



Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZAI-MS-C-2012-05

Masterarbeit

im Studiengang "Angewandte Informatik"

Process-based Data Extraction from Web Sources

Benjamin Dake

Arbeitsgruppe für
Datenbanken & Informationssysteme

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

25. September 2012

Georg-August-Universität Göttingen
Zentrum für Informatik

Goldschmidtstraße 7
37077 Göttingen
Germany

Tel. +49 (5 51) 39-17 42010

Fax +49 (5 51) 39-1 44 15

Email office@informatik.uni-goettingen.de

WWW www.informatik.uni-goettingen.de

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 25. September 2012

Master's Thesis

**Process-based Data Extraction from
Web Sources**

Benjamin Dake

September 25, 2012

Supervised by Prof. Dr. Wolfgang May
Databases and Information Systems Group
Georg-August-Universität Göttingen

Abstract

The World Wide Web holds a large amount of information in unstructured form. The Web Data Extraction is concerned with the extraction of such data into a structured form that is proper for a further processing. This Thesis develops an extendible process for the extraction of linked geographic data from Wikipedia. It uses a classifier in order to map a Web Page into a category and then applies an appropriate wrapper for the data extraction.

Contents

1	Introduction	1
2	Foundations	3
2.1	Wikipedia	3
2.2	XML	7
2.3	RDF	9
2.4	Natural Language Processing	9
3	Design	12
3.1	Classification	13
3.1.1	Category Analysis	14
3.1.2	First Sentence Analysis	15
3.1.3	Title Analysis	18
3.1.4	Class Selection	18
3.1.5	Administrative Divisions	20
3.2	Data Extraction	21
3.2.1	Data Overview	22
3.2.2	Result Format	22
3.2.3	Extraction Algorithm	24
3.3	Outgoing Link Extraction	26
3.4	Link Analysis	26
4	Implementation	28
4.1	System Architecture	28
4.1.1	MARS Framework	28
4.1.2	Components	30

Contents

4.1.3	Data Extraction Service	31
4.1.4	MARS Data Extraction Process	34
4.2	Service Implementation	37
4.2.1	Used Frameworks and Java Libraries	37
4.2.2	Class Architecture	38
4.2.3	Class Interaction	40
4.2.4	Service Configuration	41
5	Evaluation	45
5.1	Result Overview	45
5.2	Classifier Accuracy	46
5.3	Reference Analysis	47
5.4	Runtime	47
6	Related Work	48
7	Conclusion	50
	Bibliography	51

List of Figures

1.1	Outline of the Data Extraction Process	1
2.1	Wikipedia Page Structure	4
2.2	Category Hierarchy Sample	5
2.3	Infobox Template Instantiation	6
2.4	Wikipedia Redirect Notice	6
2.5	XML Example Snippet	7
2.6	Penn Treebank Part-Of-Speech Tags	10
2.7	Penn Treebank Chunk Tags	11
3.1	Data Extraction Process Flow Chart	12
3.2	Hierarchy of the "Cities in Lower Saxony" Category	14
3.3	Category Matching Patterns	15
3.4	First Sentence Matching Patterns	18
3.5	Classification Rating of the Göttingen Article	19
3.6	Infobox of the River Rhine Article	21
3.7	Extracted Class Properties	22
3.8	RDF Graph of Result Data	23
3.9	RDF Triples of Result Data	23
3.10	Infobox Instantiation of the Göttingen Article	24
3.11	Extracted Raw Data of the Göttingen Article	25
3.12	Final Result Triples of the Göttingen Article	26
4.1	MARS CCS Process Example	29
4.2	MARS Data Extraction Process Architecture	30
4.3	Query Broker LSR Entry	32
4.4	CCS Data Extraction Control Process	36

List of Figures

4.5	CCS Reference Analysis Process	37
4.6	UML Class Diagram	38
4.7	Sequence Diagram	41
4.8	Classifier and Data Extractor Configuration File	43
4.9	Reference Analyzer Configuration File	44
5.1	Data Extraction Process Test Results	45
5.2	Classifier Test Results	46

1 Introduction

The World Wide Web enjoys a great popularity and has grown to a major source of information. Unfortunately, a large amount of the information is only available in unstructured or semi-structured form, and thus cannot be used by applications that require information in a specific structured format. The Web Data Extraction is concerned with the transformation of available web data into a structured format that is proper for the further processing.

An example of a Web Source is Wikipedia. Considering the large amount of information that is present in Wikipedia, it is an attractive source for the data extraction. Several attempts were made to extract data from Wikipedia. The probably most famous project is DBpedia [4]. While it focuses on the extraction of infobox data from Wikipedia in a large scale, it lacks in the extraction of annotated relations between entities from the article text.

The goal of this Thesis is to extract geographical data from Wikipedia. An advantage over DBpedia is the extension of the extracted data by annotated relations. As part of the data extraction, it classifies articles into specific geographical classes. For this, a process that crawls through Wikipedia and extracts data has been designed. As shown in Figure 1.1, it consists of three basic steps. At first, a classifier associates an article with a classes.

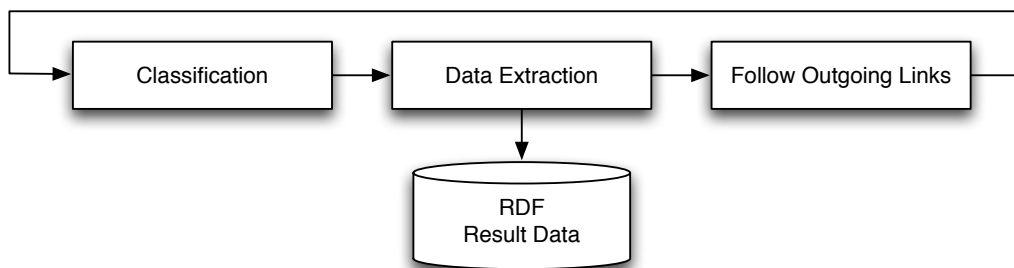


Figure 1.1: Outline of the Data Extraction Process

Depending on the classification, a wrapper that extracts information from the infobox is

then applied to the article page. In doing so, it not blindly extracts all infobox data, but it focuses on the extraction of specific properties and the mapping to a predefined ontology. The last step is to follow the outgoing links to other articles. In addition to the just mentioned data extraction process, the links between Wikipedia articles are analysed in order to extend the result data with additional annotated relations. The result is a set of linked data in RDF format.

Structure of the Thesis

Chapter 2 introduces the necessary foundations that are required for the understanding of the Thesis. Chapter 3 describes the design of the data extraction process. An overview of the implementation is presented in the subsequent Chapter 4. Chapter 5 evaluates the quality of the implemented approach. An overview of related work is given in Chapter 6. The Thesis is concluded with a short summary and outlook.

2 Foundations

This chapter introduces some basics that are required for the further understanding of the Thesis. At first, a brief overview of Wikipedia is given, followed by an introduction of the Extensible Markup Language (XML), the Resource Description Framework (RDF) and some Natural Language Processing (NLP) techniques.

2.1 Wikipedia

Wikipedia [22] is a free-content online encyclopaedia which was founded in 2001. Up to now, volunteers have written about 4 million articles for the English Wikipedia. Besides the English Wikipedia, there also exist editions in 285 other languages. Wikipedia articles are written using a special markup language, the so-called Wiki markup or Wikitext.

Article Structure

The parts of a Wikipedia article page that are relevant for this Thesis are labeled in Figure 2.1. Namely, these are

- **Title.** A unique article title.
- **Lead Section.** The lead section contains a brief summary of the article and serves as an introduction for the reader.
- **Article Text.** A Wikipedia article is a mixture of text, graphics and tables.
- **Infobox.** An optional infobox summarizes some characteristic information.
- **Categories.** An article belongs to categories that group associated articles.

2 Foundations

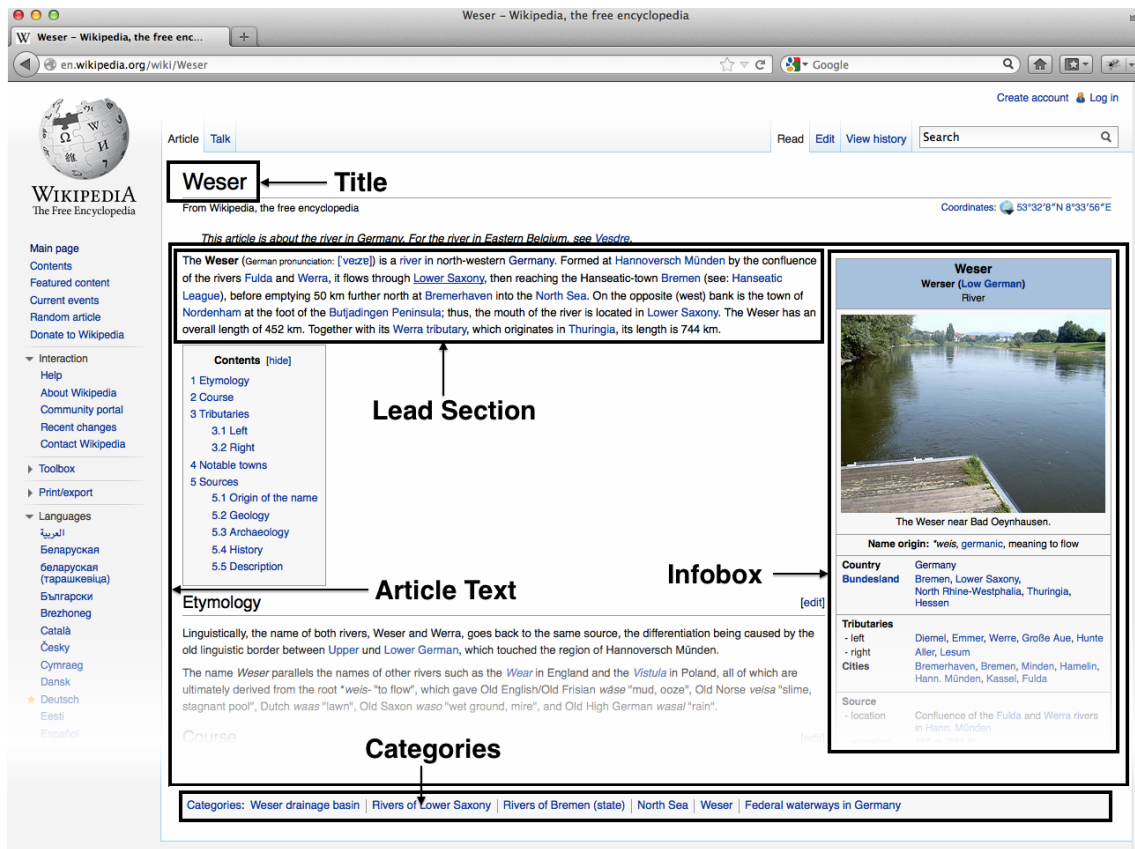


Figure 2.1: Wikipedia Page Structure

Categories

Categories are used to organize groups of articles with a similar subject. A category makes it easy to find articles of a certain topic. A Wikipedia article can belong to several categories and should be part of at least one category. The category information is located at the bottom of the article page.

