relate to classes, how parameters relate to properties, and how argument values should be post-processed. We will refer to this metadata as *Infobox Annotations*.

The schema and the Infobox Annotations used to be manually created and maintained by the DBpedia team from the FU Berlin. The design of the schema was mainly done in a bottom-up manner: Initially about 350 of the most popular infobox templates were identified and related to about 170 classes. After that, a shallow subsumption hierarchy was built. The DBpedia resources of articles containing such infobox templates become instances of the corresponding classes. The parameters of the infobox templates are related to properties. Figure 10 gives an impression of this process.



DBpedia Infobox Class Hierarchy
Thing
Event
Place
Building
Mountain
Wikipedia Article
http://en.wikipedia.org/wiki/Mount_Everest
{{Infobox_Mountain
| name = Mount Everest
| height_m = 8848
| map = Nepal
}}
Mapping Rules
DBpedia Resource
http://dbpedia.org/resource/Mount_Everest
rdf:type dbp-owl:Mountain
dbpedia-owl:label "Mount Everest"
dbpedia-owl:height "8848"^^xsd:integer
dbpedia-owl:locatedIn dbpedia:Nepal

Figure 10: RDF data generation from articles using the mapping-based extraction

The purpose of the DBpedia Infobox Ontology is threefold: Firstly it is used to improve the data quality, as semantically equivalent infoboxes and infobox parameters can be related to the same class and property, respectively. Secondly, it serves as a lightweight taxonomy for the Wikipedia-based instance data that can be used in SPARQL queries with inferencing enabled. And thirdly it may be used for consistency checks such as detecting erroneous information in Wikipedia articles or bugs in the Extraction Framework.

The reason the DBpedia Infobox Ontology schema was introduced at all, instead of reusing (the schema) of existing ontologies, is basically because it makes certain things easier: Although Wikipedia covers many different domains, already about 170 classes suffice for classifying 57% of the articles containing infoboxes. In comparison: OpenCyc and UMBEL define about 55000 and 21000 classes, respectively. Therefore the DBpedia team decided it would be easier to introduce classes and properties on demand when integrating new infoboxes, and interlinking them with other ontologies "lazily", rather than to use such ontology directly.

3.4 Framework Architecture

So far, we have described how articles are converted to RDF. In this section we explain the architecture of the DBpedia Extraction Framework and the infrastructure how RDF data is generated from pages, and eventually published on the Web.

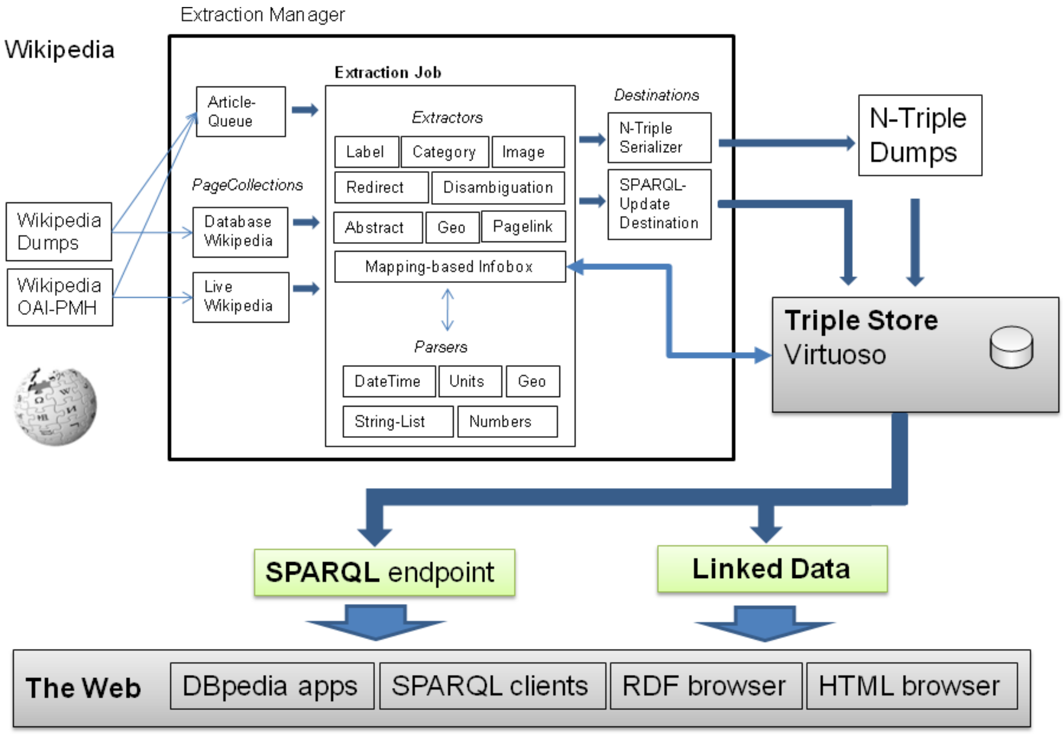


Figure 11: High level overview of the DBpedia Extraction Framework

Essentially the workflow is as follows: First pages are retrieved from some source as for example a local database loaded with a Wikipedia dump. Then the extraction process is configured and started. The process requests pages from the source, and passes them to the extractors which generate the RDF data. For that purpose extractors make use of various parsers. The generated data is finally sent to a destination, as for instance a file or a triple store. Figure 11 shows an overview of the architecture.

The original framework is written in PHP. It defines a set of core interfaces and classes whose relations between them is depicted in Figure 12. Extensions for the framework can be written by providing new implementations for these interfaces. What follows is a description of the purpose of these interfaces and classes.

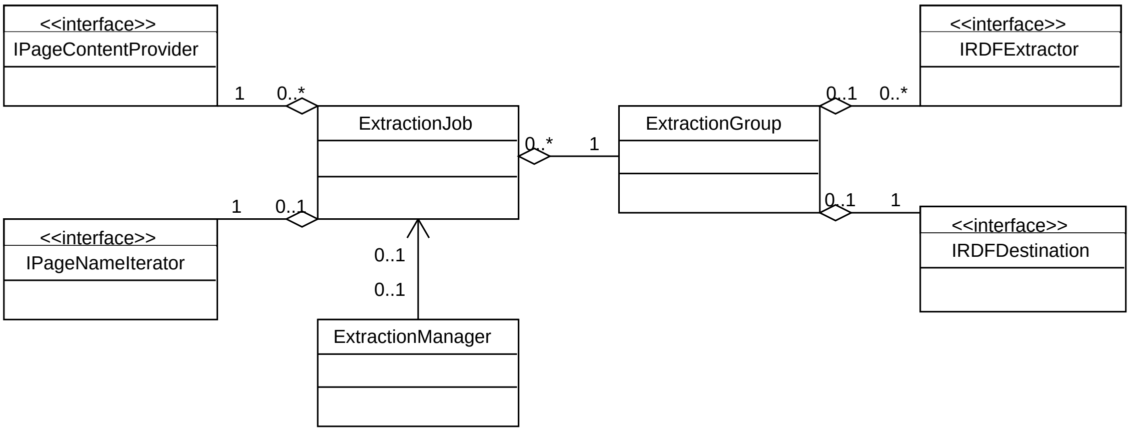


Figure 12: Class diagram of the DBpedia Extraction Framework

The following list explains the purpose of the interfaces.

- *IPageNameIterator* allows iteration of page names.
- *IPageContentProvider* resolves a page name to its content.
- *IRDFExtractor* generates RDF data from the content of a page.
- *IRDFDestination* proceeds with the generated RDF data, such as writing it to a store or printing it to the screen.

The tasks of the remaining classes are:

- *ExtractionGroup* encapsulates a specific extractor configuration and delegates its output to a given destination.
- *ExtractionJob* aggregates instances of IPageNameIterator, IPageCollection and ExtractionGroup to form a workflow.
- *ExtractionManager* executes such a workflow.

In regard to the retrieval of pages, there are implementations for fetching data from MYSQL-databases loaded with Wikipedia dumps, and from Wikipedia directly via the export-interface[28]. Various extractors have been implemented for extracting the data as described in Section 3.2. Implementations of destinations are provided for writing the RDF data to files and to triple stores directly.

In the course of this thesis modifications were made to the framework which are discussed in Section 5.

3.5 Classification and Interlinking

DBpedia is interlinked with other knowledge bases. Some of them are explained in this section.

Interlinking two OWL ontologies can be done in different ways:

1. *Class-level* interlinking relates classes to each other (e.g. via *owl:equivalentClass* and *rdfs:subClassOf*).
2. *Instance-level* interlinking relates instances to each other (e.g. *owl:sameAs*).
3. Instances of one ontology can be related to classes of the other ontology.

DBpedia itself contains links to several datasets which fall into all the above categories. A complete list is maintained on the website[29]. In the following we describe a selection of them.

Yet Another Great Ontology (YAGO) is an ontology based on Wikipedia and WordNet[30] and is described in [22]. Wikipedia's category graph is unsuitable for building taxonomies, however WordNet synsets (= sets of synonyms) already form an ontological taxonomy. YAGO's schema is created by treating Wikipedia leaf categories (those without subcategories) as classes and relating them to the WordNet taxonomy. Furthermore, YAGO also contains about a million instances based on Wikipedia articles and corresponding infoboxes. However, in contrast to DBpedia, YAGO only extracts data

[28] `http://en.wikipedia.org/wiki/Special:Export`

[29] `http://wiki.dbpedia.org/Interlinking`

[30] `http://wordnet.princeton.edu/`

for 14 properties. More details are given in [23]. A former DBpedia team wrote a script that classifies DBpedia instances against YAGO classes and relates DBpedia instances to YAGO instances.

OpenCyc [31] is the free subset of the commercial Cyc[32] knowledge base. The latest version (as of April 2009) contains about 55000 classes and 335.000 instances. DBpedia instances may be classified against OpenCyc classes and linked to OpenCyc instances via *rdf:type* and owl:sameAs, respectively.

The Upper Mapping and Binding Exchange Layer (UMBEL) [33] is an ontology with the purpose of providing a "fixed set of reference points in a global knowledge space". It defines about 21000 concepts which were completely derived from OpenCyc. DBpedia instances are interlinked with UMBEL in the same way as with OpenCyc.

Wikipedia Category graph Wikipedia categories and their relations are represented using the SKOS[14] vocabulary. The categories become instances of *skos:concept*. Relations between them are represented using *skos:broader*. DBpedia instances are related to the corresponding categories via *skos:subject*[34].

3.6 DBpedia's underlying RDF Engine - Virtuoso

Virtuoso Universal Server is a product of the company OpenLink. The name is appropriate considering that this product combines a web server, an application server and a virtual database system capable of unified handling of relational data, XML and RDF. We were particularly interested in Virtuoso's RDF capabilities. From a functional point of view it supports standard SPARQL and SPARUL, but also non-standard extensions such as counting results and sub-queries. Additionally, Virtuoso supports querying relational data as RDF through RDF Views[35]. Virtuoso comes in different flavors ranging from the freely available single-pc open source edition up to the commercial cluster version.

[31] http://www.opencyc.org

[32] http://cyc.com

[33] http://umbel.org

[34] Actually this predicate is deprecated but was not yet updated to *dc:subject*

[35] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSQL2RDF

Among the four native triples stores benchmarked in the Berlin SPARQL Benchmark[7], Virtuoso (as of version 5.0.10) achieved the best overall performance for large datasets[36].

[36] However, with a slightly modified benchmark (the "reduced query mix"), Virtuoso was outperformed by Sesame.

4 Community-Driven Engineering of the DBpedia Infobox Ontology

In this section we present a wiki-based solution which is intended to enable anyone to directly participate in the design of the DBpedia Infobox Ontology, an ontology which used to be modeled only by a small team from the FU Berlin. The structure of this chapter is as follows: We first describe the drawbacks of the original Infobox Ontology followed by an explanation of how we intend to overcome them using a community-driven ontology engineering approach. After that we describe our deployment attempt at Wikipedia and the resistance encountered there. Finally we conclude this chapter with a discussion.

4.1 A Case for Community-Driven Ontology Engineering

The mapping-based extraction, mentioned in Section 3.3, introduced new drawbacks: Firstly, it managed to relate 57% of the articles to classes of DBpedia ontology. Although this value is pretty good, considering that only about 350 infoboxes were mapped, this still leaves room for improvement. Secondly, infoboxes now and then become modified which regularly causes some of the mappings to go out of date. And thirdly, the mappings were maintained by only a few people from the FU Berlin who kept the ontology in a closed database. For that reason contributions such as adding new classes or integrating new infoboxes were only possible indirectly via requests on the mailing list. This imposed a limiting factor to the ontology's coverage, growth and up-to-dateness.

Community-driven ontology engineering seeks to overcome these drawbacks by handing a certain degree of power to a community. Wiki-like systems have emerged as a promising technology for this purpose due to several reasons: Firstly talk pages are provided that allow discussions about questions, uncertainties and conflicting views. Ideally such discussions converge to consensus. Secondly histories of all edits are kept so that undesired or even harmful changes can be reverted. And thirdly write protection mechanisms can be used to proactively protect critical resources, or reactively deal with edit-wars and repeated vandalism.

In this thesis we are taking the previous approach of the mapping-based extraction one step further: We provide the possibility for anyone to directly participate in the design of both, the DBpedia Infobox Ontology and the Infobox Annotations, in an attempt to overcome the aforementioned disadvantages. In combination with the Live Extraction Framework, described in Section 5, these changes will be reflected in near real time by updating a publicly accessible DBpedia dataset.

4.2 Template-Based Ontology Engineering

As explained in the previous sections, the schema of the DBpedia ontology is manually defined. The instance data is generated from infobox templates using an extraction method which is guided by metadata about these templates. We will refer to this metadata as *Infobox Annotations*. In this section we show how the ontology schema and the Infobox Annotations can be modeled with MediaWiki templates.

