Designing Linked Data Applications

MASTER THESIS

submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE

by

Bas Schoenmakers
born in Delft, The Netherlands

TU Delft

Web Information Systems Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, The Netherlands
http://eemcs.tudelft.nl
Abstract

A lot of (legacy) datasets can be combined with other datasets in order to increase their value. New technologies and ideas around the Semantic Web are evolving to make these enrichments possible. In recent years, the Linked Data principles have become the established standard for publishing data on the Semantic Web. With these techniques and principles it becomes possible to enrich a dataset using an interlinking between different datasets in the Linked Data cloud. However, transforming a dataset into a Linked Data dataset is a difficult process. This thesis proposes a way to perform this transformation and explains several required steps needed to be taken in order to succeed. It will discuss difficulties and experiences with entity resolution and evaluates an automatic interlinking approach with the purpose of connecting datasets and creating new opportunities with a (legacy) dataset. As an end result, this thesis will deliver a Linked Data application for the Software Technology publications dataset.

Graduation Committee:
Prof. dr. ir. G.J. Houben, Faculty EEMCS (WIS), TU Delft
Dr. Laura Hollink, Faculty EEMCS (WIS), TU Delft
Dr. Martin Pinzger, Faculty EEMCS (SE), TU Delft
Samur Araujo MSc, Faculty EEMCS (WIS), TU Delft
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Optimize the automatic interlinking process</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>5.1 Disambiguating authors to create URIs</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>5.1.1 Approximate string matching</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>5.1.2 Attribute-based entity resolution</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>5.1.3 Relational-based entity resolution</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>5.2 Interlinking with tools</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>5.2.1 Different tools to interlink</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>5.2.2 Comparison between the tools</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>5.3 Our optimized interlinking method</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>5.3.1 Architecture overview</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>5.3.2 Link Language Specification extension</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>5.3.3 Analyzing the results</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>5.3.4 Determine threshold values</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>5.3.5 Experiment results</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>5.3.6 Merging entities</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>5.4 Conclusion</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>A Linked Data application</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>6.1 Application overview</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>6.2 Enrichment of the dataset</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>6.2.1 Merge authors and publish</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>6.2.2 Location links against GeoNames</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>6.2.3 Publishers links against DBpedia</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>6.2.4 Author links against DBLP</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>6.3 Example of topic retrieving due to entity relations</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>6.4 Conclusion</td>
<td>60</td>
</tr>
<tr>
<td>III</td>
<td>Conclusions and future work</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>7.1 Summary per research question</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>7.2 Final remark</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>Future work</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>8.1 Mapping relational databases to RDF</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>8.2 Defining automatic interlinking with ontologies</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>8.3 Model the enrichments</td>
<td>65</td>
</tr>
</tbody>
</table>

References | 66

Appendix | 69

A Description of the used ontologies and vocabularies | 69

B Example of the D2R mapping | 71
List of Figures

1. The Semantic Web technology stack .................................................. 14
2. RDF statements shown in a diagram .................................................. 18
3. The first part of the database schema: administrative information ...... 29
4. The second part of the database schema: core publications data ...... 29
5. Different URIs for same entity .............................................................. 33
6. An example of the disambiguation of three different URIs that can be mapped to the same person ......................................................... 46
7. Code fragment of the Silk-LSL file with the semantic matching extension .......................................................... 47
8. Benefits of using Linked Data .............................................................. 54
9. Aggregation of publications per author .............................................. 55
10. World map of publications locations per author .............................. 56
11. DBpedia information about a city ..................................................... 57
12. Using Linked Data to retrieve topics per author ............................... 58
13. Topics of an author .......................................................................... 59
14. Top co-authors of an author .............................................................. 60
A.1 Example of D2R mapping ............................................................... 71
List of Tables

1  Overview of different BibTex to RDF tools . . . . . . . . . . . . 32
2  Comparison between interlinking tools . . . . . . . . . . . . . . 44
3  Precision and recall scores of Silk and the extension of Silk . . 50
A.1 Vocabularies used within the application . . . . . . . . . . . . . 69
A.2 The different classes describing our entities . . . . . . . . . . . . 69
A.3 The different properties used to describe our entities . . . . . . . 70
Part I
Context
1 Introduction

In recent years, the Linked Data principles [1], guidelines to interlink datasets and share structured data on the web, have become the established standard for publishing data on the Semantic Web [2]. The Semantic Web, imagined by Tim-Berners Lee (inventor of the Web) describes methods and technologies to allow machines to understand the meaning, or ‘semantics’, of information on the World Wide Web [3]. In parallel with the Semantic Web, Tim-Berners Lee proposed his idea of Linked Data as a way to publish data online according to the RDF data model [4] in such a way that data is machine-readable, its meaning is explicitly defined and data items are linked between different datasets. Within this context, publishing describes the way to present data on the web. The basic assumption behind Linked Data is that the value and usefulness of datasets increases the more it is interlinked with other data [5]. Because a lot of datasets are currently not published as Linked Data, there is a transformation process needed to convert them into Linked Data datasets. This transformation process introduces questions and problems for different kinds of datasets, but this transformation also opens up new functionalities and abilities.