For example, consider the Wikipedia Weser article. It belongs to the following Categories.

- Weser drainage basin
- Rivers of Lower Saxony

- Rivers of Bremen (state)
- North Sea
- Weser
- Federal waterways in Germany

The categories are organized in a hierarchy. Like demonstrated in Figure 2.2, a category can have multiple subcategories. Since a category can also have multiple parents, the category hierarchy is not a tree. An article should be placed in the most specific categories it belongs to.

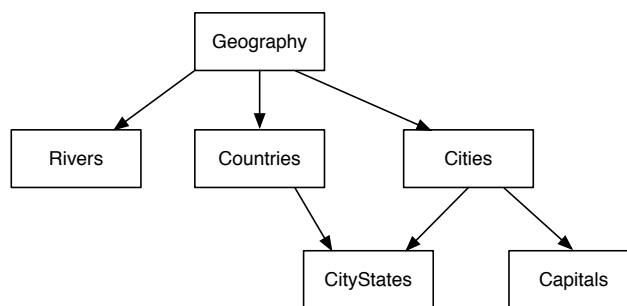


Figure 2.2: Category Hierarchy Sample

Infobox

An infobox summarizes characteristic information of an article in a table. It is displayed in the upper right corner of an article page. Articles of the same type share the same infobox template to provide a consistent presentation of some important type-specific facts.

An infobox template is a pattern for an infobox that makes it possible to instantiate the same infobox in different articles. It defines the layout of an infobox and a set of parameters. An article can then declare values for the parameters and thus instantiate the infobox template. In doing so, several articles of the same type can instantiate the same infobox with different values.

An example of the river Rhine article is shown in Figure 2.3. The left part of the Figure shows an excerpt of the parameter declaration in Wikitext. The resulting infobox is shown on the right hand side. It contains information like the length, the source and the countries the river flows through.

<pre> {{Geobox River <!-- *** Heading *** --> name = Rhine native_name = Rhein other_name = category = River <!-- *** Names **** --> etymology = [[Proto-Indo-European rc nickname = <!-- *** Image *** --> image = Loreley mit tal von linker rhei image_caption = [[Loreley]] rock in [[R image_size = 300 <!-- *** Country *** --> country = Germany country1 = Austria country2 = Liechtenstein country3 = Switzerland country4 = France country5 = Netherlands state = Luxembourg state1 = Belgium state2 = Italy state_type = Rhine Basin </pre>	<div style="text-align: center;"> <h3>Rhine (<i>Rhein</i>)</h3> <p>River</p> </div>  <p style="text-align: center;">Loreley rock in Rhineland-Palatinate</p> <table border="1"> <tbody> <tr> <td colspan="2">Name origin: Proto-Indo-European root *<i>rei-</i> ("to move, flow, run")</td> </tr> <tr> <td>Countries</td> <td>Germany, Austria, Liechtenstein, Switzerland, France, Netherlands</td> </tr> <tr> <td>Rhine Basin</td> <td>Luxembourg, Belgium</td> </tr> <tr> <td>Primary source</td> <td>Vorderrhein</td> </tr> <tr> <td>- location</td> <td>Tomasee ("<i>Lai da Tuma</i>"), Surselva, Grisons, Switzerland</td> </tr> <tr> <td>- elevation</td> <td>2,345 m (7,694 ft)</td> </tr> <tr> <td>- coordinates</td> <td>46°37′57″N 8°40′20″E﻿ / ﻿46.63250°N 8.67222°E﻿ / 46.63250; 8.67222</td> </tr> <tr> <td>Secondary source</td> <td><i>Hinterrhein</i></td> </tr> <tr> <td>- location</td> <td>Paradies Glacier, Grisons, Switzerland</td> </tr> <tr> <td>Source confluence</td> <td>Reichenau</td> </tr> <tr> <td>- location</td> <td>Tamins, Grisons, Switzerland</td> </tr> </tbody> </table>	Name origin: Proto-Indo-European root * <i>rei-</i> ("to move, flow, run")		Countries	Germany, Austria, Liechtenstein, Switzerland, France, Netherlands	Rhine Basin	Luxembourg, Belgium	Primary source	Vorderrhein	- location	Tomasee (" <i>Lai da Tuma</i> "), Surselva, Grisons, Switzerland	- elevation	2,345 m (7,694 ft)	- coordinates	46°37′57″N 8°40′20″E﻿ / ﻿46.63250°N 8.67222°E﻿ / 46.63250; 8.67222	Secondary source	<i>Hinterrhein</i>	- location	Paradies Glacier, Grisons, Switzerland	Source confluence	Reichenau	- location	Tamins, Grisons, Switzerland
Name origin: Proto-Indo-European root * <i>rei-</i> ("to move, flow, run")																							
Countries	Germany, Austria, Liechtenstein, Switzerland, France, Netherlands																						
Rhine Basin	Luxembourg, Belgium																						
Primary source	Vorderrhein																						
- location	Tomasee (" <i>Lai da Tuma</i> "), Surselva, Grisons, Switzerland																						
- elevation	2,345 m (7,694 ft)																						
- coordinates	46°37′57″N 8°40′20″E﻿ / ﻿46.63250°N 8.67222°E﻿ / 46.63250; 8.67222																						
Secondary source	<i>Hinterrhein</i>																						
- location	Paradies Glacier, Grisons, Switzerland																						
Source confluence	Reichenau																						
- location	Tamins, Grisons, Switzerland																						

Figure 2.3: Infobox Template Instantiation

Redirect Pages

A redirect page is an article without content that automatically redirects to another article. The purpose is to redirect the user to the right article, e.g if something has alternative names, alternative spellings or shortcuts. An example is *Berlin (Germany)*, which redirects to *Berlin*. A redirect page appears like a normal article, but has a notice at the top of the page like shown in Figure 2.4.

Berlin

From Wikipedia, the free encyclopedia
 (Redirected from [Berlin \(Germany\)](#))

Figure 2.4: Wikipedia Redirect Notice

2.2 XML

The *Extensible Markup Language* (XML) [23] is a specification of the W3C with the intention to build a human-readable markup language that supports a wide range of applications and is well suited for the use over the internet.

A sample of an XML document that contains data about countries is given in Figure 2.5.

```
<?xml version="1.0" encoding="UTF-8" ?>
<countries >
  <country name="Germany">
    <capital>Berlin </capital >
    <area>357021</area>
    <population >81799600</population >
  </country >
  <country name="China">
    <capital>Beijing </capital >
    <area>9640821</area>
    <population >1339724852</population >
  </country >
  ...
</countries >
```

Figure 2.5: XML Example Snippet

An XML document contains of a set of elements that are arranged in a tree. An element has a type (e.g. country), is started with a start tag (<country>) and is closed by a matching end tag (</country>). An element can contain text and other elements. Elements without content are called empty elements and can be written as <elementname />. An element that is nested inside another element is called child element of this element (country is a child element of countries). An element can have attributes, which are listed inside its start element (country has the attribute *name*).

DTD

Application-specific languages based on XML can be specified by the use of Document Type Definitions (DTD). A DTD defines a set of elements, their attributes and how the elements can be nested. An example of a language for writing web pages that is specified by a DTD is the Extensible HyperText Markup Language (XHTML) [11].

XPath

With XPath [24], parts of an XML document can be addressed. An XPath expression is a sequence of navigation steps. The following XPath expression applied to the example given above selects the capitals of all countries.

```
/countries/country/capital
```

Conditions can be stated using square brackets. The following expression selects the capital of the country that has an attribute *name* with the value *Germany*. The @ is used to address an attribute.

```
/countries/country[@name="Germany"]/capital
```

XQuery

XML documents can be queried by using the XML Query Language (XQuery) [13]. XQuery uses XPath for the node selection. The basic structure of XQuery is the FLWOR (for, let, where, order by, return) expression. XQuery is demonstrated with an example.

```
for $country in /countries/country
let $population := $country/population
where $population >= 500000000
return
<bigcountry >
  <name>{string($country/@name)} </name>
  <capital >{$country/capital/text()} </capital >
</bigcountry >
```

The for-clause iterates over all countries. For each country, the population is bound to the variable \$population. If the country has at least 500 million inhabitants, then the name and capital of the country are returned. The query returns results of the following form:

```
<bigcountry >
  <name>China </name>
  <capital >Beijing </capital >
</bigcountry >
```

2.3 RDF

The goal of the Semantic Web is to enhance the World Wide Web with computer understandable semantics. The Resource Description Framework (RDF) [17] is a data model of the Semantic Web. The idea of RDF is to represent data as a set of statements about resources. A resource may be any object like a book, a web page or a country. A resource is identified by a Unified Resource Identifier (URI). The country Germany, for example, is a resource and can be identified by the URI `http://example.org/countries/germany`.

A statement about a resource has the form *subject predicate object*. The subject is the resource the statement is made about, the predicate defines a property of the subject and the object is the value of that property. *Germany has a population of 81799600* can be expressed by the triple

```
<http://www.example.org/countries/germany>
  <http://www.example.org/population>
    81799600.
```

An RDF document is a set of triples that form a labeled directed graph, in which subjects and objects are nodes and predicates are edges. The Turtle [12] syntax is a serialization format for RDF graphs. URIs are surrounded by angle brackets, strings are enclosed in quotation marks. Triples are separated using periods. Triples about the same subject can be summarized by using a semicolon as a separator. Triples with the same subject and predicate can be listed with comma-separated objects.

```
@prefix ex: <http://www.example.org/>
<http://www.example.org/countries/germany>
  ex:name "Germany";
  ex:hasCity <http://www.example.org/city/berlin>,
             <http://www.example.org/city/goettingen>.
```

Note the prefix definition in the first line. It defines a so called namespace prefix that enables to use *ex:* as an abbreviation for *http://www.example.org/* in URIs.

2.4 Natural Language Processing

Natural Language Processing (NLP) is a field of computer science that is concerned with the computational analysis and processing of natural languages. The part of speech tag-

ging and parsing are two tasks in the NLP.

Part Of Speech Tagging

Part-of-speech (POS) tagging is the process that labels the words of a sentence according to their part of speech, such as noun, verb, adjective. A program that performs the part-of-speech tagging is called a POS Tagger. To give an example, a POS Tagger applied to the sentence "Germany is a country." returns

Germany /NNP is /VBZ a /DT country /NN

The part of speech tagging of the example sentence uses the Penn Treebank Tag Set [16] as POS labels. The tags contained in the example are described in Figure 2.6.