Our reasons for choosing templates are as follows. On the one hand their structured form allows for relatively easy parsing by the DBpedia Extraction Framework. On the other hand a template definition can be provided for rendering the given information in a human friendly way. As templates reduce the amount of work to be done for repetitive tasks, they are a heavily used feature throughout Wikipedia. This indicates that many people know how to use them and it can be assumed that the template syntax itself does not impose a high barrier for potential contributors to our project.

Initially we planned to host the ontology schema on MetaWiki and the Infobox annotations on EnWiki. The hoped-for advantages of doing so were:

- raising the public awareness of DBpedia – and the Semantic Web in general – by demonstrating the added value of structured data…
- …possibly resulting in Wikipedians making infoboxes more Semantic Web friendly.
- lowering the barrier for making contributions, and therefore ultimately increasing the likeliness of finding new users and further people willing to contribute.

In the next two sections we present the details about our template-based approach to modeling the DBpedia Infobox Ontology schema and the Infobox Annotations. After that we describe how these definitions are converted to RDF. Finally we discuss our deployment attempt. To anticipate the last point, the deployment of the ontology schema on MetaWiki succeeded, however the one of the Infobox Annotations on EnWiki did not, due to the resistance encountered there.

4.2.1 Schema Definitions

Our goal is to provide a solution which allows modeling the DBpedia Infobox Ontology schema with MediaWiki templates. As this ontology is an OWL ontology it consists of classes and properties. We introduce three templates for their definition, namely *DBpedia Class*, *DBpedia ObjectProperty*, and *DBpedia DatatypeProperty*. Defining a schema item works by creating a page and placing a single instance of one of these templates with respective values on it. The DBpedia Live Extraction Framework is able to recognize these edits and to extract the corresponding RDF data.

The templates' parameter names are chosen to relate as directly as possible to predicates of the RDF, RDFS and OWL vocabulary. Therefore all parameter names start with either *rdf:*, *rdfs:*, or *owl:*, followed by a corresponding unqualified property name. In the case of properties whose range is constrained to plain literals, it is valid to append the suffix @<*language tag*> to the corresponding argument key.

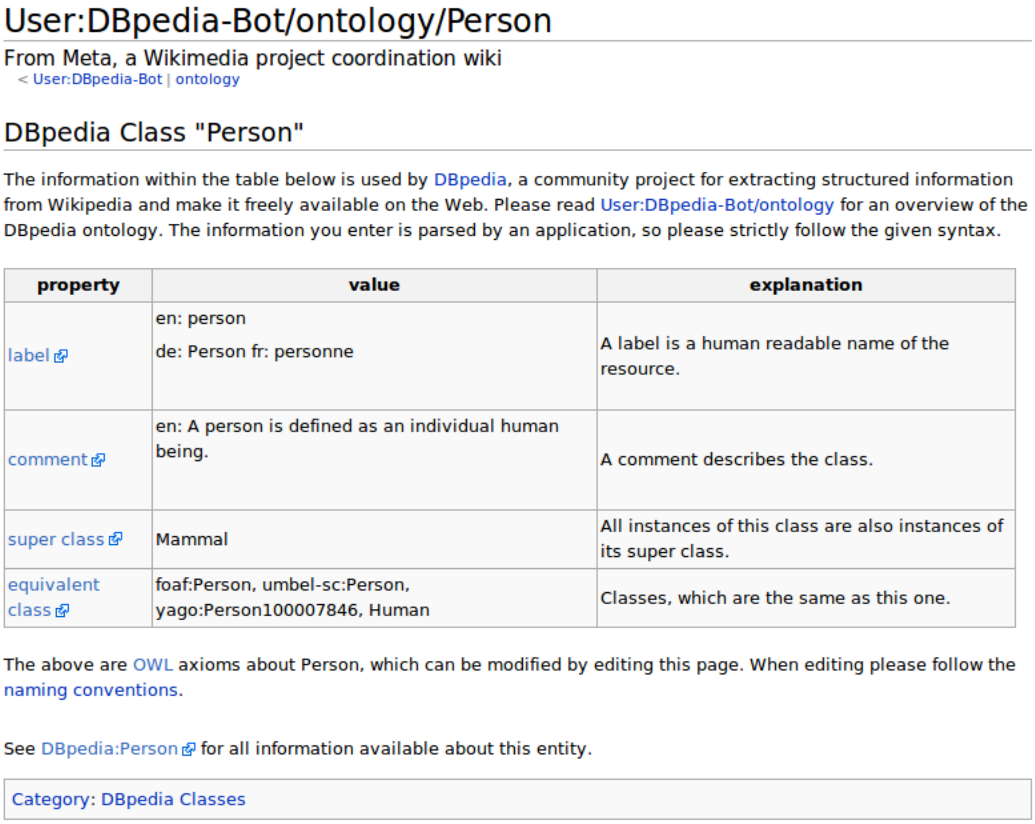
Argument values are either parsed as text or as a comma separated list of Manchester OWL Syntax expressions (see Section 2.6.2). Also, if an argument is omitted or left blank, potentially a default value will be assumed.

We are now going to present the details of each template.

Class Definitions The primary purpose of class definitions is to enable the creation of taxonomies in order to ease access to the instance data with SPARQL. Therefore the most important parameters of this template are *rdfs:subClassOf*, and *rdfs:label*. The former one is sufficient for building taxonomies, the latter is intended for assigning human friendly labels in arbitrary languages to the classes. Additionally it is usually desirable to associate a class with a short textual description of its intended meaning. This can be done via the parameter *rdfs:comment*. In order to support linking classes to other knowledge bases, the parameters *rdfs:equivalentClass* and *rdfs:seeAlso* are provided. Such external references are composed of a namespace prefix indicating the knowledge base, followed by a colon and the unqualified name of a resource addressed in it. If no namespace prefix is given, the reference is assumed to refer to an ontology item defined on another page in the same wiki. Listing 2 contains the wikitext of a class definition, whereas Figure 13 shows the corresponding presentation.

```
{{DBpedia Class
 | rdfs:label@en = person
 | rdfs:label@de = Person
 | rdfs:label@fr = personne
 | rdfs:comment@en = A person is defined as an individual human being.
 | owl:equivalentClass = foaf:Person, umbel-sc:Person, yago:Person100007846, Human
 | rdfs:subClassOf = Mammal
}}
```

Listing 2: A class definition using the *DBpedia_Class*-template



User:DBpedia-Bot/ontology/Person

From Meta, a Wikimedia project coordination wiki

< User:DBpedia-Bot | ontology

DBpedia Class "Person"

The information within the table below is used by DBpedia, a community project for extracting structured information from Wikipedia and make it freely available on the Web. Please read User:DBpedia-Bot/ontology for an overview of the DBpedia ontology. The information you enter is parsed by an application, so please strictly follow the given syntax.

property	value	explanation
label	en: person de: Person fr: personne	A label is a human readable name of the resource.
comment	en: A person is defined as an individual human being.	A comment describes the class.
super class	Mammal	All instances of this class are also instances of its super class.
equivalent class	foaf:Person, umbel-sc:Person, yago:Person100007846, Human	Classes, which are the same as this one.

The above are OWL axioms about Person, which can be modified by editing this page. When editing please follow the naming conventions.

See DBpedia:Person for all information available about this entity.

Category: DBpedia Classes

Figure 13: Presentation of a class definition

All previously mentioned parameters should be easy enough to understand by people who are not familiar with ontology engineering. Therefore at this point the barrier for participation in the DBpedia ontology design is kept rather low.

However, the expressivity of this template goes beyond these primary use cases: The final supported parameter of the template is *owl:disjointWith* which can be used for stating the disjointness of classes. The parameters *owl:disjointWith*, *rdfs:subClassOf*, *rdfs:equivalentClass* not only accept references to other ontology items, but rather a comma separated list of MOS class expressions. A simple reference is a special case of such an expressions list. A more complex example would be *Plant OR Animal*.

Table 5 contains an overview of all supported parameters of the *DBpedia class* template, whether corresponding arguments are interpreted as text or a MOS-list, and the default value that is assumed if the argument is omitted or left blank.

Parameter	Parse type	Default value
rdfs:comment	text	
rdfs:label	text	
rdfs:seeAlso	mos-list	
rdfs:subClassOf	mos-list	owl:Thing
owl:disjointWith	mos-list	
owl:equivalentClass	mos-list	

Table 5: Parameters supported by *DBpedia_Class*

Unfortunately, in order to render the information given in the template reference correctly, this approach ultimately requires the template definitions to contain parameters for every possible language tag. For example we might use the argument-key *rdfs:label@ja* to assign a Japanese label, but if the template definition does not define a parameter of the same name, it will not appear on the rendered page. The strategy we have chosen is to update these missing parameter names on demand. As the Extraction Framework only examines the argument keys of the template reference, it will recognize parameters and their language tags regardless of the parameters defined in the template definition.

Object- and Datatype Property Definitions These types of properties are defined using a similar template as the one used for class definitions. Therefore we only discuss the differences here. As with the class definitions, it is possible to specify

Parameter	Parse type	Default value
rdf:type	mos-list	
rdfs:comment	text	
rdfs:domain	mos-list	
rdfs:label	text	
rdfs:range	mos-list	
rdfs:seeAlso	mos-list	
rdfs:subPropertyOf	mos-list	
owl:equivalentProperty	mos-list	

Table 6: Parameters supported by *DBpedia_Object-/DatatypeProperty*

a label and a comment for the property being defined. The analogue of the class definitions' *rdfs:subClassOf* and *rdfs:equivalentClass* parameters for properties are *rdfs:subPropertyOf* and *rdfs:equivalentProperty*. The domain and range of properties can be specified via the *rdfs:domain*, *rdfs:range* parameters. Finally, the last parameter is *rdf:type* which is intended for defining a property to be functional, inverse-functional, transitive, or symmetric.

Listing 3 contains the wikitext of an object property definition. Its corresponding presentation is similar to the one of class definitions, and is therefore omitted. Table 6 summarizes the parameters of the respective templates.

```
{{ DBpedia ObjectProperty
 | rdfs:label = birthPlace
 | rdfs:label@de = Geburtsort
 | rdfs:label@fr = lieu de naissance
 | rdfs:comment = Relates a living thing to the place where it was born.
 | owl:equivalentProperty =
 | rdfs:seeAlso = cyc:birthPlace
 | rdfs:subPropertyOf =
 | rdfs:domain = LivingThing
 | rdfs:range = Place
 | rdf:type = owl:FunctionalProperty
}}
```

Listing 3: Example of a template-based property definition

4.2.2 RDF Generation from Schema Definitions

Here we explain how corresponding RDF is generated from the Schema Definitions that were introduced in the previous section. The first step is to create DBpedia resources for each page containing such a Schema Definition template. The corresponding DBpedia resource for such pages is *http://dbpedia.org/ontology/<name>*. Depending on whether the Schema Definition template referenced on the page is *DBpedia_Class*, *DBpedia_ObjectProperty* or *DBpedia_DatatypeProperty*, a triple is generated which relates the DBpedia resource via *rdf:type* to *owl:Class*, *owl:ObjectProperty*, and *owl:DatatypeProperty*, respectively.

As a general rule, each value of an argument whose key matches one of the templates' parameters is processed according to that parameter's parse-type. If that parse-type is *text*, then the value is turned into a plain literal with the corresponding argument key's language tag. If the parse-type is *mos-list*, then the value is split into the individual MOS expressions. Each expression is then passed to OWL API's[37] MOS interpreter, which outputs a tree structured RDF representation. Those triples themselves become part of that Schema Definition's corresponding RDF data set. Finally, triples are generated for connecting the root resources of those trees, and the plain literals to the DBpedia resource via the parameter's corresponding property. An example for the RDF data generated from a specific Schema Definition is given in Figure 14.

Wikitext

```
# This template is assumed to appear on
# an ontology page named 'LivingThing'
{{DBpedia_Class


| rdfs:label@en = "living thing"


| owl:equivalentClass = Plant OR Animal












}}
```

Generated RDF

```


dbpedia-owl:LivingThing
    rdf:type owl:Class .

dbpedia-owl:LivingThing
  rdfs:label "living thing"@en .

dbpedia-owl:LivingThing
  owl:equivalentClass bn:c .

bn:c rdf:type owl:Class .
bn:c owl:unionOf bn:l0 .

bn:l0 rdf:type rdf:List .
bn:l0 rdf:first dbp-owl:Animal .
bn:l0 rdf:rest bn:l1 .

bn:l1 rdf:type rdf:List .
bn:l1 rdf:first dbp-owl:Plant .
bn:l1 rdf:rest rdf:nil  .
```

Figure 14: Example of RDF data generated from a class definition

A question that needed answering was whether to use URIs or blank nodes for

[37] http://owlapi.sourceforge.net

inner nodes of MOS-expressions. As blank nodes are very inconvenient to query with SPARQL, we decided to use URIs. This leads to the next question of how to construct them. By default, OWL-API's MOS parser automatically generates blank nodes with identifiers that are hashes from the children of that node. Therefore the same expression will always result in the generation of the same hash. These hashes can be directly used to form URIs by prefixing them. However there are two options for choosing the prefix:

- Use a *global* prefix such as *http://dbpedia.org/ontology/expr-<hash>*
- Use a *local* prefix that contains the name of the page the triples were generated from, like http://dbpedia.org/ontology/<page>/expr-<hash>.

The advantage of the global pattern is that class expressions themselves would have unique ids and could be easily referred to[38], which might turn out to be useful in the future. However this also means, that the sets of triples extracted from schema definition pages are no longer necessarily pair wise disjoint (as the same MOS expression may be used on multiple pages). This greatly increases the complexity of managing triples in the Live Extraction Framework. Although such management strategy was implemented (see Section 5.5.3) it turned out that the performance was not good enough. This forced use to use the local URI pattern.