This chapter introduces the problem and motivation of this Master thesis project. It will define the goal of the project and introduces the related research questions which are covered throughout this document. It will conclude with an introduction to the different chapters and structure of the document.

1.1 Background and motivation

In a nutshell, the Semantic Web is about enabling computers to process information from the World Wide Web. On the web, text documents link to other text documents through hyperlinks [2]. These hyperlinks are accessed through a word, phrase or image in the document; as we all know, these links are intended for a human user to follow by browsing. Humans usually know what the intention of a link is by contextual clues in the document. However, if someone intends for computers to process this information, the exact relationship between two documents is not always clear [6]. Even within a single document, words and sentences have ambiguous meanings, and computer processing of text has a long way to go before reaching the level of human understanding.

The last several years have been the real start for the Semantic Web [2]. A web where machines are able to understand the information within webpages, because of structured meta-data. A lot of Semantic Web applications have been created and a lot of research is going on in this area [2]. One of the strongest points of the Semantic Web is its ability to interlink datasets, because of the structured meta-data available. The Linking Open Data (Linked Data) project, started by the W3C organization1, is all about this interlinking of datasets. If people are able to create an interlinking between different datasets, they are able to enrich datasets with relevant information from the interlinked dataset

1http://www.w3.org/
(e.g. if a webpage informs the reader about the city Paris in France, it would be beneficial to add relevant data about this city from Wikipedia2). In order to perform this interlinking, datasets need to be published in such a way that they present their meta-data (within the Semantic Web, the RDF data model is the standard to describe this meta-data [4]).

At the moment, most datasets are published without their meta-data. There is a transformation process needed in order to structure and present this relevant meta-data. Currently, there is not a single way to perform this transformation. The tools to perform (automatic) interlinking between datasets in order to enrich them as well as ways to publish the information are lacking.

It would be great if the ability to interlink datasets and work with the web as one huge database, will become mainstream. It will open up several possibilities and new functionalities. There are technological issues needed to be solved, like the ones discussed above, together with privacy issues and provenance problems (e.g. can we trust this data we interlink against and is this data valid?). This thesis proposes a way to use Semantic Web techniques, explains the way they can operate together, in order to transform a (legacy) dataset into a Linked Data dataset. As a validation of this transformation process, a real Linked Data application has been created, which will show the benefits and enrichments.

1.2 Problem definition and goal

This thesis proposes a way to create a Linked Data dataset out of existing or non-existing (relational) datasets. Within this transformation process, the interlinking aspect is a difficult once, because most datasets are currently not published as Linked Data. This means that the relevant meta-data is not available and machines do not understand the structure.

There are a lot of legacy datasets or applications which can or need to be transformed into Linked Data datasets in order to enrich them and enlarge the opportunities of these datasets. After this transformation process machines are able to access the relevant meta-data and they have the ability to understand the underlying structure of the data. After the transformation process, the data needs to be published. This whole process introduces problems and a lot of decisions need to be made.

1.2.1 Research goal

The research for this thesis was coordinated at the TU Delft in the WIS (Web Information Systems) group. The following research goal was defined:

Develop an approach to create Semantic Web applications by transforming a legacy (relational) dataset into RDF, enriching this RDF by (automatically) interlinking entity URIs of several different datasets from the Linked Data cloud, and publishing the resulting RDF as Linked Data.
In this research we developed several research questions in order to reach the research goal. These research questions will be answered in this thesis document. Each of the following chapters answer a research question in the following order:

- Chapter 2: What is the Semantic Web and which techniques are used in developing applications in the Semantic Web?
- Chapter 3: What is Linked Data and what can Linked Data be used for in the Semantic Web?
- Chapter 4: What steps are needed in transforming a (legacy) dataset into Linked Data?
- Chapter 5: How can the automatic interlinking process between entity URIs in Linked Data be optimized?
- Chapter 6: Demonstrate the approach for application on the ST Publications data. What are the results if we apply the steps to the ST Publication data?

Every chapter will conclude with an answer to the specific research question.