Tag	Description
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
DT	Determiner

Figure 2.6: Penn Treebank Part-Of-Speech Tags

Parsing

A natural language parser analyses the grammatical structure of a sentence. Chunk Parsing, also called Shallow Parsing, is a form of parsing, that, in contrast to full parsing, does not specify the relations of the constituents of a sentence, but separates a sentence into

so-called chunks. The performance of a chunk parser compared to a full parser is much higher.

Reconsider the example sentence. The output of a chunk parser may look like the following.

[*NP* Germany] [*VP* is] [*NP* a country]

The example sentence is split in 3 chunks. Like the example shows, a chunk consists of one or more words, but does not contain any information about the structure inside the chunk. Like the output of a POS Tagger, each chunk is labeled with a Penn Treebank Tag. The chunk tags are explained in the Figure 2.7.

Tag	Description
NP	Noun Phrase
PP	Prepositional Phrase
VP	Verb Phrase
ADVP	Adverb Phrase
ADJP	Adjective Phrase

Figure 2.7: Penn Treebank Chunk Tags

3 Design

The goal of the Thesis is to extract a set of geographic data from Wikipedia. This section deals with the design of the data extraction process. The Data Extraction Process is shown in Figure 4.4. It is designed in a recursive manner and consists of the following steps.

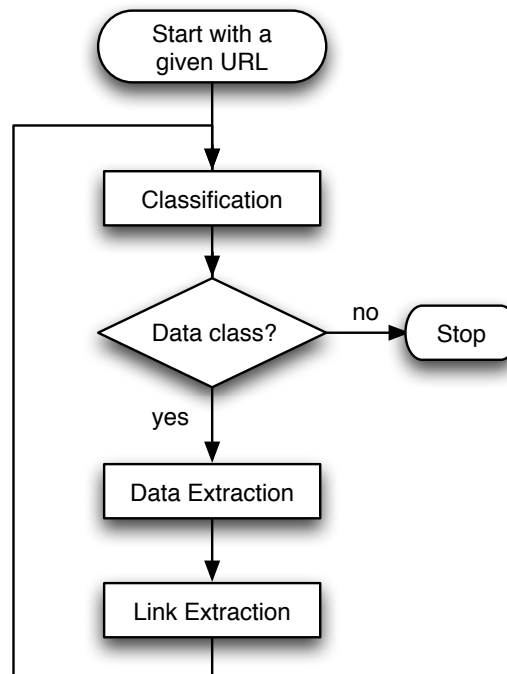


Figure 3.1: Data Extraction Process Flow Chart

1. **Classification.** The classification assigns a class to an article. There are several classes like river, lake or city. The full list of classes is given in the subsequent Section 3.1. If the assigned class is relevant for the data extraction, then the next step is performed.

2. **Data Extraction** This step extracts the relevant data of an article dependent of its class. As an example, the data extracted of a city among others contains the country it is located in, its population and its area.
3. **Follow Links** The last step extracts the outgoing links of the article. The process is then recursively called for each extracted link.

The result of the data extraction process is, on the one hand, a list of classified Wikipedia articles, and a set of extracted data on the other hand. The different steps of the data extraction process are now described in detail.

3.1 Classification

To apply a suitable wrapper for the data extraction, the Wikipedia articles need to be classified. This is done by a classifier that is presented in this section. It classifies an article into one of the following classes.

At first, the important classes that are used for the data extraction are listed.

- city
- country
- admDiv (First level administrative subdivision of a country (state, province, ...))
- island
- lake
- river
- mountain
- mountainRange
- airport

Additionally, there are some other classes that are not yet used for the data extraction, but nevertheless are classified. These classes may be used by a future version.

- person
- organization

- company
- misc (All articles that cannot be classified into another class.)

The classification is based on a multiple criteria approach. This includes the analysis of the Wikipedia categories that are associated with an article, its introduction text and its title. After the three mentioned criteria have been analysed separately, a rating algorithm combines the results to a final classification. The classification of administrative divisions is a special case. It is described in Section 3.5.

3.1.1 Category Analysis

Like already mentioned in Section 2.1, Wikipedia articles belong to categories. These categories are used for the classification. As an example consider the article about the city Göttingen. This article belongs to the category "*Cities in Lower Saxony*", which is a subcategory of *Cities* (See Figure 3.2). At the first glance, it seems to be a good idea to classify an article as a city, if it belongs to a subcategory of the *Cities* category. But due to the fact that also categories like "*Sport in Germany by city*" or "*Books about New York City*" are subcategories of *Cities*, this would result in a multitude of misclassifications. Another problem may be cycles in the category graph.

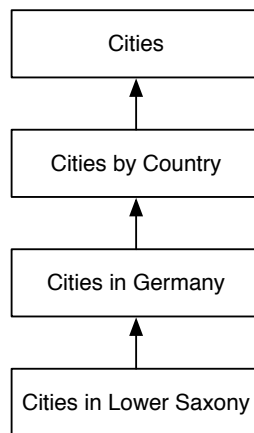


Figure 3.2: Hierarchy of the "*Cities in Lower Saxony*" Category

Instead of evaluating the category graph, a pattern matching of the category names is applied. For the most classes, there are characteristic categories that are proper for a pattern

matching. A city, for instance, often belongs to a category like *"Cities in Lower Saxony"*. Thus, the categories can be matched with the expression *"Cities in *"* in order to identify cities.

To stay with the city example, there also exists a variety of other category names like *"Port cities in Africa"*, *"Cities and towns in Molise"* or *"Mediterranean port cities and towns in Spain"*. To handle this diversity, the matching pattern for city categories has been simplified to look for categories that contain the word cities.

It has turned out to be an appropriate classification approach to match specific words in the category names. The full list of used patterns is given in Figure 3.3. Only whole words are matched, so *rivers* will not match the category *"Formula One Drivers"*. The matching is done case-insensitive.

Class	Pattern
city	cities, towns, capitals, villages
country	countries
river	rivers, streams, branches
lake	lakes
mountain	mountains
mountain range	mountain ranges
airport	airports
island	islands
organization	organizations
person	living people, deaths, births

Figure 3.3: Category Matching Patterns

Note that an article may match the categories of different classes and one class may match more than one category. This is handled by the rating algorithm presented in Section 3.1.4.

3.1.2 First Sentence Analysis

The first sentence of the lead section of an article normally specifies what the article is about. Consider the first sentence of the Wikipedia article about the Weser:

"The **Weser** is a **river** in north-western Germany."

Many articles follow the *subject is class* structure of the example sentence. This is exploited for the classification. Therefore, the subject-predicate-object-triple is extracted from the first sentence.

NLP-Based Triplet Extraction

A POS Tagger and a shallow parser are applied to the first sentence in order to break it into tagged chunks. Then, the subject-predicate-object triple is extracted from the first sentence. The algorithm is based on the idea presented in [9] and works as follows.

Consider the output of the chunk parser for the example sentence.

[NP The Weser] [VP is] [NP a river] [PP in]
[NP north-western Germany]

The first NP chunk contains the subject. The following VP chunk contains the verb of the sentence. The object is contained in the NP chunk that follows to the VP chunk.

Now consider another example sentence

"The **Orinoco** is one of the longest **rivers** in South America."

and the corresponding chunking result.

[NP The Orinoco] [VP is] [NP one] [PP of] [NP the longest rivers]
[PP in] [NP South America]

Here, the first noun phrase after the verb is *one*. In cases like *one of*, *sort of*, *kind of* and *type of* the second noun phrase chunk after the verb phrase chunk is extracted as the object. Thus, *the longest rivers* is extracted in the example.

So far, only the chunks have been extracted. The verb is determined from the VP chunk by extracting the words tagged with a VB* tag. For the object, the words annotated with a noun tag (NN*) are extracted from the object tag.

Summarized, the algorithm works as follows:

```
subjectChunk := first NP chunk
predicateChunk := first VP chunk
IF: first NP chunk after the predicateChunk ends with kind, type, sort or one
  THEN: objectChunk := second NP chunk after predicateChunk.
  ELSE: objectChunk := first NP chunk after predicateChunk.
predicate := VB* tagged words of predicateChunk
object := NN* tagged words of objectChunk
```

Note that the algorithm only works for sentences that follow the above mentioned structure. However, the algorithm has proven to work for the most articles.

Predicate Analyzation

Once the triple has been extracted, the predicate is analyzed. If the predicate is not a form of *to be*, then the algorithm "breaks".

Next, the tense of the predicate is checked. Consider the first sentence of the former European country Yugoslavia.

"Yugoslavia was a country in the western part of the Balkans during most of the 20th century."

The classifier recognises that the verb to be is in past tense and thus will not classify Yugoslavia as a country.

Object to Class Mapping

The next step is to map the identified object to the corresponding class. Therefore, an object-to-class mapping has been manually defined under consideration of synonyms. The full mapping is shown in Figure 3.4. For example, *city*, *capital*, *town* and *village* are mapped to the class city. For the sake of clarity, the table only contains the nouns in singular form, but also the plural forms are mapped (reconsider the above mentioned example: "*The Orinoco is one of the longest rivers in South America*"). Note that homonyms may cause false results.

If the object is a compound noun, then only the tail is compared. For instance, consider the two following mappings (Object => Class)

(1) mountain => mountain

Class	Words
city	city, town, capital, village
country	country, state, nation, republic
river	river, tributary, stream, branch
lake	lake, loch
mountain	mountain, summit, peek
mountain range	mountain range, mountain range system, mountain system
airport	airport
island	island
organization	organization
person	-

Figure 3.4: First Sentence Matching Patterns

(2) mountain range => mountain range

and the object *mountain range*. For mapping (1), range is compared to mountain and thus does not match. For (2), mountain range is compared to mountain range. Thus, the class mountain range is determined. Another example is *volcanic mountain*. Here, mapping (1) would match.

3.1.3 Title Analysis

Some keywords in the titles often point to a particular class. For example consider the class airport. The word *airport* is often included in the article titles of airports, like in the article about the *Frankfurt Airport*. The classification algorithm makes use of this.

3.1.4 Class Selection

So far, three different classification approaches based on the category, first sentence and title have been presented. All three criteria have the advantage that they are present in every article. The classification algorithm used in this Thesis is a combination of those and thus also applicable for article stubs. It basically works as follows. At first, each of the three classification approaches is applied separately. Then, a rating is calculated for each class. Finally, a class is determined by means of the rating.

The algorithm is now demonstrated in detail with the help of an example. Consider the Wikipedia article about Göttingen.

1) Apply the three different classification approaches

The first sentence of the Göttingen article is:

"**Göttingen** is a **university town** in Lower Saxony, Germany."

The algorithm extracts **Göttingen** (subject) **is** (predicate) **university town** (object). By use of Figure 3.4, **university town** is mapped to the class city.