4.2.3 Infobox Annotations

In this section we present two templates for annotating infobox templates. The purpose and principle of these annotations are the same as that of the Mapping-Based Extraction introduced in Section 3.3: Infoboxes and their parameters are related to classes and ontology properties, respectively. Effectively, the contribution made by this thesis in that aspect is to enable the public to participate in the configuration of the Mapping-Based Extraction using a wiki-based approach. To summarize the problem, there can be relations of arbitrary cardinality between infobox parameters and ontology properties. The following examples are repeated from Section 3.3:

1. Parameters with identical meaning may have different spellings such as *birthplace* and *place_of_birth*.

[38] The DBpedia ontology is not expected to grow big enough for hash collisions to become likely.

2. Multiple pieces of information may be assigned to a single parameter. For example an argument may provide both a date and place to for the parameter *born*. (e.g. *born = 1982, Leipzig*)

3. A single piece of relevant information may be spread across multiple parameters. For instance the infobox *Infobox_NFLactive*[39] defines separate parameters for the height of a person in feet and the remainder in inches instead of providing a single parameter.

We name the operations required to deal with these cases *map*, *split*, and *merge*. A depiction of these cases is shown in Figure 15.

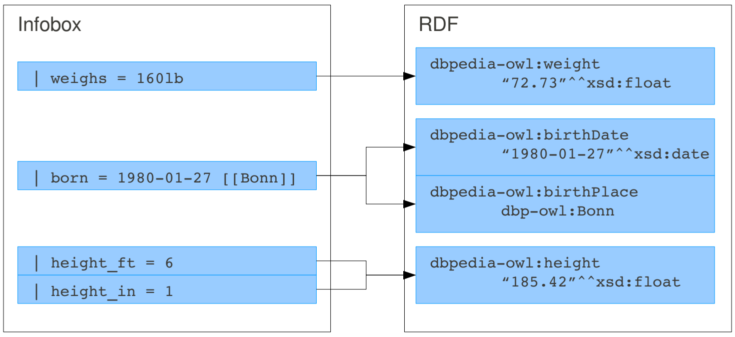

Infobox
| weighs = 160lb
| born = 1980-01-27 [[Bonn]]
| height_ft = 6
| height_in = 1
RDF
dbpedia-owl:weight "72.73"^^xsd:float
dbpedia-owl:birthDate "1980-01-27"^^xsd:date
dbpedia-owl:birthPlace dbp-owl:Bonn
dbpedia-owl:height "185.42"^^xsd:float

Figure 15: Illustration for map, split, and merge cases.

In our solution we only address the map and split cases, as for them defining mapping rules is simple enough to be understood by many people. The problem with merge cases is, that the required expressivity would not be very user friendly. For instance merging the parameters *height_ft* and *height_in* would involve converting feet and inches to a common unit and performing an addition. In the original Mapping-Based Extractor handling of such cases was hard coded.

We introduce the template *DBpedia infobox annotation* which defines the two parameters *relatesToClass* and *mapping*, which serve the purpose of relating infoboxes to classes and parameters to properties, respectively. An example of such a template is shown in Listing 4.

[39] http://en.wikipedia.org/w/index.php?title=Brian_Brohm&oldid=338078082

```
{{DBpedia infobox annotation
 | relatesToClass = Musician
 | mapping =
   {{ DBpedia map | born | birthPlace | links}}
   {{ DBpedia map | born | birthDate  | date}}
   {{ DBpedia map | weighs | weight | lb}}
}}
```

Listing 4: Example of an Infobox Annotation

The parameter *relatesToClass* is interpreted as the name of a single ontology class. As these classes are defined on DBpedia ontology definition pages, the given value will appear as a link to the corresponding page.

The parameter *mapping* is intended for specifying a list of parameter-to-property mappings via instances of the *DBpedia map* template. The three parameters of that template are (1) the name of the parameter being mapped, (2) the name of the ontology property being mapped to, and (3) the *parse hint*, indicating which parts of an argument should be used for triple generation.

Triple generation for such mapped parameters works as follows: Whenever the Extraction Framework extracts data from an article referencing an annotated infobox, the article's corresponding DBpedia resource will become an instance of the given class.

The parse hint indicates which parts to match from the argument-value, and how to process them. For each matching value, a triple with the following properties is generated: The DBpedia URI corresponding to the page containing the infobox becomes the subject, the property corresponding to the parameter becomes the predicate and the processed value becomes the object. If the parse hint results in multiple parts of the argument-value being matched, then a triple is created for each of them.

Parse hints are used by the DBpedia Extraction Framework for matching parts of an argument-value and processing these parts in order to form the objects of triples. For instance, the parse hint for pounds, *lb*, indicates to look for plain numeric values like *10*, and numeric values that have units associated with them, such as *12kg* or *5m*. If there is no unit, then the value should be assumed to be in pounds. Otherwise, only values whose quantity equals that of the parse type are considered matches. For instance, the quantity of both *lb*, and *kg* is *Weight*. However the quantity of *m* is *Length*. Therefore the value

Quantity	Unit	Parse hint
Length	metre	m
	km	km
	inch	in
	foot	ft
	mile	mi
Area	square metre	m2
	square kilometre	km2
	square mi	mi2
Volume	cubic metre	m3
Temperature	celsius	C
	fahrenheit	F
	kelvin	K
Weight	kilogram	kg
	pound	lb
Flow rate	cubic metres per second	m3/s
	cubic foot per second	ft3/s
Population density	individuals per square kilometre	pop/km2
	individuals per square mile	pop/mi2

Table 7: Parse hints for units

5m would be ignored by a mapping rule with parse hint *lb*.

Currently supported parse hints of the framework are:

- *text* The whole argument-value is matched for being converted to a plain literal.

- *links* All links to other pages are matched, and the names of the link target converted to their corresponding DBpedia resources.

- *dates* Matches dates in order to generate typed literals.

- *currency* Matches values that have a currency associated with them, such as *100$*.

- The parse hints for specific units are shown in Table 7. We want to stress, that these parse hints are not used for specifying the target unit values should be converted to. Instead, they define the default unit that should be assumed for numeric values without a unit.

A small real world example Consider the excerpt in Listing 5, taken from the infobox about “Björk”[40]

```
{{Infobox musical artist
| Born = {{birth date and age|df=yes|1965|11|21}}<br /> [[Reykjavik]],
[[Iceland]] }}
```

Listing 5: A small excerpt from an infobox

[40] http://en.wikipedia.org/w/index.php?title=Bj\%C3\%B6rk&oldid=343461245

As we can see, the parameter born contains both links relating to the birth location as well as a date relating to her birth date.

We now use the snipped in Listing 6 to annotate `Infobox_musical_artist`[41].

```
{{DBpedia template
 | relatesToClass = Musician
 | mapping =
   {{ DBpedia attribute | Born | birthPlace | links}}
   {{ DBpedia attribute | Born | birthDate  | date}}
}}
```

Listing 6: Annotation of the infobox

In this case the birth date and birth place can be easily separated using the appropriate parse hints. In this case the parse hint *date* is used to match the *birth date and age* template in order to eventually generate a RDF date literal from it. As there are multiple links given for birth place, a triple will be generated for each of them. In the case that the link targets are again articles which contain annotated infoboxes, it is indirectly possible to tell the country and the city apart.

4.3 Deployment

The deployment plan was as follows: The Schema Definitions should be hosted on MetaWiki, and the Infobox Annotations should be added to the corresponding infoboxes' documentation pages. We discuss these two deployments separately.

4.3.1 Deployment of the Schema Definitions

One of the major questions about the DBpedia ontology was whether there should be an individual DBpedia Infobox Ontology for each language version of Wikipedia or whether there should be only a single ontology for all language versions. Eventually the DBpedia team decided in favor of the latter option as it seemed easier having to only maintain a single ontology. As a consequence, language specific Wikipedias were considered to be unsuitable hosts.

Eventually *MetaWiki* was chosen as a suitable host for the following reasons:

[41] `http://en.wikipedia.org/wiki/Template:Infobox_musical_artist`

- One of the roles MetaWiki serves is being a multilingual discussion forum for various matters concerning all Wikimedia related projects. These projects include (but are not limited to) all language versions of Wikipedia. It seemed that the Schema Definitions could be one of those matters.

- It is possible to create interwiki-links to MetaWiki from any Wikipedia. Therefore classes and properties referenced in Infobox Annotations could be displayed as links to Schema Definition pages on MetaWiki.

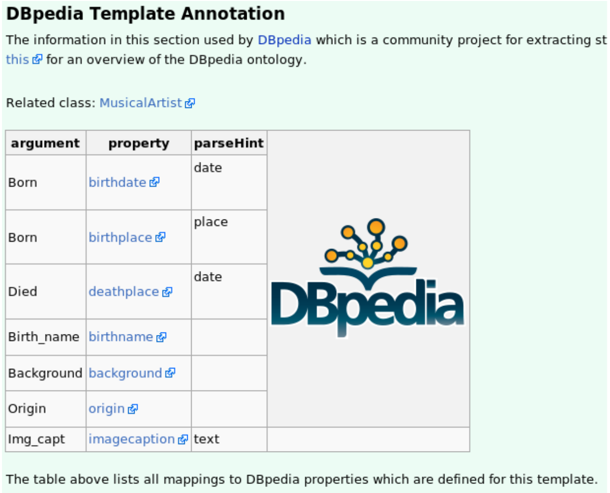
- As MetaWiki and Wikipedia are both based on the MediaWiki software, they both support the same wikitext.

The original DBpedia ontology was automatically converted into the template representation using a script. Each of these templates then needed to be placed on appropriate pages located at *http://meta.wikimedia.org/wiki/User:DBpedia-Bot/ontology/<name>*. As about 200 classes, 400 object and 300 data type properties were converted, a total of 900 pages needed to be upladed. In order to avoid having to do the upload manually, a script was written which automatically performed the upload using the MediaWiki API. Wikipedia policies state, that large amounts of automatic edits must be done using a dedicated user account with permission to perform these edits. As a consequence, the user *DBpedia-Bot* was created. There is a reason why the ontology resides in the user space of this bot, rather than the main namespace. Initially the ontology was uploaded to *.../wiki/DBpedia/ontology*, but it was found out to clutter up searches, as various labels are used throughout the ontology. As by default, searches on MediaWiki are only carried out on the main namespace, it was suggested to move it to a different namespace.

4.3.2 Deployment of the Infobox Annotations

The infobox annotations were intended to be added to corresponding infoboxes' documentation pages on EnWiki. In contrast to the ontology schema pages, which were all newly created, these documentation pages usually already contained content contributed by other Wikipedians. As there were only about 350 infobox annotations, we considered that the risk of damaging these pages with automatic edits is high enough to favor a manual approach. Therefore the annotations were distributed among the members of the

DBpedia team who manually performed the edits. At that time, an early design of the Infobox Annotation templates was used, which is depicted in Figure 16.



DBpedia Template Annotation

The information in this section used by DBpedia which is a community project for extracting st
this for an overview of the DBpedia ontology.

Related class: MusicalArtist

argument	property	parseHint
Born	birthdate	date
Born	birthplace	place
Died	deathplace	date
Birth_name	birthname	
Background	background	
Origin	origin	
Img_capt	imagecaption	text

The table above lists all mappings to DBpedia properties which are defined for this template.

Figure 16: Early design of the presentation of an Infobox Annotation

Unfortunately during this step the teams' user accounts were blocked. An entry was made on the adminstrators' notice board[42] where the team was accused of spamming. Eventually a large discussion arose there of whether the Infobox Annotations could be placed on EnWiki, and if yes, whether the realization is appropriate. At some point, the discussion was continued on the technical village pump[43].

What follows is a summary of the arguments for and against the DBpedia approach. Some arguments we encountered in the discussion showed a lack of understanding of our intentions. Other arguments showed our lack of understanding the Wikipedia community processes. Although there was support for the general idea of making infobox data better accessible for machines, there were concerns about our technical implementation. In the following we discuss some arguments that were brought up in the discussion.

- "*Putting the Infobox annotations on En-Wiki makes extraction easier for DBpedia*".

  This is not true. On the contrary, the extraction job becomes more complex by that. However, what we do expect in the mid run is, that the maintainance of the Infobox Annotations becomes easier once a community has been established and some tools have been created.

[42] http://en.wikipedia.org/wiki/Wikipedia:Administrators'_noticeboard/IncidentArchive576#DBpedia_spamming_infobox_templates

[43] http://en.wikipedia.org/wiki/Wikipedia:Village_pump_(technical)/Archive_67#For_your_attention_-_DBpedia_templates

- "*The annotations are very bloated and ugly, and totally dominating the pages they were on. Also, the enormous logo gives a strong impression of spam.*"

  Unfortunately this is correct. While many Infobox Annotations were relatively small and therefore did not look too bad, some annotation tables exceeded fifty rows. This issue was addressed with a revised presentation of the infobox templates, as explained later.

- *We were not granted permission to do these changes.*

  Ultimately this is true, although we talked with people, who were to a certain extend involved in Wikipedia, about our plans beforehand. Since they did not oppose them, we though it would be ok to proceed. In fact none of us was an expert at Wikipedia community processes.

- "*Wikipedia is not your web host and your content does do nothing to serve Wikipedia. If we allowed DBpedia to put such unrelated content on Wikipedia, everyone else would also have to be allowed to do so. Therefore the content does not belong there.*"

  Our hopes were that Wikipedia would also benefit from the project: The annotations would make Wikipedia more attractive for other extraction projects as well, since they could freely reuse our annotations.