1.3 Structure of this document

This thesis document is divided into three different parts. The first part starts with a context sketch of the main subjects: the Semantic Web and Linked Data. It will show what the ideas of these visions are and what techniques are required in order to create applications for them. This part will also introduce example applications, technological issues and current achievements of Linked Data. It presents the state of the art and it will show what can be created with the existing techniques.

The second part will talk about the transformation process of a legacy (relational) dataset to RDF in order to create a Linked Data application out of it. It will show problems and decisions popping up during this process. This process will be exemplified through the creation of a real Linked Data application. It will also show different problems needed to be solved in order to get to an appropriate result. Because ambiguity within datasources is a big problem during the interlinking process of creating a Linked Data application, this part focuses on techniques to solve this ambiguity problem. The second part concludes with a chapter about the benefits we get from transforming the dataset into RDF and interlinking the dataset against external datasets.

The third and last section will conclude the work. It will analyze the research questions to see whether or not the research goal has been reached. It will end with future work and final remarks.
2 Semantic Web

This chapter will introduce the Semantic Web [3]. The Semantic Web is an addition to the normal web we all know. The Semantic Web is the basis for the creation of Linked Data applications. This chapter will discuss the following research question: what is the Semantic Web and which techniques are used in developing applications in the Semantic Web? Supported by literature, it will discuss the techniques, the way they work together and how applications can be created with them.

2.1 Introduction

In order to understand what the Semantic Web is about, it is good to have an understanding about what ‘semantic’ really means in this context. Semantics is about the study of meaning, and in this case, the meaning of data. The web is a collection of documents, linked with each other via hyperlinks. These documents contain the information we, as ordinary users, read and understand. There is, however, no meaning behind these documents. This means that machines or computers will not understand the intention behind this information. If people, in this case the information providers, are able to provide this semantic information to machines, these machines can do a lot of extra work, which they were not able to do before. If machines are able to understand the meaning of data, they become able to provide the user with information based on the user’s intentional needs.

The Semantic Web became an official initiative of the World-Wide-Web Consortium (W3C) [7] and has attracted a lot of attention lately. The initiative was started by the founder of the web, Tim Berners-Lee[1]. The web he imagined had to be self-adaptable, flexible and more automatic than the web everyone uses at the moment. The first clear vision of the Semantic Web was expressed in an article written by Tim Berners-Lee, Jim Hendler and Ora Lassila in Scientific American in 2001 [3].

“The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users.”

Listing 1: Adding semantics (rdf:about=””) to hyperlinks within webpages

```
<ul>
    <li rdf:about="http://dbpedia.org/resource/Paris">City 1</li>
</ul>
```

An example of this meaningful content in practice is shown in Listing 1, where the code describes the semantics of the hyperlink. The semantic infor-
mation is provided using the `rdf:about` tag. This tag tells (machines) that City 1 is about the resource: `http://dbpedia.org/resource/Paris`. This resource is called a URI [1] (this will be explained in the following section 2.2). It means that City 1 is annotated with this resource (the URI) and this resource is able to provide a machine with information about City 1. So in this particular case: City 1 is about DBpedia’s ‘Paris’ and city 2 is about DBpedia’s ‘Berlin’.

### 2.2 Semantic Web technology stack

The techniques used to create Semantic Web applications are well defined in a technology stack (see Figure 1) [8]. The stack will be discussed top-down and this section will then dive into the most important techniques and give a detailed explanation.

![Figure 1: The Semantic Web technology stack](image)

- **Trust**: The top layer of the stack is **Trust**: this layer addresses issues of trust that the Semantic Web can support. This component has not progressed far beyond a vision of allowing people to ask questions of the trustworthiness of the information on the web, in order to provide an assurance of its quality.

- **Logic and Proof** is the second layer: this layer provides an (automatic) reasoning system on top of the ontology structure to make new inferences. Thus, using such a system, a machine can make deductions as to whether a particular resource satisfies its requirements (and vice versa).
2.2.1 Ontologies

The term ontology has its roots in the philosophical domain [9]. In order to understand the basic structure of our world, the word ontology has been connected with a branch of metaphysics. The problem is that the philosophical definition of ontology does not correspond well with the definition in Computer Science. Within this domain, an ontology is defined by [9] as: “An ontology is a detailed model/picture/schema (can be intertwined) of a slice of reality which is based on the facts that we know about that reality. This model/picture/schema is a description of some of the things and some of the relationships between the things that are known about that reality.”