The article belongs to the following categories:

- 1) Cities in Lower Saxony
- 2) Göttingen
- 3) Göttingen (district)
- 4) University towns in Germany

Reconsider the table 3.3. The city class matches category 1 (cities) and category 4 (towns). Neither category 2, nor category 3 has a match.

The title check yields no result.

2) Rating

A rating is calculated for each class. Each title, category and first sentence match adds one point. The rating is the sum of the points. Figure 3.5 shows the rating of the Göttingen article for the class city. Note that the city class matches 2 categories and thus adds 2 points.

Class	city
Categories	2
First Sentence	1
Title	0
Rating	3

Figure 3.5: Classification Rating of the Göttingen Article

3) Class selection

Now the final classification is done. On the one hand, the algorithm should be robust against individual misclassifications of the different classification approaches. On the other hand, also articles with only a few matches should be classified correctly. To cope with this, a required minimum rating has been defined. In several runs it has turned out that a minimum rating of 2 provides the best results. The classification selects the class with the highest rating that has at least 2 points. In the running example, this is the class city.

To verify the selected class, a wrapper check is performed. For the data extraction algorithm that will be presented in the following Section 3.2, a set of properties that are extracted from the Infoboxes have been specified for each class. For example, the IATA-Code is a property that is extracted from airport articles. If not at least 50 percent of the specified properties of the selected class can be extracted, then a misclassification is assumed. Note that this means that an article of a city without an infobox will not be classified as a city, but in this case there is no data to extract from this article anyway. For some classes (person, organization, company) there is no wrapper, since they are not used for the data extraction. In this case, the wrapper check is skipped.

3.1.5 Administrative Divisions

A major problem in the article classification concerns the administrative divisions class. Originally, it was tried to classify the first level divisions of a country (e.g. federal states in germany) by the use of the presented classification approach. The problem with this is that the names of the first level divisions differ from country to country (region, state, province, county, ...). Besides the different names of the first level subdivision, the names have also different meanings depending on the county. For example, Switzerland is divided in 26 cantons, whereas a canton in France is a subdivision of arrondissements and départements. This makes it hard to recognize the first level divisions of a country. Because of this, another approach is used for the classification of administrative divisions .

The ISO 3166-2 [1] standard defines codes for the subdivisions of countries. The tables in the Wikipedia articles that contain the mapping between the codes and the subdivision articles have been wrapped in order to get a list of the first level subdivisions articles.

3.2 Data Extraction

This section is concerned with the data extraction from Wikipedia articles. The focus lies on the extraction of geographic data. Like mentioned in Section 2.1, a Wikipedia article can contain an infobox that provides the important information about the article. In principle, an infobox presents characteristic data of a class in form of property-value pairs. Like the article about the river Rhine, whose infobox is shown in Figure 3.6, especially geographical articles often contain infoboxes. The data extraction algorithm that is presented in this chapter takes advantage of this and uses the infoboxes for the data extraction.





Rhine (<i>Rhein</i>) River	
	
Loreley rock in Rhineland-Palatinate	
Name origin: Proto-Indo-European root <i>*reie-</i> ("to move, flow, run")	
Countries	Germany, Austria, Liechtenstein, Switzerland, France, Netherlands
Rhine Basin	Luxembourg, Belgium
Primary source	Vorderrhein
- location	Tomasee (" <i>Lai da Tuma</i> "), Surselva, Grisons, Switzerland
- elevation	2,345 m (7,694 ft)
- coordinates	 46°37′57″N 8°40′20″E
Secondary source	Hinterrhein
- location	Paradies Glacier, Grisons, Switzerland
Source confluence	Reichenau
- location	Tamins, Grisons, Switzerland
- elevation	596 m (1,955 ft)
- coordinates	 46°49′24″N 9°24′27″E
Mouth	North Sea
- location	Hoek van Holland, Rotterdam, Netherlands
- elevation	0 m (0 ft)
- coordinates	 51°58′54″N 4°4′50″E
Length	1,233 km (766 mi)
Basin	170,000 km ² (65,637 sq mi)
Discharge	
- average	2,000 m ³ /s (70,629 cu ft/s)

Figure 3.6: Infobox of the River Rhine Article

The use of infoboxes as a basis for the data extraction has several advantages. The property-value-design of the infoboxes makes it easy to select the right data. Since articles of the same class normally instantiate the same infobox template, it is possible to use the same wrapper for all articles of one class. Another advantage of infoboxes is that they can be easily transferred from one language into another. Thus, many stub articles contain only sparse text but provide infoboxes that have been transferred from other languages.

3.2.1 Data Overview

For each class a set of properties for the data extraction has been selected. These are presented in Figure 3.7. Most of the properties are normally present in the infoboxes of the corresponding class. However, there is no guarantee that an infobox contains all the desired information.

Class	Properties
city	name, country, area, elevation, population, coordinates
country	name, official name, ISO code, capital, area, population
river	name, catchment area, length, countries, estuary, estuary coordinates, source elevation, source coordinates, cities
lake	name, width, length, depth, elevation, surface area, basin countries, location, coordinates
island	name, area, country, elevation, coordinates
mountain	elevation, range, location, coordinates
mountain range	name, elevation, country, coordinates
airport	name, elevation, IATA code, coordinates, serves, location
admDiv	name, country, area, population

Figure 3.7: Extracted Class Properties

3.2.2 Result Format

The goal of the data extraction process is not only to extract properties of different geographic classes, but also to extract references between entities. Due to the highly linked articles, Wikipedia provides a good basis for the extraction of linked data. For example consider the capital property of a country. Instead of extracting the capital name, the URI

of the referenced Wikipedia article is extracted. In doing so, a set of interlinked data like shown in Figure 3.8 is extracted. The result format of the extraction process is a set of triples. The RDF offers the proper format to store such data. The triples of the above example are given in Figure 3.9.

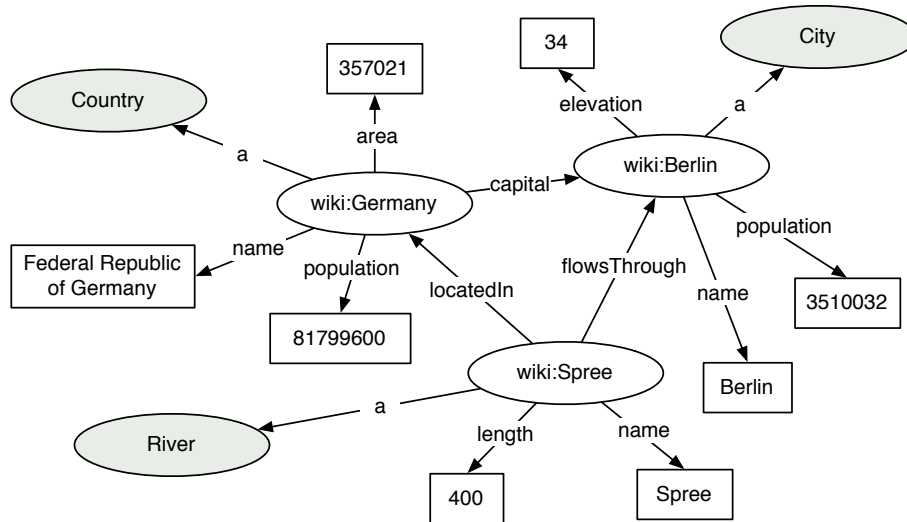


Figure 3.8: RDF Graph of Result Data

```
<http://en.wikipedia.org/wiki/Germany> a m:Country ;
  m:name "Federal Republic of Germany";
  m:population "81799600";
  m:area "357021";
  m:capital <http://en.wikipedia.org/wiki/Berlin >.

<http://en.wikipedia.org/wiki/Spree> a m:River .
  m:name "Spree";
  m:length "400";
  m:locatedIn <http://en.wikipedia.org/wiki/Germany>.

<http://en.wikipedia.org/wiki/Berlin> a m:City .
  m:name "Berlin";
  m:elevation "34";
  m:population "3507004";
  m:locatedAt <http://en.wikipedia.org/wiki/Spree>.
```

Figure 3.9: RDF Triples of Result Data

3.2.3 Extraction Algorithm

As already mentioned, the Wikipedia infoboxes are used for the data extraction. In theory, an infobox template should be used by all instances of the same class. However, in practice, different templates are used for the same class. For example, several infobox templates like *Infobox settlement*, *Infobox Russian city* or *Infobox German location* are used by cities. Furthermore, some of these templates are transferred from other language Wikipedia versions into the English Wikipedia. This results in foreign-language property names in the article definitions. During the HTML parsing, these templates are mapped to English infoboxes. For example, consider the article about the German city Göttingen. It uses the *Infobox German location* template. The left part of Figure 3.10 shows the Wikitext infobox declaration, which uses German property names. The resulting infobox that is displayed in the HTML version is shown in the right part of the Figure.