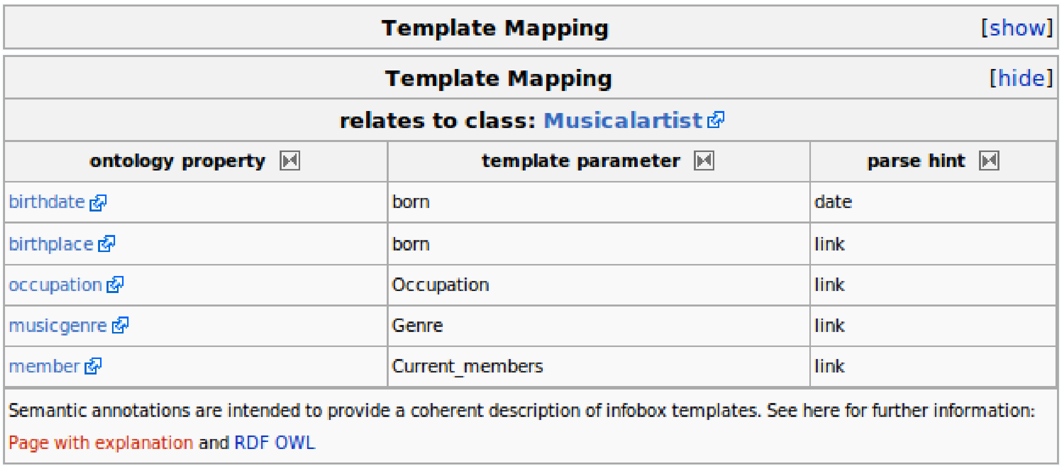
- "*Some redundancy of infoboxes is due to the lack of proper programming language support which results in many similar infoboxes to be created which serve a similar purpose. This is another reason why parameters are not streamlined. But instead of introducing mappings to fix that, rather introduce a proper programming language.*"

  A programming language would be a nice thing to have, but it would not help us relating infoboxes to classes, or streamlining the names of certain parameters. Our approach would still be useful.

- "*Infoboxes are a mess. But building a layer of abstraction over this mess is the wrong thing to do. It would not be hard standardizing infoboxes once the need occurs.*"

Our point is, that there is a high barrier of making modifications to the infoboxes directly as they affect all articles which use them. However the Infobox Annotations have no effect on Wikipedia. The burden of processing such changes is on DBpedia's side.

Revised Presentation of Infobox Annotations In the course of the discussion the DBpedia team filed a "Request for Comment" (RFC) with the purpose of explaining our goals and approaches to the Wikipedia community. In this RFC[44] a revised version of the appearence of the Infobox Annotation template is presented. The DBpedia logo is no longer present, the font is sleeker, and most importantly, its visibility can be toggled. A depiction of it is shown in Figure 17. Therefore if that template was allowed on the documentation pages, it could by default be shown collapsed, resulting in as little space as possible being used up, effectively being no longer as bloated as our original version was.



Template Mapping [show]

Template Mapping [hide]

relates to class: Musicalartist

ontology property	template parameter	parse hint
birthdate	born	date
birthplace	born	link
occupation	Occupation	link
musicgenre	Genre	link
member	Current_members	link

Semantic annotations are intended to provide a coherent description of infobox templates. See here for further information: Page with explanation and RDF OWL

Figure 17: The revised presentation for Infobox Annotations

Unfortunately the outcome of the discussion was, that there was no consensus for another deployment attempt with the revised Infobox Annotations. As a consequence, the DBpedia team decided to set up their own MediaWiki instance[45] for hosting the Schema Definitions and the Infobox Annotations. The templates used on the new wiki are based on the design presented in this chapter, but were slightly improved.

[44] http://en.wikipedia.org/wiki/Wikipedia_talk:Requests_for_comment/infobox_template_coherence

[45] http://mappings.dbpedia.org

4.4 Discussion

The discussion is divided into three parts: First we discuss general issues with the DBpedia Infobox Ontology and the Schema Definitions. Afterwards we focus on usability aspects of our approach. Finally, we conclude this chapter with future work.

4.4.1 What Kind of Ontology

During the development of the Extraction Framework for the ontology definitions it became apparent that even among the DBpedia team there were different views about the nature of the DBpedia ontology. Essentially we identified three major diverging view points:

- The ontology should capture the essence of EnWiki's infoboxes. Therefore it should neither contain very abstract class hierarchies nor superficial classes (classes with hardly any instances). For example, OpenCyc defines *physical information bearing object*[46] as a subclass of *hexalateral object*[47]. Both classes do not seem very useful for classifying infoboxes.

- The ontology should be "unconstrained": Eventually the contributors decide what kind of ontology they want to have. Their edits will lead to discussions which ultimately lead to the definition of rules of what content should be permitted and what not.

- An application ontology: It should be permitted to add application specific data to the ontology. For instance, it was suggested to allow stating that the heights of people should be extracted in centimeters, as this would make it easier to display that information in a specific user interface.

In our opinion the first option is the only sensible way to go. The schema of the Infobox Ontology should provide an intuitive and pragmatic entry point to the instance data, and a basis for linking it to other knowledge bases. In regard to the second option, it is likely that such approach leads to confusion. At the very least, we have to set up

[46] http://sw.opencyc.org/2009/04/07/concept/en/HardcopyInformationBearingObject

[47] http://sw.opencyc.org/2008/06/10/concept/en/HexalateralObject

the basic rules that point into the direction the project should head. Application specific information should be kept elsewhere.

As for issues with the Schema Definitions, just one wrong *rdfs:subClassOf* could already have a significant effect on the classification of instances (assume someone stating that *Place* is the same class as *Person*). Even worse, a wrong value for *owl:disjointWith* may render the schema inconsistent. A possible solution to these cases would be, to set the live extraction framework up the way, that the Schema Definitions are piped through a reasoner. This would allow for detecting major schema changes and inconsistencies as soon as they become introduced. Unfortunately due to the other challenges and barriers we faced during this thesis, we ran short of time. Therefore we need to leave such an implementation as well as the design of what procedure to follow, if such events are detected, for future work.

4.4.2 User Friendliness vs Expressivity

A major problem is keeping the balance between user friendliness and expressivity. For instance, the current implementation of the Infobox annotations does not support specifying rules for mapping multiple infobox parameters to a single ontology property. These cases are still handled with the original Extraction Framework, where these rules are hard coded. Furthermore, the Infobox Annotations do not provide any means of conditional mappings, which would be very useful for example for the infobox *Infobox_musical_artist*: According to its documentation[48], the value given for the parameter *background* should categorize the topic being described, so whether it's for example a solo singer, a band, or a classical ensemble. In regard to our Infobox Annotations, it would mean that the related class depends on the value given for that parameter. In that case, the question is, whether to extend the Infobox Annotations with conditionals, or define such classes implicitly using OWL axioms. Unfortunately either solution is rather complex to use. For example, the content of Listing 7 would be necessary for stating: "Any resource using the template *Infobox_musical_artist*, and having the value *Classic ensemble* for the parameter *background*, should become an instance of *ClassicEnsemble*.

[48] `http://en.wikipedia.org/w/index.php?title=Template:Infobox_musical_artist/doc&oldid=343114884`

```
Class: ClassicEnsemble
  EquivalentTo: wikiPageUsesTemplate Infobox_musical_artist AND
                background value "Classic ensemble"
```

Listing 7: Classification with OWL axioms

Another issue is, that there are articles which cover multiple very similar topics and therefore transclude multiple infoboxes. For example many manga/anime related articles deal with the comic, the computer game, the TV series, and the movie at once. Some of the infoboxes in question are named *Infobox animanga/Film*, *Infobox animanga/Video*, *Infobox animanga/Print*. In such cases it would be necessary to generate distinctive resources for each meaning of the article. A possible solution would be to extend the Infobox Annotation language to allow for specifying arbitrary strings that will become appended to the respective DBpedia-URI. For instance if an article named <*name*> uses the infobox *Infobox animanga/Film*, a resource .../<*name*>/*film* could be created.

Additional complexity is introduced as the more Infobox Annotations and Schema Definitions there are, the harder it it becomes to keep track of them.

4.4.3 Future Work

The deployment of the Infobox Annotations turned out to be a very heavyweight process due to the resistance encountered. Even the successful deployment of the ontology schema on MetaWiki took its time, as permissions for automatic edits had to be requested, and once even the whole ontology had to be moved to a different namespace. Eventually the DBpedia team decided that at current state, the flexibility gain by setting up their own wiki outweighs the advantages of a deployment on EnWiki and MetaWiki. Eventually, the DBpedia team from the FU Berlin set up their own publicly accessible MediaWiki instance[49] for the purpose of modeling the Schema Definitions and Infobox Annotations. The templates used on that wiki are a slightly improved version of the ones presented in this thesis and were jointly developed with us. Currently the wiki provides the following tools:

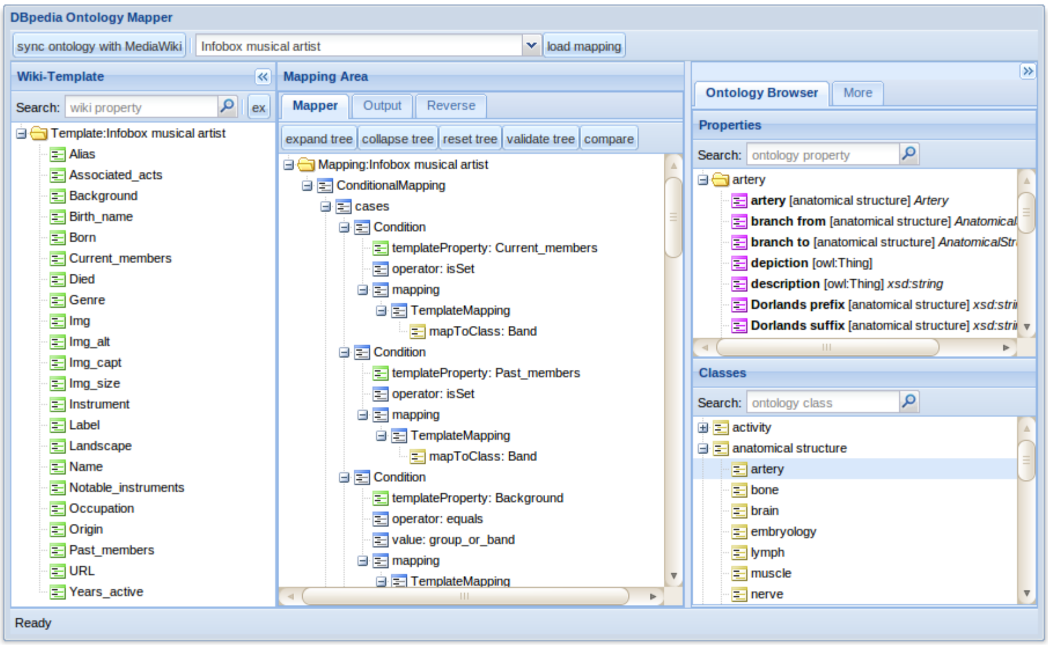
- An *Ontology View*, which provides an overview of the DBpedia Ontology.

[49] http://mappings.dbpedia.org

- A *Mapping Validator*, which checks for syntactical correctness and ontological consistency.
- An *Extraction Tester* for extracting EnWiki pages against given Infobox Annotations, in order to provide direct feedback on the mapping.

The reason MediaWiki is used, is, that depending on the success of this new wiki, another deployment of the templates, together with the newly developed tools, might be attempted on EnWiki in the future. An alternative goal would be to move the mapping-wiki into the Wikipdia farm in the sense that potential DBpedia-contributors could reuse their Wikipedia login rather than having to register at the mapping-wiki separately.

Currently the *DBpedia Ontology Mapper*, depicted in Figure 18, is being developed by the FU Berlin with the aim of easing the annotation and schema definition process. It provides a search field with auto-completion for browsing available infoboxes. When one is selected, its corresponding parameters are displayed in the left panel, whereas the Infobox Annotations can be edited in the center panel. The right panels are used for editing the Schema Definitions. Any changes made with this tool should be eventually written back as templates on the wiki again. Unfortunately, there has not been an official release at the time this thesis was finished.



Template:Infobox musical artist
Alias
Associated_acts
Background
Birth_name
Born
Current_members
Died
Genre
Img
Img_alt
Img_capt
Img_size
Instrument
Label
Landscape
Name
Notable_instruments
Occupation
Origin
Past_members
URL
Years_active

```
Mapping:Infobox musical artist
  ConditionalMapping
    cases
      Condition
        templateProperty: Current_members
        operator: isSet
        mapping
          TemplateMapping
            mapToClass: Band
      Condition
        templateProperty: Past_members
        operator: isSet
        mapping
          TemplateMapping
            mapToClass: Band
      Condition
        templateProperty: Background
        operator: equals
        value: group_or_band
        mapping
          TemplateMapping
```

artery
artery [anatomical structure] *Artery*
branch from [anatomical structure] *Anatomical*
branch to [anatomical structure] *AnatomicalStr*
depiction [owl:Thing]
description [owl:Thing] *xsd:string*
Dorlands prefix [anatomical structure] *xsd:stri*
Dorlands suffix [anatomical structure] *xsd:stri*

activity
anatomical structure
artery
bone
brain
embryology
lymph
muscle
nerve

Figure 18: The DBpedia Ontology Mapper

5 DBpedia Live Extraction

The aim of the DBpedia Live Extraction is to extract RDF data from relevant pages on EnWiki and MetaWiki immediately after they are edited. For each of these pages a corresponding dataset is maintained in a triple store. Whenever a page is edited, any previously existing dataset is discarded and the dataset corresponding to the page's latest revision is inserted. In our case the relevant pages are the articles and the Infobox Annotations (see Section 4.2.3) on EnWiki, and the Schema Definitions on MetaWiki (see Section 4.2.1). Although so far the deployment of the Infobox Annotations failed, as discussed in Section 4.3.2, we still refer to these two wikis in order to ease the discussion.

DBpedia Live enables us to overcome the manual efforts originally required to produce the DBpedia datasets. But even more interesting is the fact that the datasets will contain up-to-date data, thereby making certain use cases more appealing. These are summarized as follows:

- *List maintenance*: Lists containing up-to-date data from EnWiki can be easily aggregated with SPARQL queries. Therefore in some cases DBpedia Live can be used instead of having to write custom scrapers or even having to manually keep such lists up-to-date. For instance a fan site of a movie could power a widget displaying a list of current movies done by the same director with DBpedia Live.