Ontologies can be shared by different applications, people and databases within a domain. A domain can be an area of knowledge, like medicine or a more specific subject area like publications. Ontologies are able to specify the following kinds of concepts:

- Classes (of things)
- Relationships between classes
- Properties (attributes) of classes
There are many motivations for developing and using ontologies [10]:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge

Within the Semantic Web it is common to, wherever possible, reuse ontologies instead of creating new ones [5]. In order to make it as easy as possible for machines to process your data, it is best to reuse terms from well-known ontologies. Only define new terms if it is impossible to find the required terms in existing ontologies, so that redundancy is minimized.

### 2.2.2 XML and RDF

The basis for the whole Semantic Web framework is XML (eXtensible Markup Language). The XML syntax is a subset of the international text processing standard SGML specifically intended for use on the web. XML has the ability to structure data (see Listing 2 for an example).

#### Listing 2: Example XML document

```xml
<authors>
  <author>Jan Hidders</author>
  <author>Geert-Jan Houben</author>
</authors>
```

The XML standard is not sufficient to add semantics to webpages. It allows users to add arbitrary structure to their documents, but does not say anything about what the structure means. From Listing 2 can be concluded that XML allows users to create their own markup (e.g. `<author>`), which seems to carry some semantics. A machine however, does not know what an author is and how the concept ‘author’ is related to, for example, the concept ‘person’.

RDF tries to overcome the problems of XML mentioned in the previous paragraph. The basic concept of RDF is to encode (meta-)data in sets of triples, each triple being the subject, verb (or predicate) and object of an elementary sentence. Assertions are made that particular things (e.g. people, webpages, or whatever imaginable) have properties (such as ‘is a sister of’ or ‘is created by’) with certain values (another person or another webpage).

This structure turns out to be a natural way to describe the vast majority of the data processed by machines. Subjects and objects are each identified

---

3Standard Generalized Markup Language, defined by ISO 8879
by a URI. The predicates are also identified by URIs, which enables anyone to define a new predicate just by defining a URI for it. Because RDF uses URIs to encode this information in a document, the URIs ensure that concepts are not just words in a document but can be tied to a unique definition that everyone can find on the Web.

2.2.3 RDF specifications

The RDF data model is a syntax-neutral way of representing RDF expressions. The basic data model consists of three object types:

- **Resources** All things being described by RDF expressions are called resources. A resource may be a webpage (HTML document), a part of a webpage (a fragment) or a collection of pages, e.g. an entire website. A resource may also be an object that is not directly accessible via the web (e.g. a printed book or a person).

- **Properties** A property is a specific aspect, characteristic, attribute or relation used to describe a resource. Each property has a specific meaning and can define its permitted values, the types of resources it can describe and its relationship with other properties.

- **Statements** A statement is a specific resource together with a named property plus the value of that property.

RDF uses a particular terminology for describing various parts of statements similar to the grammar rules of neutral languages such as English. Specifically, the part that identifies the thing the statement is about (the webpage in this example) is called the subject. The part that identifies the property or characteristic of the subject that the statement specifies (e.g. creator, creation-date or language) is called the predicate and the part that identifies the value of that property is called the object [4]. Hence, a statement is a triple of the following form:

{sub, pred, obj}
A nodes-and-arcs diagram can be used to visualize RDF statements pictorially, as shown in Figure 2\(^4\). It represents the hierarchy of an article together with people related to the article. The subjects and objects are represented by circles, while the predicates are within the arcs.

### 2.2.4 SPARQL

The SPARQL Query Language for RDF (SPARQL) is a query language for RDF and can be used to express queries across diverse data sources [11]. SPARQL has much in common with the SQL query language for databases, however it is currently lacking a lot of features and in terms of speed SPARQL can not compete to SQL.

To be able to understand how SPARQL works, a SPARQL query example is presented in Listing 3.