<code>{{Refimprove date=January 20 10}}</code>																																							
<code>{{Infobox German location</code>																																							
<code> Art = Stadt</code>																																							
<code> Name = Göttingen</code>																																							
<code> Wappen =</code>																																							
<code> lat_deg = 51 lat_min = 32 lat_sec = 02</code>																																							
<code> lon_deg = 09 lon_min = 56 lon_sec = 08</code>																																							
<code> Lageplan = Göttingen in GÖ.svg</code>																																							
<code> Bundesland = Niedersachsen</code>																																							
<code> Landkreis = Göttingen</code>																																							
<code> Höhe = 150</code>																																							
<code> Fläche = 140.27</code>																																							
<code> Einwohner = 127917</code>																																							
<code> Stand = 2010-12-31</code>																																							
<code> PLZ = 37001-37085</code>																																							
<code> PLZ-alt = 3400</code>																																							
<code> Vorwahl = 0551</code>																																							
<code> Kfz = GÖ</code>																																							
<code> Gemeindeschlüssel = 03 1 52 012</code>																																							
<code> Gliederung = 18 districts</code>																																							
<code> Adresse = Hiroshimaplatz 1-4
37070 Göttingen</code>																																							
<code> Website = [http://www.goettingen.de/ goettingen.de]</code>																																							
<code> Bürgermeister = Wolfgang Meyer</code>																																							
<code> Bürgermeistertitel = Oberbürgermeister</code>																																							
<code> Partei = SPD</code>																																							
	<table border="1"> <thead> <tr> <th colspan="2">Administration</th> </tr> </thead> <tbody> <tr> <td>Country</td> <td>Germany</td> </tr> <tr> <td>State</td> <td>Lower Saxony</td> </tr> <tr> <td>District</td> <td>Göttingen</td> </tr> <tr> <td>City subdivisions</td> <td>18 districts</td> </tr> <tr> <td>Lord Mayor</td> <td>Wolfgang Meyer (SPD)</td> </tr> <tr> <th colspan="2">Basic statistics</th> </tr> <tr> <td>Area</td> <td>140.27 km² (54.16 sq mi)</td> </tr> <tr> <td>Elevation</td> <td>150 m (492 ft)</td> </tr> <tr> <td>Population</td> <td>121,364 <i>(31 December 2011)</i>^[1]</td> </tr> <tr> <td>- Density</td> <td>865 /km² (2,241 /sq mi)</td> </tr> <tr> <td>First mentioned</td> <td>953</td> </tr> <tr> <td>Received city status</td> <td>ca. 1200</td> </tr> <tr> <th colspan="2">Other information</th> </tr> <tr> <td>Time zone</td> <td>CET/CEST (UTC+1/+2)</td> </tr> <tr> <td>Licence plate</td> <td>GÖ</td> </tr> <tr> <td>Postal codes</td> <td>37001–37085</td> </tr> <tr> <td>Area code</td> <td>0551</td> </tr> <tr> <td>Website</td> <td>goettingen.de ↗</td> </tr> </tbody> </table>	Administration		Country	Germany	State	Lower Saxony	District	Göttingen	City subdivisions	18 districts	Lord Mayor	Wolfgang Meyer (SPD)	Basic statistics		Area	140.27 km ² (54.16 sq mi)	Elevation	150 m (492 ft)	Population	121,364 <i>(31 December 2011)</i> ^[1]	- Density	865 /km ² (2,241 /sq mi)	First mentioned	953	Received city status	ca. 1200	Other information		Time zone	CET/CEST (UTC+1/+2)	Licence plate	GÖ	Postal codes	37001–37085	Area code	0551	Website	goettingen.de ↗
Administration																																							
Country	Germany																																						
State	Lower Saxony																																						
District	Göttingen																																						
City subdivisions	18 districts																																						
Lord Mayor	Wolfgang Meyer (SPD)																																						
Basic statistics																																							
Area	140.27 km ² (54.16 sq mi)																																						
Elevation	150 m (492 ft)																																						
Population	121,364 <i>(31 December 2011)</i> ^[1]																																						
- Density	865 /km ² (2,241 /sq mi)																																						
First mentioned	953																																						
Received city status	ca. 1200																																						
Other information																																							
Time zone	CET/CEST (UTC+1/+2)																																						
Licence plate	GÖ																																						
Postal codes	37001–37085																																						
Area code	0551																																						
Website	goettingen.de ↗																																						

Figure 3.10: Infobox Instantiation of the Göttingen Article

Even if a class uses different infobox templates, the resulting infoboxes that are displayed in the final HTML version broadly share the same properties presented in a similar layout.

This is the reason why the HTML versions of the infoboxes, and not the property value definitions in Wikitext, are used for the data extraction.

Since the Wikipedia pages are in XHTML markup, and thus XML documents, an XML query language has been chosen. The data extraction from the infoboxes is done by the use of XQuery wrappers. A snippet of the wrapper for cities that extracts the elevation is shown in the listing below. It looks for a table row in the infobox that has a first column with the text *Elevation* and binds the value of the second column to the variable \$elevation.

```
let $infobox := $document//html:table[contains(@class,"infobox")][1]
let $elevation := $infobox//html:tr[./*[1]/string() = "Elevation" ]/*[2]/string()
```

The city wrapper applied to the Göttingen article extracts the information shown in Figure 3.11.

Property	Value
name	Göttingen
country	http://en.wikipedia.org/wiki/Germany
population	121364
area	140.27 km ² (54.16 sq mi)
elevation	150 m (492 ft)
longitude	51.53389
latitude	9.93556

Figure 3.11: Extracted Raw Data of the Göttingen Article

Once the values have been extracted, they are processed depending on the datatype of the property.

Numbers and Units. Infoboxes often contain numeric values in different units. For example, consider the area property in Figure 3.11. The area is stated in square kilometres and square miles. A fixed unit has been declared for each dimension, values in other units are converted to this dimension.

URLs. Like mentioned above, the goal is to extract interlinked data. A city infobox, for example, normally references the country where the city is located in. However, some infoboxes only contain the name of the country without a link. In this case, a link with a link text that is equal to the specified name is searched within the article text, where it normally can be found. Redirects have been introduced in Section 2.1. The redirect

resolving is also part of the URL processing. If an extracted URL is a redirect, it is resolved and replaced by the URL it redirects to, in order to create triples that point to the actual URL.

The final triples that are extracted from the Göttingen article are shown in Figure 3.12.

```
<http://en.wikipedia.org/wiki/Goettingen> a m:City .  
  m:name "Göttingen";  
  m:elevation 150;  
  m:population 121364;  
  m:area 140.27;  
  m:longitude 51.53389;  
  m:latitude 9.93556;  
  m:country <http://en.wikipedia.org/wiki/Germany>.
```

Figure 3.12: Final Result Triples of the Göttingen Article

3.3 Outgoing Link Extraction

The third step in the crawler process, after an article has been classified and its data has been extracted, is to follow its outgoing links. For this, the HTML links are extracted from the article body. Since the data extraction process currently focuses on Wikipedia articles, it only follows links that reference other Wikipedia articles. But it would be conceivable to include external pages in the data extraction process.

3.4 Link Analysis

Some relations, like the country where a city is located in, or the country a river flows through, can be extracted directly from the infoboxes. In the following, an approach that tries to extend the extracted data with additional relations that are not present in the infoboxes is described.

The outgoing links of an article are used to extend the extracted data. Therefore, it is tried to conclude relations between articles on the basis of the links in the article text. For example, if a river article links to a city article, it could be assumed that the river flows through that city. By defining a set of relations between pairs of classes this information can be automatically derived using the classification results.

As an example, consider the Wikipedia article about the river Leine. The article text contains the following sentence.

Important towns upstream to down along its course are **Göttingen**, **Einbeck**, **Alfeld** and **Gronau**, before the river enters **Hanover**, the largest city on its banks.

Links to other Wikipedia articles are printed bold. By using the classifier results, it is known that the links point to city articles. Thus, the result data is extended by following triples

```
<http://en.wikipedia.org/wiki/Leine> m:flowsThrough
  <http://en.wikipedia.org/wiki/Göttingen>,
  <http://en.wikipedia.org/wiki/Einbeck>,
  <http://en.wikipedia.org/wiki/Alfeld>,
  <http://en.wikipedia.org/wiki/Gronau,_Lower_Saxony>,
  <http://en.wikipedia.org/wiki/Hanover>.
```

A problem of this approach is, that it does not consider the context in that a link occurs. Consider the following sentence of the river Oder article.

Further downstream the river is free flowing, passing the towns of Eisenhüttenstadt (where the Oder–Spree Canal connects the river to the Spree in **Berlin**) and Frankfurt upon Oder.

The wrong statement that the Oder flows through Berlin is derived from this sentence.

A further idea is to also check backlinks, so that not only a river has to link the city, but also the city has to link the river. Though, this will not guarantee that the relations are correct. A problem would be that some relations will very rarely have a backlink. For example, consider a relation between a city and a country. A country usually only links its most important cities.

A more complex approach, that also considers the text context in that the links occurs is currently under development as part of another project work.

4 Implementation

Instead of implementing the data extractor in form of a hard-coded program, a process-based approach using the MARS Framework has been chosen as a basis for the implementation. A service that provides several tasks like the classification of a URL and the extraction of infobox data for a given URL has been implemented. A MARS process is used to control the data extraction. In principle, it requests tasks of the service and decides the next steps depending on the results. In doing so, the data extraction process is very flexible, it can be easily extended with new tasks of other services and the integration of the different services is handled by the MARS Framework.

The implementation of the data extraction process is described in this chapter. At first, the design of a MARS service for the data extraction and its integration into the MARS Framework is presented. The implementation of the service is described in the second part of this chapter.

4.1 System Architecture

After the introduction of the MARS Framework, an overview of how it is used for the data extraction is given, followed by the detailed description of a service that offers tasks like the classification and the data extraction. Then, a MARS process that uses the service is presented.

4.1.1 MARS Framework

The MARS (Modular Active Rules for the Semantic Web) Framework [5], developed by the DBIS group of Göttingen University, is a general framework for implementing Event Condition Action (ECA) Rules and for processes. It is an open framework that allows to embed arbitrary languages.

This Thesis uses the MARS Framework to execute CCS processes. In principle, such a process is a sequence of service calls that take variable bindings as input, do some processing and return new variable bindings as output. Consider the example CCS process in Figure 4.1.

```

1 <execute>
2 <subject>http://www.semwebtech.org/mars/example</subject>
3 <owner>http://www.semwebtech.org/mars/persons/bdake</owner>
4 <ccs:Sequence
5   xmlns:ccs="http://www.semwebtech.org/languages/2006/ccs#"
6   xmlns:eca="http://www.semwebtech.org/languages/2006/eca-ml#"
7   xmlns:reldb="http://www.semwebtech.org/languages/2010/reldb#">
8 <ccs:Query eca:bind-to-variable="COUNTRY">
9   <eca:Opaque eca:language="http://www.w3.org/XPPath">
10    "D"
11   </eca:Opaque>
12 </ccs:Query>
13 <ccs:Query>
14   <reldb:Query>
15     <ccs:has-input-variable name="COUNTRY"/>
16     <reldb:tablename useValue="CITY"/>
17   </reldb:Query>
18 </ccs:Query>
19 <reldb:Store>
20   <ccs:has-input-variable name="CITY"/>
21   <ccs:has-input-variable name="POPULATION"/>
22   <reldb:tablename useValue="germancities"/>
23 </reldb:Store>
24 </ccs:Sequence>
25 </execute>

```

Figure 4.1: MARS CCS Process Example

In line 8-12, the value "D" is bound to the variable *COUNTRY*. This results in a single tuple that contains the variable *COUNTRY* bound to the value "D". The RELDB service offers tasks to access a relational database. For example, it can be used to query tables and to insert data into tables. In line 13-18, a table that contains city data (city, population, country) is queried. It uses the *COUNTRY* variable as input. The existing variable binding is sent to the RELDB service, which selects the city data of Germany thereupon. The returned answer is a set of tuples of the form

$$\{CITY/"x", POPULATION/"y", COUNTRY/"D"\}$$

where x and y are the values bound to the variables *CITY* and *POPULATION* like

```

{{COUNTRY/"D", CITY/"Berlin", POPULATION/3400000} ,
 {COUNTRY/"D", CITY/"Munich", POPULATION/1348650} ,...} .

```

In the last step, line 19-23, the tuples are sent to a service that stores the *CITY* and *POPULATION* variable bindings in the table *germancities*.

4.1.2 Components

Figure 4.2 gives an overview of the architecture. This includes the following components.