  As a side note: Such widget would have to present the DBpedia data in some way. Fortunately there are tools which simplify this task. For instance, arbitrary presentations of RDF data (e.g. as HTML) can be defined with the Fresnel vocabulary[15]. Examples of corresponding implementations are given in the same reference. Another solution for this problem is provided by LESS[2] where presentations are defined based on templates.

- *Concept Tagging*: In contrast to free tagging which allows people to use arbitrary pieces of text as tags, concept tagging limits the choice of tags to a predefined set of concepts. One aim is to overcome the problems of synonyms (e.g. New York City and NYC) and homonyms (e.g. Jaguar could be an animal or the brand of a car). Among others, Wikipedia and DBpedia have been used as such concept sources. However, DBpedia Live will now be on a par with Wikipedia's up-to-dateness and therefore recent events such as the Haiti-Earthquake[50] will be available as concepts immediately.

In the following sections we first describe the DBpedia live datasets. Then we present the contributions made to the DBpedia extraction framework. Afterwards we go into its technical details, especially concerning the triple management strategies involved. Finally we describe the new extractors that were created in the course of this thesis.

5.1 The DBpedia Live Dataset

The DBpedia Live dataset made publicly available is composed of the following four major parts: (1) the dataset based on EnWiki pages, (2) the dataset containing the schema definitions extracted from MetaWiki, (3) metadata for the schema definitions with the purpose of simplifying certain queries and finally (4) these links to the knowledge base described in Section 3.5. The first three datasets are dynamically updated whenever respective pages are modified. The fourth dataset is "static", in the sense that is unaffected by such edits.

All datasets reside in the graph `http://dbpedia.org`, except for (3) which is stored in `http://dbpedia.org/meta`. We will refer to these graphs as *DataGraph* and *MetaGraph*, respectively. Access is provided via SPARQL[51] and the Linked Data interface[52]. As a side note, during development, the MetaGraph was intended to also hold metadata about every single triple in (1). Especially the information about the page, the extractor, and the point in time a triple was generated is essential for creating incremental database dumps. Due to performance reasons we eventually refrained from solving this using the MetaGraph. Instead we decided to keep this information in a separate database as described in Section 5.5.5.

The metadata vocabulary about the schema definitions consists of the following properties:

- *dbpmeta:sourcePage* The DBpedia-resource of the page the triple was extracted from e.g. `dbpedia:ontology/birthPlace`.

[50] `http://en.wikipedia.org/wiki/2010_Haiti_earthquake`
[51] `http://dbpedia-live.openlinksw.com/sparql`
[52] `http://dbpedia-live.openlinksw.com/resource/`

- *dc:modified* When the triple was extracted the last time e.g. *2009.11.24T09:10:36*
- *dbpmeta:usedExtractor* The URI of the extractor that was used for generating the triple. Currently the only value is `dbpmeta:TBoxExtractor`.
- *dbpmeta:aspect* Multiple triples may be generated from arguments of ontology definition templates such as *owl:equivalentClass = Plant OR Animal* (see Table 14 in Section 4.2.2 for an example of the generated RDF). In order to be able to easily query those triples with SPARQL, their reifiers are related to the argument-key's corresponding property (in this example to *owl:equivalentClass*) via the *dbpmeta:aspect* property. An example query is given in Listing 8.

```
SELECT ?s ?p ?o
    FROM <http://dbpedia.org/meta>
WHERE {
    _:b owl:annotatedSource     ?s .
    _:b owl:annotatedProperty   ?p .
    _:b owl:annotatedTarget     ?o .
    _:b dbpmeta:sourceResource dbpedia-owl:LivingThing .
    _:b dbpmeta:aspect         owl:equivalentClass .
}
```

Listing 8: Querying sets of triples for specific properties using *dbpmeta:aspect*

5.2 Requirements

What follows is a list of the most important requirements that had to be met.

- *Performance* On average, about 2.3 pages are edited on EnWiki per second[53], totalling nearly 200.000 pages per day. Among these pages, approximately 150.000 are articles, 2500 are templates, and less than 150 are template-doc pages. These are the page types that are relevant to DBpedia Live. Ideally, their edits are handled immediately, which means, that the average extraction time of a single page must not exceed 0.56 seconds, which is very little time.

  However, if pages were repeatedly edited within a certain period of time, the overall amount of pages required to be processed could be reduced, by dealing with only the most recent of their edits.

[53] Measured on 14th May 2010.

We measured the potential savings with the following two strategies:

1. Every time a page is edited, a count down is started for it. We then count the total number of subsequent edits while the count down has not reached zero. This is the number of edits that need not be processed.
2. Same as (1), except that every subsequent edit resets the count down.

Figure 19 shows the ratio of saved edits to the total number of edits for count downs ranging from 0 to 30 minutes. For instance, a value of 5 minutes reduces the workload by 20%, which increases the allowed average processing time per page to 0.7 seconds. DBpedia Live is not required to process edits immediately, however, it should not use delays greater than 5 minutes.

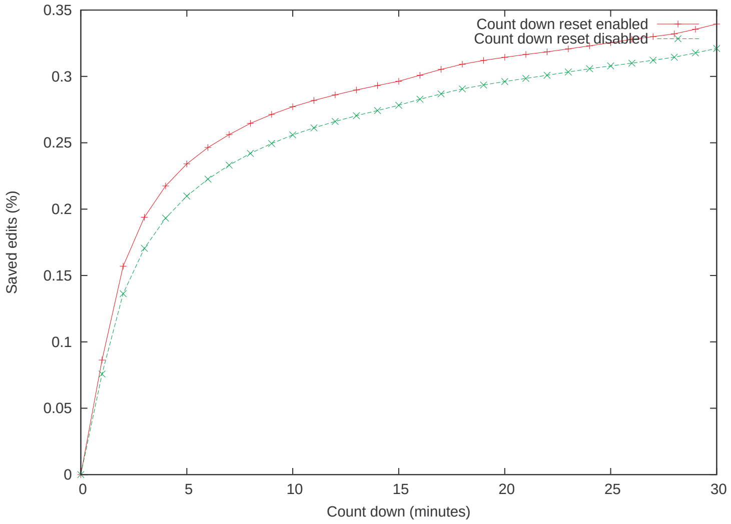


Figure 19: Workload reduction vs article process delay

MetaWiki's total edit rate is more relaxed with less than 1500 edits per day, and since we are only interested in the ontology schema definitions, we expect far less than a hundred daily relevant edits.

- *Cleanliness* The store should only contain the most recent triples extracted from a page. Therefore outdated triples must be removed.
- *Deployment* The Live Extraction was destined to be hosted at one of OpenLink's servers. In the early stages this was a test server. After the extraction was running

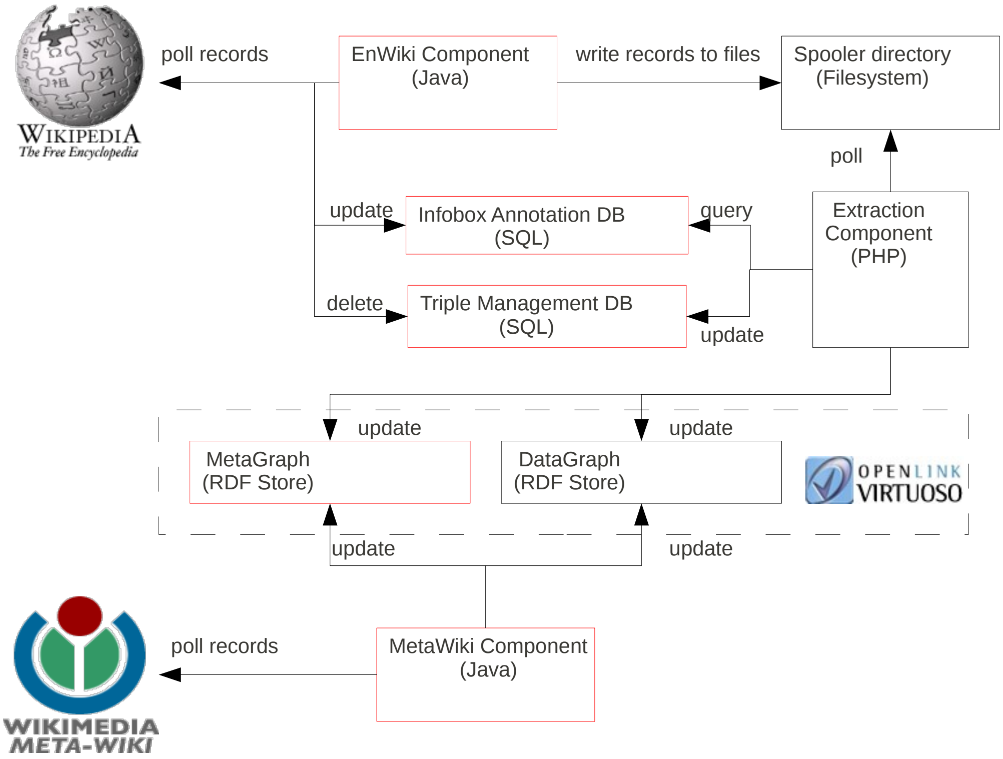
stable, it has been moved to its final destination. Due to the hosting of other services besides DBpedia on those servers, we were not granted direct access to them. For that reason it was important to pay attention to robustness and logging.

- *Robustness* The Live Extraction process is a long running one. Once the process is initialized successfully, it must recover from most errors encountered during runtime. Concretely these errors are: connection loss to any of the frameworks own relational databases, its triple stores, EnWiki, and MetaWiki. Furthermore it must be considered that extractors may sometimes bail out (e.g. due to invalid wikitext).

- *Logging* The purpose of logging is to keep track of all relevant internal events encountered during the extraction process. Since we did not have access to the server the extraction was running on, this was crucial for debugging.

5.3 DBpedia Live Architecture

The Live Extraction Framework is an extension of the original Extraction Framework. New components were introduced, which will be briefly described in this section. The components and their relationships are depicted in Figure 20.



Contributed components are marked red.

Figure 20: Overview of the DBpedia Live Extraction Framework

DataGraph The graph containing the RDF data extracted from articles and Schema Definitions.

MetaGraph A newly introduced graph providing metadata about the triples in the DataGraph, such as which page they originated from, which extractor produced them, or when they were created.

Infobox Annotation DB A database holding the Infobox Annotations extracted by the EnWiki component. The *LiveInfoboxExtractor* uses this information in order to guide its extraction process.

Triple Management DB A database used for managing triples. Sets of triples are associated with the page and extractor that generated them. This database is needed for identifying the triples that need to be removed from the DataGraph in the event of page edits.

MetaWiki component This component solely serves the purpose of extracting the Schema Definitions from MetaWiki. The extracted triples are stored in the DataGraph. Furthermore, metadata about those triples comprising the corresponding extractor, page, and timestamp are inserted into the MetaGraph.

EnWiki component This component was initially introduced in order to account for fetching recent edits from EnWiki's update stream and saving them to files. Java was chosen as existing libraries such as Jena and OWL-API could be reused. As time passed, this component grew, so it now also handles the deletions of articles and the extraction of Infobox Annotations.

Extraction component This is the original extraction framework. During the development of the Live Extraction Framework, its core architecture (outlined in Section 3.4) remained unchanged. However, some sub-components were created or adapted. These changes are:

1. Implementations of the interfaces *IPageNameIterator* and *IPageContentProvider* were created, which support processing the files generated by the EnWiki component.

2. A new extractor for infoboxes taking Infobox Annotations into account was created. This extractor is known as the *LiveMappingBasedExtractor*.

3. The *ActiveAbstractExtractor* generates abstracts from articles. While its performance surpasses that of the previously existing *AbstractExtractor*, the quality of the output is unfortunately noticeable worse.

5.4 Extraction Workflows

In this section we first explain how pages that were recently edited can be retrieved from EnWiki and MetaWiki. Then we describe how the live extraction framework processes them.

5.4.1 Retrieving updates from MediaWiki

A fundamental prerequisite for DBpedia Live is to have access to updates from Wikipedia in near realtime. This functionality is provided by the freely available *OAIRepository* MediaWiki-extension which is especially deployed at EnWiki and MetaWiki. This extension makes it possible to poll modifications of pages.

The OAIRepository MediaWiki extension is an implementation of the *Open Archive Initiative Protocol for Metadata Harvesting* (OAI-PMH)[54]. Its key concepts are *items*, *records* and *repositories*. In general, items are identifiers, while its representation in a certain format is called a record. We will refer to these item identifiers as *oaiIds*. In our context, items correspond to page names and records to pieces of XML providing information about that page as for example its content and last modification date. A *metadata prefix* is used to choose between representations. A *repository* is a server which is capable of processing the HTTP-requests defined in the specification. These request types are called *verbs*. In our case the most important verbs were `listIdentifiers` and `listRecords` which allow retrieval of sets of identifiers and records, respectively, which were modified after a given date. Additionally the verb `getRecord` is used to retrieve a single record for a specific oaiId.

The OAIRepository extension only stores the latest revision of a page and discards the others. Therefore requesting pages in a certain period of time will not return all modifications that occurred in that range, but only the pages whose last modification date lies within that range. Whenever a record is deleted, only the oaiId and the deletion date can be retrieved, not the page title. For that reason the oaiId must be associated with the extracted data in order to allow for clean deletions. The oaiId is a URI composed of the parts *oai:<domain>:<wikiName>:<pageId>* and have the specific forms

[54] `http://www.openarchives.org/OAI/openarchivesprotocol.html`

oai:en.wikipedia.org:enwiki:<pageId> and *oai:meta.wikimedia.org:metawiki:<pageId>* for EnWiki and MetaWiki, respectively. Access to Wikipedia's OAIRepository is not public and therefore requires credentials which were kindly provided to us by Brion Vibber[55].

5.4.2 English Wikipedia Extraction Workflow

The process of extracting RDF data from the wikitext of EnWiki pages works as follows: The EnWiki component polls EnWiki's OAIRepository at a fixed rate (about thirty seconds) for the most recent records. If the result of a poll reveals that there are still more records available, polling continues with a shorter delay (about five seconds) in order to catch up. This is repeated until no more records are available. A more sophisticated solution could consider records-per-time ratios in order to reduce the polling to a minimum.

A record either denotes a modification of a page's content (including its creation) or the deletion of a page. In the former case it is determined whether the page is an article or a template documentation. Articles are put into the spooler directory to be processed by the Extraction component. Template documentation pages are scanned for Infobox Annotations. If such an annotation is found, it is extracted and the Infobox Annotation DB is updated accordingly. The current implementation does not support reprocessing articles containing infoboxes, whose corresponding Infobox Annotations were changed.

In the event of deletions, we are only given the oaiId of the deleted page. Therefore we need to check the Triple Management DB as well as the Infobox Annotations DB whether triples or Infobox Annotations were associated with that identifier, and remove the corresponding data. The Extraction component polls the spooler directory on a regular basis and sends the pages it finds to the extractors which extract RDF data. This data is then written into the DataGraph and the Triple Management DB. In the latter DB this data is associated with the page's corresponding oaiId. A graphical overview of the workflow is shown in Figure 21.

[55] Lead developer of MediaWiki until late 2009.

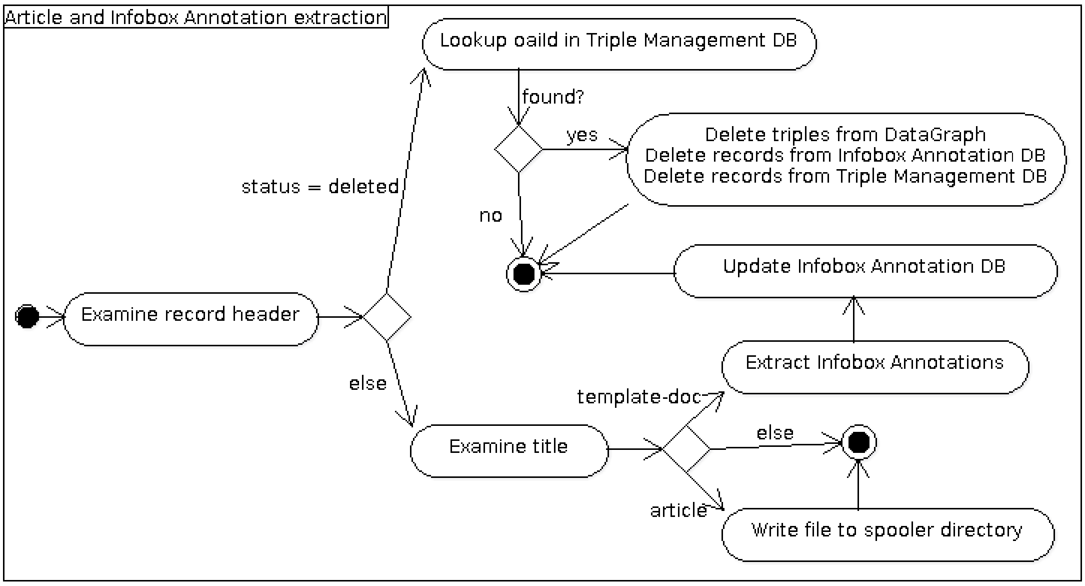

Article and Infobox Annotation extraction
Lookup oaiId in Triple Management DB
found?
yes
Delete triples from DataGraph
Delete records from Infobox Annotation DB
Delete records from Triple Management DB
status = deleted
no
Update Infobox Annotation DB
Examine record header
Extract Infobox Annotations
else
template-doc
Examine title
else
article
Write file to spooler directory

Figure 21: Article and Infobox Annotation extraction workflow

5.4.3 MetaWiki Extraction Workflow

The extraction workflow from MetaWiki is aimed at extracting the ontology Schema Definitions located on the subpages of *User:DBpedia-Bot/ontology*. In fact, these are the only pages on MetaWiki we are interested in. Whenever such a page is edited, any previously extracted data that is contained in the DataGraph and MetaGraph is discarded. Figure 22 depicts this workflow.

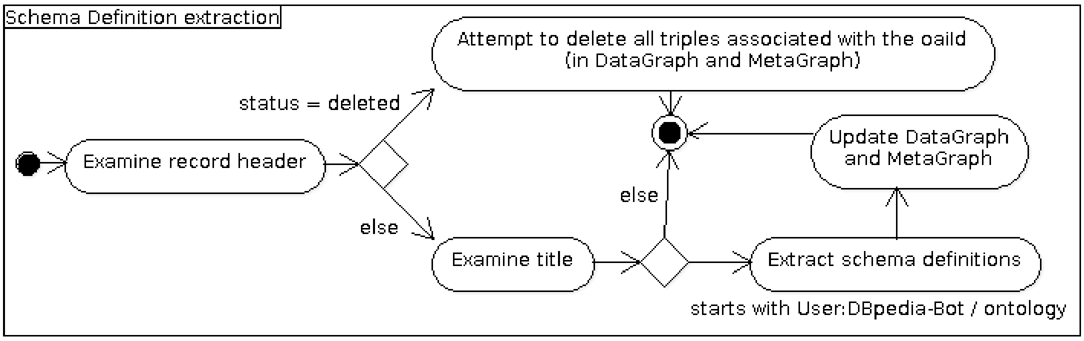

Schema Definition extraction
Attempt to delete all triples associated with the oaiId
(in DataGraph and MetaGraph)
status = deleted
Update DataGraph
and MetaGraph
Examine record header
else
else
Examine title
Extract schema definitions
starts with User:DBpedia-Bot / ontology

Figure 22: Schema Definition extraction workflow

5.5 Triple Management

In this section we discuss various strategies for updating a page's corresponding set of triples in the DataGraph after an edit. Large parts of this section (including its subsections) have been published in [20].

The main difference between the original and the live version of the framework is, that a strategy for updating the DBpedia dataset needs to be employed, as only the data corresponding to the articles' latest revisions should be retained. Such update strategy needs to respect two things: Firstly, the DBpedia live store is seeded with the latest DBpedia dataset, which is ver. 3.4 at the time of writing. The seeding is done in order to provide initial data about articles that have not been edited since the start of the Live Extraction process. Therefore the extraction takes place on an existing dataset, which not only contains data extracted from EnWiki, but also the third party datasets, that were outlined in Section 3.5. As the data of these third party datasets can neither be constructed from articles nor ontology definition pages, this data must remain unaffected by the Live Extraction process. However, when an article gets edited, its corresponding data in the seeding dataset must be updated. Since all data resides in the DataGraph, this becomes a rather complex task.

Secondly, the state of the extractors need to be taken into account. An extractor can be in one of the states *Active*, *Purge*, and *Keep*, which affects the generation and removal of triples as follows:

- *Active* The extractor is invoked on that page so that triples are generated.
- *Purge* The extractor is disabled and all triples previously generated by the extractor for a that page should be removed.
- *Keep* The extractor is disabled but previously generated triples for that page should be retained in the store.

The state *Keep* enables us to deactivate an extractor without losing its previously generated data (if it exists) for pages that are edited. This is useful in order to turn off extractors for maintenance. For instance when it is discovered that an extractor occasionally produces wrong data, it is neither desirable for the errors to spread further (*Active*), nor to lose all previously generated - presumably mostly correct - data (*Purge*).

Our initial strategy is summarized as follows: When the extraction framework recognizes an edit of an article for the first time, a clean up is performed, which removes all but the static facts from the seeding data set for the article's corresponding resource.

The new triples are then inserted into the DataGraph. Addionally, the following metadata is added to the MetaGraph: A resource representing the extraction process that generated these triples is created. The page, the extractor and the time of the extraction, as well as all generated triples[56] are then associated with that resource. On subsequent edits of the respective article, this metadata can be used to easily identify the triples needed to be updated.

As DBpedia consists of approximately 300 million facts, annotations would boost this value by a factor greater than four[57].

OpenLink provided us with a test server that was claimed to be capable of handling these amounts of data. Unfortunately, as the amount of data in the store grew, we soon realized that the update performance became so slow that edits on Wikipedia occurred more frequently than they could be processed. Even skipping articles that are edited repeatedly within a certain period in time (see Section 5.2) did not lead to a sufficient performance gain.

Obviously there was a misunderstanding: The server actually was capable of answering queries on these amounts of data in reasonable time, however, it was not capable of processing the updates fast enough. Before resorting to acquire better hardware, we considered alternative triple management approaches.

In order to ease the discussion, we explain the "simple annotation-based update strategy" before the generic one, although we implemented them in reverse order.

- *Clean-Up*: This is the strategy required for cleaning up the seed dataset, for which no explicit information about the triples' corresponding extraction processes exists. Although other strategies may introduce their own metadata for managing the triples, they must fall back on this strategy at least on an article's first recognized

[56] Actually their reifiers.

[57] Three triples in order to form the reifier (omitting the rdf:type-triple), and one triple for associating the reifier with the extraction process resource. Additionally, each extraction process requires three additional triples, describing its page, extractor, and time.

edit.

- *Simple annotation-based*: Each triple is directly annotated with the corresponding extractor, page and time.

- *Generic annotation-based*: This is our initial strategy. Each extraction process, identified by the page, extractor, and time, is represented using a distinct resource. The corresponding triples are associated with it. This is our initial implementation.

- *Resource-Specific Graphs*: A strategy where all triples generated from a specific page-extractor pair reside in their own graph.

- *RDB-Assisted*: In this strategy, metadata about triples is stored in a relational database, rather than the MetaGraph.

Finally we present an evaluation of some of the implemented strategies.

5.5.1 Clean-Up Strategy

The problem we are facing with the seed-dataset is: Whenever a page is edited, we need to update its corresponding triples in the DataGraph according to the extractors' states. In order to do so, we must be able to determine what the corresponding triples are, and what extractor was used to generate them. However, in regard to the seed dataset, we lack metadata that would enable us to easily achieve that.

Fortunately this problem can be solved by describing an extractor's output with patterns. Based on the states of the extractors and its pattern definitions, a clean up query can be generated. Listing 9 contains an example of such pattern definition, and Listing 10 shows an excerpt of the clean up query generated from it.

```
’HomepageExtractor’ => array(
  PRODUCES => array(
        // Pattern matching triples whose predicate equals foaf:homepage
        array(’type’=>EXACT, ’s’=>’’, ’p’=>FOAF_HOMEPAGE, ’o’=>’’)
        )),
’AlwaysFilterExtractor’ => array(
  PRODUCES => array(
        // Pattern matching triples whose predicate matches rdf:type and
        // whose objects starts with the yago namespace
        array(’type’=>STARTSWITH,
                'S'=>'', 'p'=>RDF_TYPE, 'o'=>DB_YAGO_NS, 'pexact'=>true),
        )),
```

Listing 9: Patterns describing extractors' outputs

```
DELETE
FROM <http://dbpedia.org>
{ <http://dbpedia.org/resource/London> ?p ?o . }
{ <http://dbpedia.org/resource/London> ?p ?o .
  # Dynamically generated filters based on extractors in
  # active and purge state
  FILTER(?p = foaf:homepage ||
         # more conditions for other extractors
  ) .
  # Static filters preventing deletion of the static DBpedia part
  FILTER((?p != <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ||
         !(REGEX(?o, '^http://dbpedia.org/class/yago/')) &&
         # more conditions for the static part
  ) .
}
```

Listing 10: Clean up query generated based on extractors' output patterns

As can be seen in the example, the query becomes very complex as every pattern definition of an extractor results in more filter criterias being added. Therefore we seek to reduce the complexity of updates at least for subsequent edits of an article.

We want to mention, that the same patterns used for describing an extractor's output can also be used for validation. This is useful for debugging, as it allows for identifying cases where an extractor's output is not matched by its corresponding patterns.

5.5.2 Simple Annotation-Based Update Strategy

Here we describe an update strategy based on annotating triples directly with its generating extractor and corresponding page as shown in Figure 23. These annotated triples are stored in the MetaGraph, whereas the actual triples reside as usual in the DataGraph.

This approach has the constraint that at any point in time, every triple in the DBpedia graph may only be generated by a single extraction process. Otherwise a triple would end up having multiple *dbpmeta:usedExtractor* and *dbpmeta:sourceResource* statements and it would be impossible to uniquely relate it back to its generating extraction processes. In other words: This approach only supports relations between processes and triples of

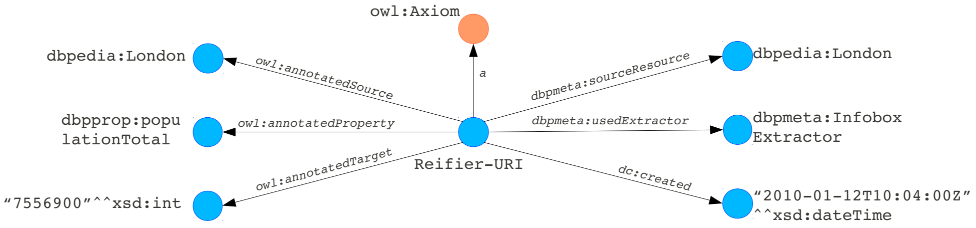


Figure 23: Direct annotation of a triple

cardinality 1:N.

With these annotations present, the update procedure becomes fairly simple: Upon an edit of a page, a delete query is built which removes all those triples from the DataGraph, that have a reifier with respective annotations in the MetaGraph. Immediatly afterwards, the corresponding triples in the MetaGraph are also removed. Finally, the newly extracted triples are inserted into the DataGraph, and their new annotations are added to the MetaGraph.