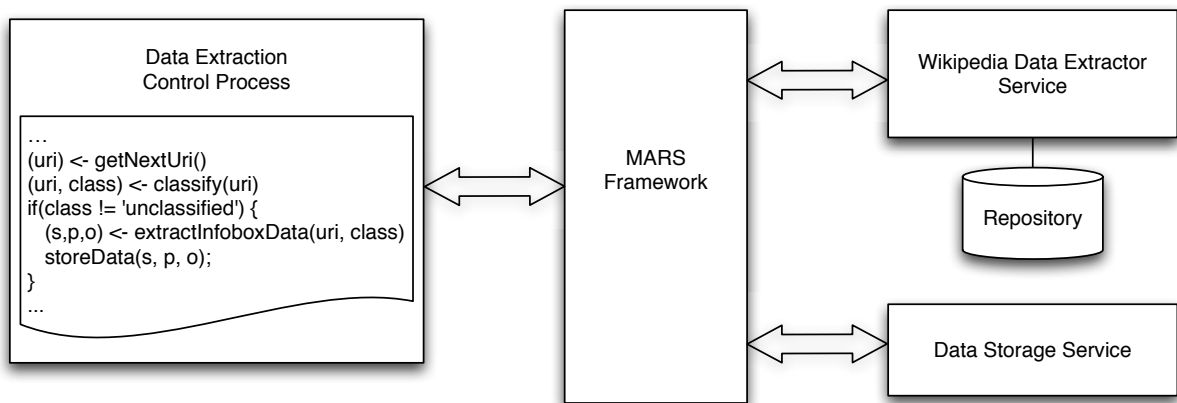


Figure 4.2: MARS Data Extraction Process Architecture

Data Extraction Control Process. The process that controls the data extraction by arranging tasks of different services.

MARS Framework. The MARS Framework is used for the execution of the data extraction control process and handles the communication with the services.

Data Extraction Service. A MARS service that performs the actual data extraction work. It provides a set of tasks like `classify(url)` or `extractInfoboxData(url)`, which are requested by the Data Extraction Control Process. The supported tasks are presented in detail in the subsequent Section 4.1.3.

Storage Service. For the storage of the extracted data, an RDF Storage component is used. It stores the extracted triples of the data extraction service.

Repository. The service maintains a repository separately from the RDF storage service where the classification results are stored. It includes a base table that stores the URL, the HTML document of that URL and a timestamp that states the creation time of the entry. The classifications are stored in a separate table, since a URI can potentially have more than one class. A third table stores the outgoing links for the reference analysis. Other services may access the repository to perform further tasks.

4.1.3 Data Extraction Service

The languages and services that are available in the MARS Framework are managed using a Language Service Registry (LSR). Further information about the LSR can be found in [6]. The LSR Entry of the data extraction service is shown in Figure 4.3. It specifies a language for the data extraction process with the tasks evaluate-query and execute-action and analyze-variables.

```

1 <mars:QueryLanguage
2   rdf:about="http://www.semwebtech.org/languages/2012/wde#">
3   <mars:name>Web Data Extractor </mars:name>
4   <mars:shortname>wde</mars:shortname>
5   <mars:isa rdf:resource="&mars;#ActionLanguage"/>
6   <mars:is-implemented-by>
7     <mars:Service
8       rdf:about="&wde-host;/services/2012/wde"
9       xml:base="&wde-host;/services/2012/wde/">
10    <mars:isa rdf:resource="&mars;#QueryService"/>
11    <mars:isa rdf:resource="&mars;#ActionService"/>
12    &wde-db;
13    <has-task-description>
14      <TaskDescription>
15        <describes-task rdf:resource="&mars;/query-engine#evaluate-query"/>
16        <provided-at rdf:resource="eval-query"/>
17        <Reply-To>body</Reply-To>
18        <Subject>body</Subject>
19        <input>element request</input>
20        <variables>*</variables>
21        <mode>asynchronous</mode>
22      </TaskDescription>
23    </has-task-description>
24    <has-task-description>
25      <TaskDescription>
26        <describes-task rdf:resource="&mars;/query-engine#analyze-variables"/>
27        <provided-at rdf:resource="analyze-variables"/>
28        <Reply-To>n.a.</Reply-To>
29        <Subject>n.a.</Subject>
30        <input>item</input>
31        <variables>n.a.</variables>
32      </TaskDescription>
33    </has-task-description>
34    <has-task-description>
35      <TaskDescription>
36        <describes-task rdf:resource="&mars;/action-service#execute-action"/>
37        <provided-at rdf:resource="execute-action"/>
38        <Reply-To>body</Reply-To>
39        <Subject>body</Subject>
40        <input>element execute</input>
41        <variables>*</variables>
42      </TaskDescription>
43    </has-task-description>
44  </mars:Service>
45 </mars:is-implemented-by>
46 </mars:QueryLanguage>

```

Figure 4.3: Query Broker LSR Entry

The Service provides the following tasks:

classify(url) -> (class)

This task classifies an URL and creates a corresponding entry in the repository.

```
<wde:Query wde:name="Classify" />
Input-Variables: (url)
Output-Variables: (url, class)
```

extractInfoboxData(url) -> (s,p,o)

As the name suggests, this task extracts the infobox data of a given URL and returns the extracted triples.

```
<wde:Query wde:name="ExtractInfoboxData" />
Input-Variables: (url)
Output-Variables: (url, s, p, o)
```

ExtractOutgoingLinks(url)

Extracts the outgoing links of a URL. Redirects are resolved. The links are not returned, but added to an internal to-do-list of the service, which can be accessed by the next task. The to-do-list is a queue for URLs in first-in-first-out order. Each URL is only returned once by the getNext function of the to-do-list. The intention of the to-do-list is to prevent that a URL is used more than once in the data extraction process. For example, if two articles have the same outgoing link, it will only be analyzed once. Another problem that is solved by the to-do-list is the synchronization of parallel running extraction processes.

```
<wde:Action wde:name="ExtractOutgoingLinks" />
Input-Variables: (url)
```

GetNextOutgoingLinks() -> (url)

Returns the next #quantity links from the to-do-list. If quantity is set to "all", then the complete content of the to-do-list is returned.

```
<wde:Query wde:name="GetNextOutgoingLinks" wde:quantity="10" />
Input-Variables: ()
Output-Variables: (url)
```

RemoveFromCache(url)

This task signals to the service that it can remove cached data for that URL.

```
<wde:Action wde:name="RemoveFromCache" />
Input-Variables: (url)
```

AnalyzeReferences()

As described in Section 3.4, the links between articles are used to derive additional relations. After the data extraction process is completed, this task can be called to extend the extracted data with additional reference properties. Since this task is based on the classification results, it should be called if all relevant articles have been classified. It returns the set of triples with the derived relations.

```
<wde:Query wde:name="AnalyzeReferences()" wde:quantity="all" />
Input-Variables: ()
Output-Variables: (s, p, o)
```

4.1.4 MARS Data Extraction Process

So far, the tasks provided by the data extraction service have been introduced. Now a combination of the tasks to a data extraction process is presented. The definition of the recursive process is shown in Figure 4.4.

Line 6-8 The next 1000 URLs are requested from the to-do-list.

Line 9-11 The URLs are classified.

Line 12 The process forks (A and B) depending on the class. The URLs that are classified as data classes (A) are used for the data extraction and outgoing link extraction. The non data class URLs (B) are not considered any further.

Line 22 (A) The outgoing links are added to the to-do-list.

Line 23-25 (A) The infobox data is extracted.

Line 26-30 (A) The infobox data triples are stored using the RDF storage component.

Line 31 (A) The URLs are removed from the cache.

Line 41 (B) The URLs are removed from the cache.

Line 44-47 The existing variable bindings are cleared and the process recursively calls itself.

After the process has finished, the link analysis can be called using the process shown in Figure 4.5.

Line 8-10 The links are analyzed.

Line 12-16 The derived relations are stored using the RDF storage component.

4 Implementation

```
1 <ccs:ProcessDefinition ccs:name="foo:process#wikicrawler"
2   xmlns:ccs="http://www.semwebtech.org/languages/2006/ccs#"
3   xmlns:wde="http://www.semwebtech.org/languages/2012/wde#"
4   xmlns:rdfstorage="http://www.semwebtech.org/languages/2012/rdfstorage#">
5   <ccs:Sequence>
6     <ccs:Query>
7       <wde:Query wde:name="GetNextOutgoingLinks" wde:quantity="1000" />
8     </ccs:Query>
9     <ccs:Query eca:bind-to-variable="class">
10      <wde:Query wde:name="Classify" />
11    </ccs:Query>
12    <ccs:Alternative >
13      <ccs:Sequence>
14        <ccs:Test>
15          <eca:Opaque eca:language="http://www.w3.org/XPath">
16            <eca:has-input-variable eca:name="class" />
17            <![CDATA[ $class = "river" or $class = "city" or $class = "country" or
18              $class = "island" or $class = "mountain" or $class = "mountainRange" or
19              $class = "lake" or $class = "airport" or $class = "admDiv"]]>
20          </eca:Opaque>
21        </ccs:Test>
22        <wde:Action wde:name="ExtractOutgoingLinks" />
23      <ccs:Query>
24        <wde:Query wde:name="ExtractInfoboxData" />
25      </ccs:Query>
26      <rdfstorage:store >
27        <ccs:has-input-variable name="s" />
28        <ccs:has-input-variable name="p" />
29        <ccs:has-input-variable name="o" />
30      </rdfstorage:store >
31      <wde:Action wde:name="RemoveFromCache" />
32    </ccs:Sequence>
33    <ccs:Sequence>
34      <ccs:Test>
35        <eca:Opaque eca:language="http://www.w3.org/XPath">
36          <eca:has-input-variable eca:name="class" />
37          <![CDATA[ $class = "misc" or $class = "person" or $class = "organization"
38            or $class = "company" ]]>
39        </eca:Opaque>
40      </ccs:Test>
41      <wde:Action wde:name="RemoveFromCache" />
42    </ccs:Sequence>
43  </ccs:Alternative >
44  <ccs:Projection >
45    <ccs:keep-variable name="" />
46  </ccs:Projection >
47  <ccs:CallProcess ccs:name="foo:process#wikicrawler" />
48 </ccs:Sequence>
49 </ccs:ProcessDefinition >
```

Figure 4.4: CCS Data Extraction Control Process

```
1 <execute>
2 <subject>http://www.semwebtech.org/mars/webdataextraction </subject>
3 <owner>http://www.semwebtech.org/mars/persons/bdake </owner>
4 <ccs:Sequence xmlns:ccs="http://www.semwebtech.org/languages/2006/ccs#"
5   xmlns:rdfstorage="http://www.semwebtech.org/languages/2012/rdfstorage#"
6   xmlns:wde="http://www.semwebtech.org/languages/2012/wde#">
7
8   <ccs:Query>
9     <wde:Query wde:name="AnalyzeReferences" wde:quantity="all" />
10  </ccs:Query>
11
12  <rdfstorage:store>
13    <ccs:has-input-variable name="s" />
14    <ccs:has-input-variable name="p" />
15    <ccs:has-input-variable name="o" />
16  </rdfstorage:store>
17 </ccs:Sequence>
18 </execute>
```

Figure 4.5: CCS Reference Analysis Process

4.2 Service Implementation

Now the implementation of the service is presented. The implementation has been done in the object-oriented programming language Java.