5.5.3 Generic Annotation-Based Update Strategy

In this section we discuss a strategy for managing triples, which – in contrast to the strategy in the previous section – supports relations of arbitrary cardinality between triples and their generating extraction processes. The basic view point is as follows: Every triple is generated by one or more *extraction processes*. Such process is carried out on a certain *date*, with an *extractor*, which takes a *page* as its input. These facts can be represented in RDF as illustrated in Figure 24.

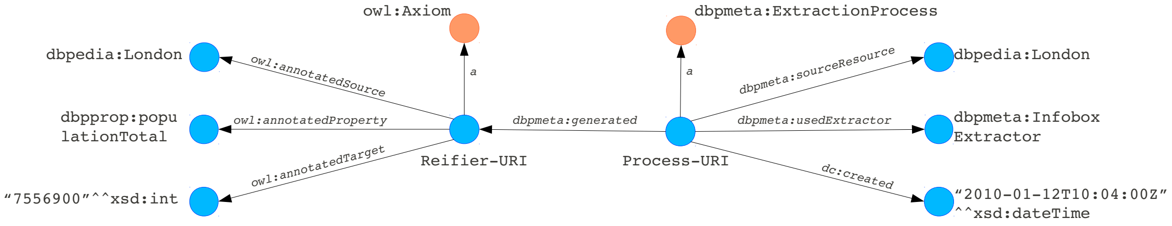


Figure 24: Relating a triple to its generating extraction process

In this example, we state that the triple corresponding to "London has 7556900 inhabitants" was generated by an extraction process, performed on the *12th January 2010*, from the resource *dbp:London*, using the InfoboxExtractor. We create URIs for reifiers based on MD5 hashes from the subject, predicate, and object of the triples. The process URI is created similarly; except that its MD5 hash is based on the values of the *dbpmeta:usedExtractor* and *dbpmeta:sourceResource* predicates.

This strategy was implemented in order to support the global URI scheme for class expressions, as explained in Section 4.2.2.

The update strategy works as follows: Whenever an extraction process p takes place, we are left with two sets of triples: The (possibly empty) set of triples already residing in the DataGraph that was previously generated by an equivalent process, and the one that was recently generated and should replace the old one. We call these sets O and N, respectively. Based on these sets, we derive two further sets: The set of removal candidates $R = O/N$ and the set of insertion candidates $I = N/O$. Generally any triple in R loses its relation to p. Furthermore, the triple itself needs to be removed from the graph when it loses its last relation to a process. As a consequence, a triple that is not related to any process in the first place is not deleted by this method. This property is exploited for preventing damage to the static DBpedia part in the case that a process generates triples that are already members of the static part: Before a triple is inserted and related to a process, it is first checked whether the same triple without any relations to processes already exists in the graph. If that is the case, insertion of that triple is skipped.

The update strategy becomes very heavyweight because of the many database queries that are involved: First the whole set O needs to be retrieved, followed by determining process relations of triples in R and I, and additionally pre-existence of triples in I.

Although our implementation worked correctly, the performance was rather bad. The query execution times jumped at random times from about 1 to 300+. seconds. This rendered this approach useless for us as this meant that these queries would not only delay processing of individual pages, but they would also very likely slow down all queries against the database in general. Our attempts in rewriting and optimizing these queries did not solve that problem. After too much time went into this, we abandoned this strategy.

5.5.4 Resource-Specific Graphs

Here we describe an update strategy based on partitioning the DBpedia live dataset into multiple graphs, whereas each graph corresponds to a single extraction process.

The previously mentioned attempts have the disadvantage of either introducing a high overhead with respect to the amount of triples needed to store meta data or being very complex. A different approach is to put each set of triples generated by an extractor from an article into its own graph. For instance a URI containing a hash of the extractor and page name could serve as the graph name. The update process then becomes greatly simplified as upon an edit, it is only necessary to clear the corresponding graph and insert the new triples. This approach requires splitting the seeding DBpedia dataset into separate graphs from the beginning. As the DBpedia dataset v3.4 comes in separate files for each extractor, the subjects of the triples in these files determine the target graph. The downside of this approach is, that the data no longer resides in a single graph. Therefore it is not possible to specify the dataset in the SPARQL FROM clause. Instead, a FILTER over the graphs is required as shown in Listing 11.

```
SELECT ?s ?p ?o
{ GRAPH ?g {?s ?p ?o} .
  FILTER(REGEX(?g, ’\^http://dbpedia.org/’)) .
}
```

Listing 11: Selecting triples across multiple graphs

In fact, the breakage of the FROM clause was one of the main reasons we decided to avoid this strategy. DBpedia Live already comprises two graphs (the DataGraph and the MetaGraph), and the server where it gets deployed may contain more graphs, therefore it is desired to be able to chose graphs in the usual way.

5.5.5 RDB-Assisted Update Strategy

The third approach we evaluated and implemented is to store RDF statements in a relational database (RDB) in addition. This approach is motivated by the observation that most changes made to Wikipedia articles only cause small changes in the corresponding RDF data. Therefore, the idea is to have a method for quickly retrieving the set of triples previously generated for an article, comparing it to the new set of triples and only performing the necessary updates.

For selection of resources which have to be updated after a periodically finished Wikipedia extraction process, we firstly created an RDB table as illustrated in Figure 25. Whenever a Wikipedia page is edited, the extraction method generates a JSON object holding information about each extractor and its generated triples. After serialization of such an object, it will be stored in combination with the corresponding page identifier. In case a record with the same page identifier already exists in this table, this old JSON object and the new one are being compared. The results of this comparison are two disjoint sets of triples which are used on the one hand for adding statements to the DBpedia RDF graph and on the other hand for removing statements from this graph. Therefore the update procedure becomes straight forward:

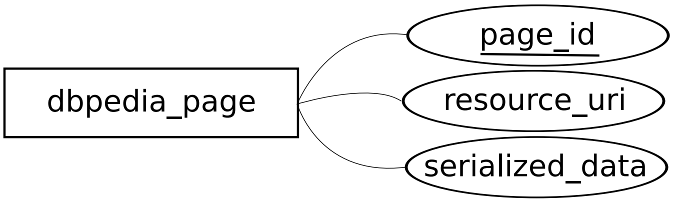


Figure 25: Definition of the RDB table

With this strategy, once the initial clean up for a page has been performed, all further modifications to that page only trigger a simple update process. This update process no longer involves complex SPARQL filters, instead it can modify the affected triples directly.

```
SELECT data FROM dbpedia_page
WHERE page_id = "http://dbpedia.org/resource/London";

INSERT INTO dbpedia_page(page_id, data)
VALUES("http://dbpedia.org/resource/London", <JSON-Object>);

UPDATE dbpedia_page SET data = <JSON-Object> WHERE page_id = <pageId>
```

Listing 12: SQL Statements for fetching data for a resource

```
Delete From <http://dbpedia.org> { ... concrete triples ... }
Insert Into <http://dbpedia.org> { ... concrete triples ... }
```

Listing 13: Simple SPARQL Delete and insert queries

Algorithm 1 Algorithm of the RDB assisted update process

```
//**The data to be put into the store is included in the extractionResult**
//**object**
$pageId \leftarrow extractionResult[pageId]$
$resourceUri \leftarrow extractionResult[resourceUri]$
$newTriples \leftarrow extractionResult[triples]$

//**Attempt to retrieve previously inserted data for the pageId**
$jsonObject \leftarrow fetchFromSQLDB(pageId)$
**if** $jsonObject \neq NULL$ **then**
    $oldTriples \leftarrow extractTriples(jsonObject)$
    $insertSet \leftarrow newTriples - oldTriples$
    $removeSet \leftarrow oldTriples - newTriples$
    $removeTriplesFromRDFStore(removeSet)$
    $addTriplesToRDFStore(insertSet)$
**else**
    $cleanUpRDFStore(pageId)$
    $insertIntoRDFStore(newTriples)$
**end if**

$jsonObject \leftarrow generateJSONObject(pageId, resourceUri, newTriples)$
$putIntoSQLDB(jsonObject)$
```

5.5.6 Evaluation

We did a small evaluation by comparing the RDB assisted update process to a simplified version of the *Clean Up* strategy. This simplified version deletes triples with a certain subject using the query of Listing 14 instead of the one of Listing 10. The difference is only that the complex filter patterns were omitted.

```
FROM <http://dbpedia.org>
{ <http://dbpedia.org/resource/London> ?p ?o . }
{ <http://dbpedia.org/resource/London> ?p ?o . }
```

Listing 14: Example of the simplified delete query

The benchmark simulates edits of articles and was set up as follows. 5000 distinct resources were picked at random from the DBpedia dataset. For each resource two sets O and N were created by randomly picking $p\%$ of the triples whose subject starts with the resource. The sets O and N represent the sets of triples corresponding to an article prior and posterior to an edit, respectively. A run of the benchmark first clears the target graph and `dbpedia_page` table. Then each resource's O-set is inserted into the store.

Finally the time to update the old sets of triples to the new ones using either the simplified specialized update strategy or the RDB assisted one[58] is measured. Additionally the total number of triples that were removed ($O - N$), added ($N - O$) and retained ($N \cap O$) were counted. Three runs were performed with $p = 0.9$, $p = 0.8$, and $p = 0.5$ meaning that the simulated edits changed 10%, 20% and 50% of the triples, respectively. We assume that the actual ratio of triples updated by the Live Extraction process in the event of repeated edits of articles is between 10 and 20 percent. However the exact value has not been determined yet. The benchmark was run on a machine with a two core 1,2GHz Celeron CPU and 2GB RAM. The triple store used was "Virtuoso Open-Source Edition 6.1.1" in its default configuration with four indices GS, SP, POGS, and OP.

p	**Added**	**Removed**	**Retained**	**Strategy**	**Time taken (sec)**
0.5	124924	124937	123319	SQL RDF	240 200
0.8	79605	79710	318149	SQL RDF	200 250
0.9	44629	44554	402748	SQL RDF	170 300

Table 8: Benchmark results

In Table 8 the value *SQL* indicates the RDB assisted approach, and *RDF* the specialized one. As can be seen from the table, the former approach - which reduces the updates to the triple store to a minimum - performs better than specialized version when there is sufficient overlap between O and N ($p = 0.8$ and $p = 0.9$) On the other hand, the smaller the overlaps the more the RDB becomes a bottleneck ($p = 0.5$). This is expected as in the worst case there is no overlap between O and N. In this situation the specialized approach would delete and reinsert triples directly. The RDB assisted approach would ultimately do the same; however with the overhead of additionally reading from and writing to the `dbpedia_page` table.

[58] As this approach involves a JSON object holding information about each extractor, the generation of the sets O and N was related to a dummy extractor

5.5.7 Conclusion

Since the RDB assisted update strategy turned out to be the fastest one, we used it in the version of the DBpedia Live Extraction Framework, that is now deployed at OpenLink[59].

However, even with this strategy we did not reach the necessary performance. The problem was finally solved, when OpenLink provided us better hardware. In fact, the new server is now fast enough, that it now seems possible to re-extract RDF data from all articles containing infoboxes whose Infobox Annotations were edited.

5.6 Contributed Extractors

In this section we summarize the extractors that were added to the framework.

TBoxExtractor This extractor is responsible for generating RDF from the Schema Definition templates. It proceeds as described in Section 4.2.2. It is part of the Java-Component.

InfoboxAnnotationExtractor The extractor for processing the Infobox Annotations, which were described in Section 4.2.3. The extracted data is written into the Infobox Annotation DB. The extractor is part of the EnWiki component.

LiveMappingBasedInfoboxExtractor This component is a rewrite of the original MappingBasedExtractor. When invoked on a page it proceeds as follows: First the page is scanned for template references which do not appear within other template references. Each of these template references is then decomposed into its name and the key-value pairs used as the arguments. Afterwards the Infobox Annotation DB is consulted in order to determine if any related classes or parameter mappings are defined. If such mappings exist, the infobox arguments are extracted accordingly. Otherwise the generic extraction, described in Section 3.3, is used.

LiveAbstractExtractor The original AbstractExtractor extracts abstracts from articles and connects them to the corresponding DBpedia resource via the predicate *dbpedia-*

[59] http://dbpedia-live.openlinksw.com/sparql

owl:abstract. However, the wikitext of abstracts often contains helper templates such as for phonetic spelling. In order to produce quality abstracts, these need to be resolved. The original extractor does that using the API of a local MediaWiki instance with those templates loaded. The API of EnWiki cannot be used because the load imposed on the server would be too high.

In order to reuse this extractor, the Live Extraction Framework would have to additionally keep such MediaWiki instance in sync. Although technically this is not very difficult, there were two reasons against it:

- We were already struggling with performance issues, and keeping another wiki - which implies another database - in sync would have meant even more overhead.
- The Live Extraction Framework was going to be deployed at one of OpenLink's servers. As this company develops their own database (Virtuoso), we wanted to avoid potential conflicts by requiring them to install a MySQL database, as up to now MediaWiki provides no Virtuoso backend.

While searching for an alternative we found the ActiveAbstract MediaWiki extension[24]. The core of this extension is made of a set of regular expressions which essentially remove the MediaWiki markup such as template references or smart links. Although parsing is very fast, as it only takes a few milliseconds, unfortunately the quality of the abstracts is sometimes rather low. Therefore we decided not to overwrite the existing abstracts, and introduce the new property `dbpedia-owl:abstract_live` instead.

6 Related work

The related work is structured into the sections *Research*, *Tools*, and *Applications*.

6.1 Research

Wikipedia has been and still is a target for a vast amount of research. An incomplete list is maintained by Wikipedia itself[60]. Another now slightly outdated list containing more than a hundred resources related to how the semantification of Wikipedia aids the Semantic Web is compiled at Michael Bergman's blog[61].

DBpedia is not the only project focusing on RDF information extraction from Wikipedia articles. For instance [12] uses similar extraction techniques as DBpedia such as extracting information from infoboxes and links. Although DBpedia already existed at this time, the authors wanted to make their own experiences with Wikipedia extraction. In contrast to DBpedia, they also experimented with information extraction from free text using list-lookup extraction, fillers (essentially regular expression patterns), and spatial/proximity analysis (such as if in a sentence two values are found out to correspond to a year and a month, a third value will probably correspond to a day).

More sophisticated techniques from the field of machine learning are used in KYLIN[26]. One of this project's goals is to extract key-value pairs from sentences summarizing an article's properties. This information could be used for things such as: consistency checking between texts and infoboxes, filling out missing values in exisiting infoboxes, and even suggesting new infoboxes.

KYLIN proceeds in first training *document classifiers* and *sentence classifiers*, whereas the latter are trained in respect to a certain document class. Finally, for each resulting sentence class *extractors* are trained for matching the desired data. The training is done by first assigning feature vectors to the documents and sentences, and then applying learning algorithms such as Support Vector Machines and Conditional Random fields.

In a further step, the developers of KYLIN created the KYLIN Ontology Generator (KOG)[27]. This system is capable of automatically deriving subsumption hierachies

[60] `http://en.wikipedia.org/wiki/Wikipedia:Academic_studies_of_wikipedia`

[61] `http://www.mkbergman.com/417/99-wikipedia-sources-aiding-the-semantic-web/`

between infobox-based classes. Again the creators use machine learning techniques, specifically Support Vector Machines and Markov Logic Networks.

DBpedia could greatly benefit from incorporating such automatic ontology generation techniques: The output of these systems could be presented to people editing the Schema Definitions and Infobox Annotations. These people can then review, adapt, and accept (or reject) those suggestions.

6.2 Tools

In the following, two other projects, which serve a similar purpose as DBpedia are explained.

Semantic MediaWiki Semantic MediaWiki (SMW)[62] is a MediaWiki extension which directly integrates Semantic Web technologies. It allows for ontology engineering by introducing - among other things - special namespaces for properties, datatypes and classes, a syntax extension for typed links, inline queries for autmatic generation of lists, form-based template editing and RDF exports. There already exist many extensions which are based on SMW core functionality. SMW and most, if not all, extensions can be downloaded and used free of charge.

The *typed link* is the most fundamental feature. Its syntax is *[[<property-name>::<value>]]*. The double colon indicates a typed link between the page the link appears on and the given value. The interpretation of the value depends on the type-definition given on the property's page. If no such definition exits, the value is assumed to be the name of page. Therefore if a property such as *population* was undefined, any value assigned to it would be treated as reference to a page instead of a number. The type of a property is defined by adding the snippet *[[has type::<type>]]* on its page, where *has type* is a built-in property.

It isn't hard to see that these typed links can be mapped to RDF. An RDF export of a page can be obtained through the page *Special:ExportRDF*.

Because SMW covers similar grounds as DBpedia, it is very likely to supersede it once it becomes deployed on Wikipedia. However, currently their website states "The

[62] http://semantic-mediawiki.org/wiki/Semantic_MediaWiki

Wikimedia Foundation has general plans to do its own performance testing, and code review, of SMW, at an unknown date in the future."[63]

Freebase Freebase is proprietary platform for collaborative knowledge acquisition. While SMW provides features for annotating free text, Freebase is all about structured data. Each topic is given it's own site, where information about it is kept in the form of key-value pairs, which can be viewed and edited. Users can can further create *Domains* which are collections of *Types* and *Properties*. A domain may become adapted and improved by a larger community when it considers it useful. Types indicate *is-a* relationships while properties indicate *has-a* relationships.

6.3 Applications

A number of applications have been built that use DBpedia as their datasource (or at least as one of their datasources). An incomplete list is maintained at the DBpedia website itself[64]. These applications are currently based on the original dumb-based DBpedia dataset. However, in the future some of them may become adpated to the live one.

DBpedia Facetted Browser The browser allows facetted navigation of the DBpedia dataset. Facetted navigation essentially means browsing from set of things to sets of things. The initial set contains every DBpedia item. After each navigation step, a user is presented a summary of the features about the items in the current set. A user can then refine such a set by successively setting constraints on these features. For example, a user could first choose to only select people. In subsequent steps he or she can refine the set further to ones born in a specific country after a certain year.

DBedia Mobile "is a location-centric DBpedia client application for mobile devices"[65]. This application augments maps (e.g. provided from Google Maps and OpenStreetMap) with information (e.g. labels and icons) from DBpedia and its interlinked datasets. Based

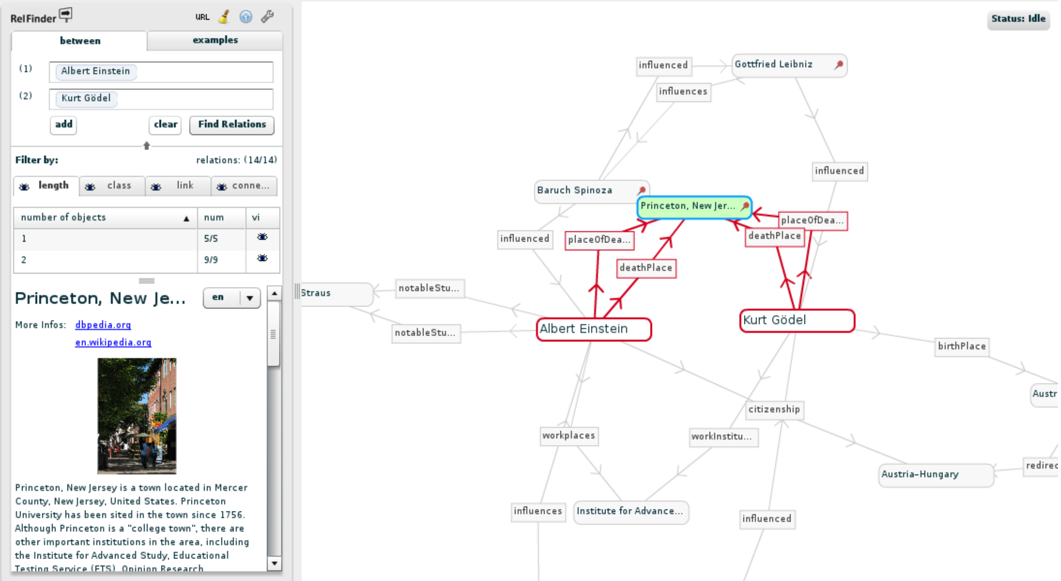
[63] `http://semantic-mediawiki.org/w/index.php?title=FAQ&oldid=2733` (retrieved 31st May 2010)

[64] `http://wiki.dbpedia.org/Applications`

[65] `http://wiki.dbpedia.org/DBpediaMobile`

on the GPS position of the mobile device, it can provide this information for nearby locations.

DBpedia Relationship Finder The DBpedia Relationship Finder[66] is a browser-based application for searching connections between two DBpedia URIs and displaying them graphically. This gives the opportunity to serendipitous discoveries. Technically the DBpedia RDF graph is searched for all paths connecting these resources. A screenshot is shown in Figure 26.



(1) Albert Einstein

(2) Kurt Gödel

Filter by: relations: (14/14)

number of objects ▲	num	vi
1	5/5	👁
2	9/9	👁

Princeton, New Je...

More Infos: dbpedia.org en.wikipedia.org

Princeton, New Jersey is a town located in Mercer County, New Jersey, United States. Princeton University has been sited in the town since 1756. Although Princeton is a "college town", there are other important institutions in the area, including the Institute for Advanced Study, Educational Testing Service (ETS), Opinion Research

Status: Idle

Figure 26: The DBpedia Relationship Finder

[66] http://relfinder.dbpedia.org

7 Conclusions and Future Work

The three major contributions made by this thesis are the Infobox Annotations, the Ontology Schema Definitions and the DBpedia Live Extraction Framework.

We introduced templates for defining the schema of the ontology (consisting of classes, object and datatype properties) and templates for defining the mappings between infobox parameters and ontology properties. These templates were intendend to provide a wiki based solution for engineering the DBpedia Infobox Ontology. We were able to convince the MetaWiki community to allow us host the schema definitions there, but we failed to convince the english Wikipedia community of permitting us to host our Infobox Annotations. From the discussion that arose we got more insight into community processes. Specifically, we learnt that we first need to show more benefits and solve some of the currently unaddressed problems of our approach, before we can expect community acceptance.

In a reaction to this, the FU Berlin recently decided to set up their own wiki, in order to host the Schema Definition- and Infobox Annotation-templates, developed in this thesis, there. The Live Extraction Framework was configured accordingly.

This wiki hosts tools for: validating the templates (syntactically and semantically), performing test extractions against Infobox Annotations, and displaying an overview of ontology. The *DBpedia Ontology Mapper* tool, which is currently in development, will provide a web-based GUI for editing the Schema Defintions and Infobox Annotations.

The DBpedia Live Extraction Framework was enhanced to be capable of processing edits of EnWiki articles, Schema Definitions, and Infobox Annotations in near realtime. During the development of the framework, several strategies for updating the DBpedia Live dataset have been evaluated.

Currently the extraction framework is being completely redesined from scratch in Scala[67], which will hopefully get rid of many of the struggles we had with the mixed PHP-Java code base.

The immediate next step is restoring the status quo of the Live Extraction Framework in Scala. In subsequent steps, strategies for lowering the barrier for making constributions

[67] http://www.scala-lang.org

to DBpedia further, will be evaluated. For instance, approaches based on presenting automatically generated suggestions to users seem promising. Ultimately, the goal is to bring some of the results back to Wikipedia in order to improve its data (even more).

8 Appendix

8.1 Source code

The source code that was written in the course of this thesis is available in the DBpedia SVN repository at

```
http://dbpedia.svn.sourceforge.net/svnroot/dbpedia/extraction
```

The author of this thesis made changes to the code base under the name *Aklakan.* Revision 3440 represents the state of the repository at the time of submission.
The wikitext of the templates for the Infobox Annotations and the Schema Definitions is located at

```
http://dbpedia.svn.sourceforge.net/viewvc/dbpedia/extraction/MediawikiPages
```

List of Figures

Listings

List of Tables

References

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2008.

[2] S. Auer, R. Doehring, and S. Dietzold. Less- template-based syndication and presentation of linked data, 2010.

[3] S. Auer and J. Lehmann. What have innsbruck and leipzig in common? extracting semantics from wiki content. In E. Franconi, M. Kifer, and W. May, editors, *ESWC 2007: Proceedings of the 4th European Semantic Web Conference, Innsbruck, Austria*, volume 4519 of *Lecture Notes in Computer Science*, pages 503–517, Berlin, 2007. Springer.

[4] T. Berners-Lee. Linked data design issues. W3C design issue document, June 2009.

[5] T. Berners-Lee, R. Fielding, and L. Masinter. Rfc 3986: Uniform resource identifier (uri): Generic syntax. `http://www.ietf.org/rfc/rfc3986.txt`, 2005.

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[7] C. Bizer and A. Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[8] W. O. W. Group. Owl 2 web ontology language document overview. `http://www.w3.org/TR/owl2-overview`, October 2009.

[9] S. Hellmann, C. Stadler, J. Lehmann, and S. Auer. Dbpedia live extraction. In *Proc. of 8th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, volume 5871 of *Lecture Notes in Computer Science*, pages 1209–1223, 2009.

[10] M. Hepp, K. Siorpaes, and D. Bachlechner. Harvesting wiki consensus: Using wikipedia entries as vocabulary for knowledge management. *IEEE Internet Computing*, 11(5):54–65, 2007.

[11] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. H. Wang. The manchester owl syntax. In B. C. Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *Proceedings of OWL: Experiences and Directions (OWLED'06)*, Athens, Georgia, USA,, 2006.

[12] J. Isbell and M. H. Butler. Extracting and re-using structured data from wikis. Technical Report HPL-2007-182, Hewlett-Packard, 2007.

[13] B. M. G. Klyne. Resource description framework (rdf): Concepts and abstract syntax. W3C Recommendation, February 2004.

[14] A. Miles, B. Matthews, M. Wilson, and D. Brickley. SKOS core: Simple Knowledge Organisation for the Web. In *Proceedings of the 2005 international conference on Dublin Core and metadata applications*, pages 1–9, Madrid, Spanien, 2005. Dublin Core Metadata Initiative.

[15] E. Pietriga, C. Bizer, D. Karger, and R. Lee. Fresnel: A Browser-Independent presentation vocabulary for RDF. In *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 158–171. Springer-Verlag, 2006.

[16] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

[17] L. Sauermann and R. Cyganiak. Cool uris for the semantic web. w3c interest group note 03. `http://www.w3.org/TR/cooluris/`, December 2008.

[18] A. Seaborne and G. Manjunath. Sparql/update - a language for updating rdf graphs, 2008.

[19] Several. Linking Open Data. `http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`, August 2009.

[20] C. Stadler, M. Martin, J. Lehmann, and S. Hellmann. Update strategies for dbpedia live, 2010.

[21] R. Studer, R. Benjamins, and D. Fensel. Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering*, 25:161–197, 1998.

[22] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 697–706, New York, NY, USA, 2007. ACM Press.

[23] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Journal of Web Semantics*, 6(3):203–217, 2008.

[24] B. Vibber. ActiveAbstract MediaWiki extension. `http://www.mediawiki.org/wiki/Extension:ActiveAbstract`.

[25] W3C. W3c semantic web activity. `http://www.w3.org/2001/sw/`.

[26] F. Wu and D. S. Weld. Autonomously semantifying wikipedia. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 41–50, New York, NY, USA, 2007. ACM.

[27] F. Wu and D. S. Weld. Automatically refining the wikipedia infobox ontology. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 635–644, New York, NY, USA, 2008. ACM.

Declaration

This diploma thesis is the result of my own work. Material from the published or unpublished work of others, which is referred to in the thesis, is credited to the author in the text. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this thesis and the degree examination as a whole.

Leipzig, 31. May 2010 Signature