At first, an overview of Java libraries that are used for the implementation is given, followed by an overview of the class architecture. How the service can be configured is described in the last subsection.

4.2.1 Used Frameworks and Java Libraries

This section gives an overview of the frameworks respectively libraries that are used for the implementation of the Thesis.

JDom

JDom [7] is an open source library that provides a Java representation of XML documents. It can be used to read, manipulate and write XML documents.

Saxon-HE

Saxon-HE [25] is an open source XSLT, XPath and XQuery processor implemented in Java. It is used to evaluate XQuery queries against the XHTML Wikipedia pages.

jsoup

The open source Java library jsoup [8] provides an API for working with real-world HTML documents. It can be used to parse, manipulate and query HTML documents. A strength of jsoup is the ability to handle invalid HTML documents and transform these into valid XHTML documents.

Apache OpenNLP

Apache OpenNLP [15] is a machine learning-based Java library for the natural language processing. Among other things, it supports the segmentation of text into sentences, part-of-speech tagging and chunk parsing.

4.2.2 Class Architecture

Figure 4.6 shows the main classes of the service implementation.

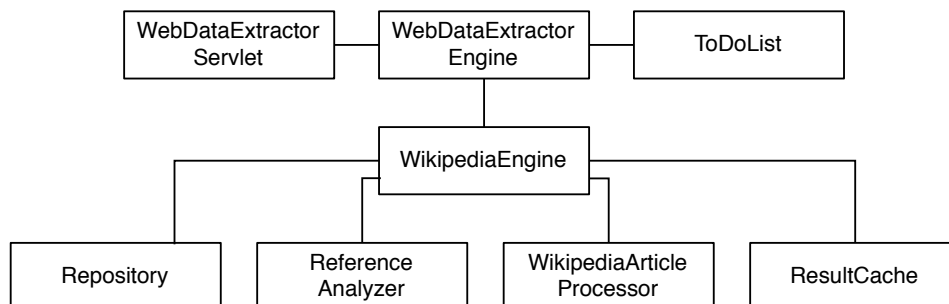


Figure 4.6: UML Class Diagram

WebDataExtractorServlet

The servlet handles the HTTP communication with the MARS Framework and passes the requests to the engine. Reconsider the LSR Entry that is given in Figure 4.3, it is accessible

under the following URLs:

http://service-url/eval-query Processes a query and returns the answer (Classify, Extract-InfoboxData, ...).

http://service-url/execute-action Executes an action (RemoveFromCache, ExtractOutgoingLinks).

http://service-url/analyze-variables Provides the analyze variables functionality.

http://service-url/reset Resets the complete Service. This includes the Repository, the ToDoList and the ResultCache.

Additionally, a web interface that can be used to query the database tables of the service is available under the service url.

WebDataExtractorEngine

The engine handles the MARS specific tasks. It has a method for each of the three main tasks analyze variables, execute query and perform action. It unwraps the request, extracts the variable bindings, and calls the functions depending on the request and wraps the results as variable bindings.

ToDoList

Like already mentioned above, the ToDoList implements a queue of URLs. It provides a method to add a new URL and to get the next URLs. The data is stored in a database.

Repository

The Repository class handles the database communication with the repository. It provides functions to add new entries, update existing entries and to query the repository.

WikipediaEngine

The WikipediaEngine handles the Wikipedia-specific tasks. Since the fetching of an HTML page and its XML parsing requires much runtime, the data extractor tries to do this only

once. For this, it maintains a cache that is used to cache infobox data and outgoing links of classified articles.

ResultCache

A cache that keeps infobox data and outgoing links of an article. It buffers the data in the memory and swaps to a database at a certain size.

WikipediaArticleProcessor

This class performs the actual processing of the Wikipedia article pages. It has methods for the classification, the infobox data extraction and the extraction of the outgoing links.

ReferenceAnalyzer

The ReferenceAnalyzer derives additional relations from the classified article links. For this, it accesses the repository in order to get the classification results and the outgoing article links.

4.2.3 Class Interaction

To get a better understanding of how the interaction of the classes works, it is now demonstrated with the help of an example.

Consider Figure 4.7. The WebData ExtractorServlet receives an HTTP request that requests the classify task of the service. It then calls the corresponding function of the WebDataExtractorEngine to handle that task. The engine unwraps the MARS-Message and extracts the input variables, which contain the URLs to be classified. It then calls the classify method of the WikipediaEngine and passes the URLs. Thereupon, the WikipediaEngine checks the Repository if the URL has already been classified. If not, then the URL is accessed and the document is loaded and parsed. The document is then handed to the classify method of the WikipediaArticle Processor. Since the fetching and parsing of the HTML document is very expensive, the parsed document is also used to extract the infobox data and the outgoing links in advance. The results are kept in the ResultCache and a Repository entry for the analyzed URLs is created. After this, it returns the classification to the

WebDataExtractorEngine. The Engine then sends a MARS-Response that contains the (url, class) tuples as answer bindings.

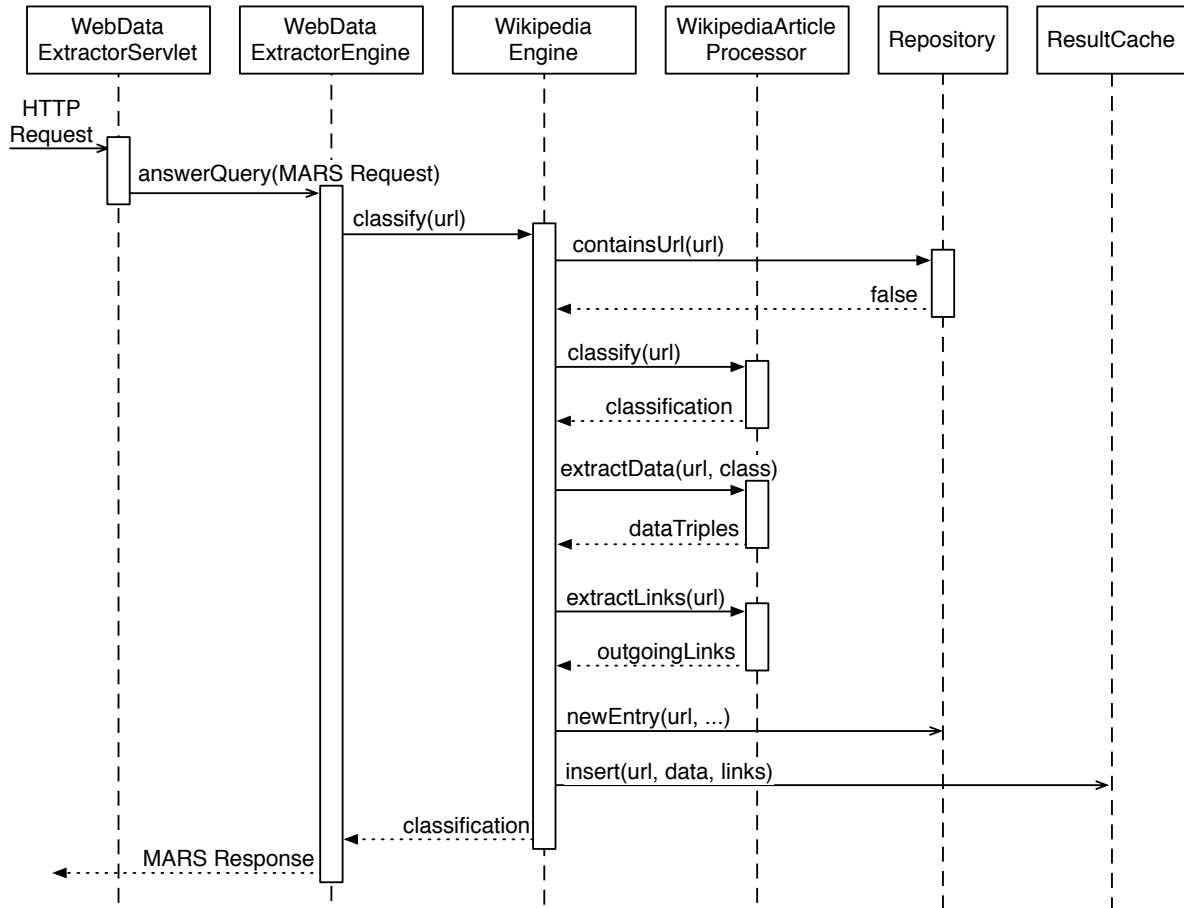


Figure 4.7: Sequence Diagram

4.2.4 Service Configuration

In the following, a short description of how the service can be configured is given. For this, XML configuration files that are located in the WEB-INF directory of the servlet are used.

Basic Configuration

The web.xml specifies the URL under that the service is accessible, the JDBC access information and the database table names that are used by the service.

Class Configuration

The class XML configuration file specifies the classifier classes, their properties and the wrapper for the data extraction. An excerpt of the XML file that specifies the city class is shown in Figure 4.8 .

The category elements specify the patterns that are used for the matching with the Wikipedia categories.

The fspattern elements specify the patterns that are used for the first sentence matching. Note that the example only includes fspattern in singular. The plural forms, which are required for cases like *"Uslar is one of the cities in Germany."* are automatically added when the file is loaded.

The properties that are extracted from the data extraction algorithm are specified by the property elements. The check attribute defines whether or not a property is considered by the wrapper check. For example, a mountain has the property range. But since a mountain is not necessarily part of a mountain range, the property is not considered for the wrapper check. The datatype and dimension attributes are required for the datatype-dependent value parsing.

The XQuery wrapper that handles the data extraction from the different infobox templates is contained in a CDATA section that is enclosed in the wrapper element.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <wikipedia-extractor-config>
3    <class name="city">
4      <categories>
5        <category>cities </category>
6        <category>towns </category>
7        <category>capitals </category>
8        <category>populated places </category>
9        <category>villages </category>
10     </categories>
11     <firstsentence>
12       <fspattern>capital </fspattern>
13       <fspattern>city </fspattern>
14       <fspattern>town </fspattern>
15       <fspattern>village </fspattern>
16     </firstsentence>
17     <properties>
18       <property name="name" check="false"
19         datatype="http://www.w3.org/2001/XMLSchema#string" />
20       <property name="country" check="true"
21         datatype="http://www.w3.org/2001/XMLSchema#anyURI" />
22       <property name="area" check="true"
23         datatype="http://www.w3.org/2001/XMLSchema#decimal"
24         dimension="http://www.semwebtech.org/mars/dimensions#area" />
25       <property name="elevation" check="true"
26         datatype="http://www.w3.org/2001/XMLSchema#decimal"
27         dimension="http://www.semwebtech.org/mars/dimensions#length" />
28       <property name="population" check="true"
29         datatype="http://www.w3.org/2001/XMLSchema#decimal" />
30       <property name="longitude" check="true"
31         datatype="http://www.w3.org/2001/XMLSchema#decimal" />
32       <property name="latitude" check="true"
33         datatype="http://www.w3.org/2001/XMLSchema#decimal" />
34     </properties>
35     <wrapper>
36       <![CDATA[
37         let infobox := document//html:table[contains(@class,"infobox")][1]
38         ...
39       ]]>
40     </wrapper>
41   </class>
42   ...
43 </wikipedia-extractor-config>

```

Figure 4.8: Classifier and Data Extractor Configuration File

Reference Analyzer Configuration

The reference analyzer has a separate configuration file. Like presented in Figure 4.9, the relations to be derived are specified as relation elements. Consider the first relation element. If a river article has an outgoing link to a city article, then the property *flowsThrough* is derived. The inverse flag specifies whether the referencing article or the referenced article is the subject of derived triple. The checkBacklink attribute specifies if backlinks are required to derive the relation.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <relations >
3   <relation from="river" to="city"
4     property="flowsThrough" inverse="false" checkBacklink="true" />
5   <relation from="airport" to="airport"
6     property="hasRouteTo" inverse="false" checkBacklink="false" />
7   ...
8 </relations >
```

Figure 4.9: Reference Analyzer Configuration File

5 Evaluation

In this Chapter, the implemented Web Data Extraction Process is evaluated.

5.1 Result Overview

Starting from the Berlin article, 100,000 articles have been analyzed in a test run.

- 89,554 data triples have been extracted
- 24,113 additional relations have been derived by the link analysis
- The extracted data contains a total number of 39,531 links between entities.

The number of classified articles per class is shown in Figure 5.1.

Class	Articles
misc	72,103
person	11,426
city	9,251
admDiv	2,512
company	1,389
airport	784
organization	602
island	536
river	440
mountain	420
country	231
lake	202
mountainRange	104

Figure 5.1: Data Extraction Process Test Results

It can be seen that the number of articles belonging to geographic classes is very high. This is because the process follows geographic articles, which often reference many other geographic articles.

5.2 Classifier Accuracy

In order to evaluate the accuracy of the classifier, 100 articles of each classified class have been chosen randomly. They were manually classified and then compared to the automatic classifications. The result is given in Figure 5.2. A row contains a class, the number of articles that have been classified as this class but belong to another class (false positive) and the number of articles that belong to this class but have been classified into another class (false negative).

It can be seen, that the classification of the classes that use a wrapper check works accurate. The false positives of the classes that use a wrapper check can be explained by the fact that some classes widely share the same properties and thus can pass the wrapper check of a false class (e.g. mountain vs. mountain range). On the other side, the classes without a wrapper check, especially organizations, are more error-prone.

Class	False Positive	False Negative
city	2	2
country	1	0
river	0	0
lake	0	1
mountain	4	0
mountainRange	0	4
airport	0	0
island	3	1
organization	9	1
person	0	0
company	4	2
admDiv	0	0

Figure 5.2: Classifier Test Results

5.3 Reference Analysis

The analysis of the outgoing links that is described in Section 3.4 has derived 24,113 additional relations from the article text. Like already mentioned, the current approach does not pay attention to the context in that a link occurs and thus is error-prone. A check of 100 randomly selected relations has been done and shows 16 wrong relations.

5.4 Runtime

The data extraction process makes a lot of blocking database requests (mars database variables, repository, cache). Thus, the runtime highly depends on the connection to the database. Another main influence of the runtime is the connection to the Wikipedia servers and their workload. In practice, the average processing time per article lies between 0.5 seconds (good connections) and 2 seconds (bad connections).

6 Related Work

Due to the large number of articles, Wikipedia is an attractive target for researchers. This chapter presents some work that is related to this Thesis.

Information extraction from Wikipedia in a large scale is done by the DBpedia [4] [2] project. It extracts structured information from Wikipedia and makes it available in the Web. DBpedia focuses on the extraction of information like the article name, infoboxes, categories and coordinates of an article rather than on the actual article text. Extracted entities are classified using different classification schemata. Besides the use of Wikipedia categories, the entities are also classified using the YAGO Classification. But since the YAGO Classification is derived from the Wikipedia categories using WordNet, its quality depends on the Wikipedia categories. A third classification based on the infobox templates is available. In contrast to this Thesis that uses the HTML version for the data extraction, DBpedia works with the Wikitext template declarations. Thus, it extracts German property names like *bundesland* and not the English name *state* from the Göttingen infobox like done in this Thesis.

[21] implements a self-supervised information extraction system for Wikipedia. It uses pages with similar infoboxes to learn about their attributes and to use this for the automatic creation of extractors in order to create new infoboxes and to complete other infoboxes.

A technique to extend Wikipedia with machine-readable semantic annotations is presented by [10]. The extension called Semantic MediaWiki of the Wikipedia engine enhances the wiki markup and enables to annotate links and to declare properties of an article by annotating values.

For the classification of Wikipedia articles into classes of named entities several attempts were made. [20] uses a combination of Wikipedia and WordNet [14] for automatically building gazetteers for the Named Entity Recognition. It determines the entity class a Wikipedia article it belongs to based on the nouns in the first sentence of the article and

a noun hierarchy of WordNet. A similar approach that also uses the first sentence of an article page for the named entity recognition is presented by [9]. In contrast to [20], they focus on the first noun phrase after the verb "to be" and use it as a category label.

[3] deals with the extraction of named entities and synonyms from Wikipedia. It uses a pattern matching of the Wikipedia categories to recognize articles about companies, organizations and people.

An approach to identify Wikipedia articles about persons, locations and organizations is presented in [18]. The classification is done by a Support Vector Machine that is trained among others with infobox data, text tokens and category links in multiple languages using the Wikipedia article cross language links.

A classification method for classifying Wikipedia articles into a set of 15 named entity classes is proposed by [19]. It first uses a binary classifier to distinguish between articles about named entities and other articles. Then, the named entity articles are classified using a support vector machine. It is trained with Wikipedia features like categories, templates and infoboxes and uses a bag-of-words of the first paragraph of an article page.

7 Conclusion

A process for the web data extraction that uses Wikipedia to extract geographic data has been implemented in this Thesis. The use of a classifier that assigns a class to an article in combination with class-specific infobox wrappers has turned out to be an appropriate approach.

A challenge was the design of the classification algorithm, that, in contrast to the most existing Wikipedia classification approaches that use rather general classes, classifies articles into specific geographical classes. The developed classification algorithm works properly for most classes, but especially the classification of administrative divisions could be improved.

The adaptation of the data extraction wrapper to different infobox templates could be coped by using flexible XQuery wrappers. By using the links that are present in the article text and in the infoboxes, the extracted data exhibits a large amount of annotated relations between entities that are not included in DBpedia.

Further Work

Like already mentioned in Section 3.4, the analysis of the outgoing links of an article provides potential for improvements. A more complex approach is currently under development.

Due to the use of a MARS process for the control of the data extraction, the data extraction process could also be easily extended with new functionality. Even if only Wikipedia is used for the data extraction by the approach implemented in this Thesis, it could also be applied to other Web pages by adapting the classification algorithm and data extraction wrapper.

Bibliography

- [1] ISO 3166. http://www.iso.org/iso/country_codes.html. 2012.
- [2] C. Bizera, J. Lehmann, G. Kobilarova, S. Auerb, C. Beckera, R. Beckera, and S. Hellmann. DBpedia - a crystallization point for the Web of Data. In *Web Semantics: Science, Services and Agents on the World Wide Web Volume 7 Issue 3*, pages 154–165, 2009.
- [3] C. Bohn and K. Norvag. Extracting Named Entities and Synonyms from Wikipedia. In *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 1300–1307, 2010.
- [4] DBpedia. <http://dbpedia.org/>. 2012.
- [5] MARS Framework. <http://www.dbis.informatik.uni-goettingen.de/mars/>. 2012.
- [6] O. Fritzen, W. May, and F. Schenk. Markup and component interoperability for active rules. In *The Second International Conference on Web Reasoning and Rule Systems*, pages 197–204, 2008.
- [7] Jdom. <http://www.jdom.org/>. 2012.
- [8] jsoup. <http://jsoup.org/>. 2012.
- [9] J. Kazama and K. Torisawa. Exploiting Wikipedia as External Knowledge for Named Entity Recognition. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 698–707, 2007.
- [10] M. Krötzsch, D. Vrandecic, M. Völkel, H. Haller, and R. Studer. Semantic Wikipedia. In *Journal of Web Semantics*, pages 251–261, 2007.
- [11] The Extensible HyperText Markup Language. <http://www.w3.org/tr/xhtml1/>. 2002.

- [12] Turtle - Terse RDF Triple Language. <http://www.w3.org/teamsubmission/turtle/>. 2011.
- [13] XQuery 1.0: An XML Query Language. <http://www.w3.org/tr/xquery/>. 2010.
- [14] WordNet - A lexical database for English. <http://wordnet.princeton.edu>. 2012.
- [15] Apache OpenNLP library. <http://opennlp.apache.org/>. 2012.
- [16] The Penn Treebank Project. <http://www.cis.upenn.edu/treebank/>. 2012.
- [17] Resource Description Framework (RDF). <http://www.w3.org/rdf/>. 2004.
- [18] I. Saleh, A. Fahmy, and K. Darwish. Classifying Wikipedia Articles into NE's using SVM's with Threshold. In *Proceedings of the 2010 Named Entities Workshop*, pages 85–92, 2010.
- [19] M. Tkachenko, A. Ulanov, and A. Simanovsky. Fine Grained Classification of Named Entities In Wikipedia. In *HP Laboratories Technical Report - HPL-2010-166*, 2010.
- [20] A. Toral and R. Munoz. A proposal to automatically build and maintain gazetteers for Named Entity Recognition by using Wikipedia. In *EACL 2006*, 2006.
- [21] D. Weld, R. Hoffmann, and F. Wu. Using Wikipedia to bootstrap open information extraction. In *ACM SIGMOD Record Volume 37 Issue 4*, pages 62–68, 2009.
- [22] Wikipedia. <http://http://www.wikipedia.org/>. 2012.
- [23] Extensible Markup Language (XML). <http://www.w3.org/tr/rec-xml/>. 2008.
- [24] XML Path Language (XPath). <http://www.w3.org/tr/xpath/>. 1999.
- [25] Saxon XSLT and XQuery Processor. <http://saxon.sourceforge.net/>. 2012.