**Listing 3: Example SPARQL query**

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?website FROM <http://planetrdf.com/bloggers.rdf>
WHERE {
  ?person foaf:weblog ?website;
  foaf:name ?name .
  ?website a foaf:Document
}
```

By having a look at the syntax, one can clearly see the similarity to the SQL query syntax. Going deeper in detail, the SPARQL syntax enables the

\(^4\)http://www.semanticfocus.com/media/insets/rdf-graph.png
usage of prefixes with the PREFIX notation, which enables the abbreviation of query resources. In the given example (Listing 3), the prefix defines the abbreviation foaf (abbreviation of URI http://xmlns.com/foaf/0.1/). These prefixes can later be used in the query to point to resources without having to type the whole URI of the resource, which makes the queries more readable. In the presented example (Listing 3) the foaf:weblog can be resolved to http://xmlns.com/foaf/0.1/weblog. The example query selects names and websites of people contributing to the http://planetrdf.com website.

The results of the query are sets of triples of a variable name with a value. So the above query binds the two variables (?name and ?website) with pattern matching algorithms traversing the RDF. This means that every name and website of people with a website and name are bounded and selected.

2.3 Conclusion

The Semantic Web is a self-adaptable, flexible and more automatic web than the web currently known. There are a lot of techniques being used to create applications within the Semantic Web. The Semantic Web technology stack describes these techniques. This chapter discussed the most important techniques (ontologies, RDF and SPARQL) in order to get a feeling what the opportunities of these techniques are. While ontologies define the representation of the knowledge, the knowledge is actually defined within RDF. SPARQL is then able to query the RDF data. In this thesis these techniques will all be used in the creation of a Linked Data application.
3 Linked Data

The Linking Open Data project (Linked Data) is a project started by the W3C and builds upon the Semantic Web ideas. In this chapter the following research question will be answered: what is Linked Data and what can Linked Data be used for in the Semantic Web? The adoption of the Linked Data practices has lead to the extension of the web with a global data space connecting data from diverse domains [2]. The idea behind Linked Data is simple: if everyone puts his created structured data on the web and creates links between this data, we are emerging to a Web of Data. This Web of Data enables new types of applications [2]. By linking data it becomes possible to see relations which were never seen before. Interlinking has the ability to create new information, by finding new and interesting combinations. This chapter starts with the main vision and principles of Linked Data and some example applications in order to illustrate the benefits. The latter sections continue with an explanation of the interlinking principles and a conclusion.

3.1 Vision and principles

In 2006, Tim Berners-Lee outlined four rules, which are known as the ‘Linked Data principles’, and described them as follows [1]:

1. Use URIs as a name for things.
2. Use HTTP URIs so that people can look up those things.
3. Provide useful information, using the standards (SPARQL, RDF), when someone looks up a URI.
4. Include links to other URIs, so that people can discover more things.

Berners-Lee states that if people enhance their datasets by providing links to other datasets, the Web of Data will grow and it will allow users to browse it. The principles rely on two technologies: URIs and the HyperText Transfer Protocol (HTTP), in combination with two specific Semantic Web technologies: RDF and SPARQL. The Web of Data or Linked Data cloud is growing fast. In October 2007, datasets consisted of over two billion RDF triples, which were interlinked by over two million RDF links. By May 2009 this had grown to 4.2 billion RDF triples, interlinked by around 142 million RDF links [2].

3.1.1 Example applications

An example of a Linked Data application is Revyu [12]. Revyu is a freely accessible rating website following the Linked Data principles. The site archives ratings of every entity you could think of. All these entities are represented by URIs. The reviews, written by users, are linked against several websites on the Web of Data. An example of such a link is DBpedia [13]. DBpedia extracts information from the information boxes of Wikipedia websites and exposes this
data as RDF. For example, Revyu uses the abstracts from DBpedia as information about a movie or book and creates the ability of browsing the Web of Data. All pages are delivered in HTML format and in RDF. Developers from other applications are able to use the RDF-triples, create new links and combine it with their data, while ordinary users see the HTML views. The DBpedia information source is very important in the Linked Data community. It was the first dataset contributing to the Linked Data idea and enables a lot of other data sources to link against it [13]. All these datasets are bundled in the Linked Data cloud [2]. The cloud is a great source for finding interesting or related datasets to work with.

3.1.2 Linked Data at a company

The Semantic Web technologies, and especially Linked Data applications, are currently being used by several companies. One of the great Linked Data advocates is the BBC, the largest broadcasting corporation in the world [14].

The BBC publishes a large amount of its content online. The types of these contents can differ from audio and video to pure text. The BBC currently serves 8 national TV channels and 10 national radio stations. All the information on its website involves a relation with the topic subjected to the TV or radio show. The different types of information are all linked together with the help of Linked Data and its Semantic Web techniques. With this approach, the BBC is able to interlink information about TV or radio shows and show the user relevant information.

Back in the old days, the BBC made its data available through different feeds and through its own API (Application Programmers Interface, see section 3.3.1). This method enables developers to consume the BBC data and create mash-ups around it. Problem with this solution was that it is not centralized and it increases redundancy. This means that if the Dutch prime minister occurs in both a TV show and a radio show, we are not aware of the fact that this same person did this. By giving everything its own identification, by using URIs, we overcome this problem. It then becomes possible to interlink real world entities and to combine the knowledge we have about them.

The key benefits of using Linked Data content and Semantic Web technologies in this case are:

- User experience, having structured data apart from the view, which means that knowledge can be visualized in several different ways.
- Website is API, by exposing the content as RDF, the website is its own API.
- Design advantages, teams can do loosely coupled design, work on their own subject of interest.
3.2 Mapping data to RDF

In order to create a Linked Data dataset out of a regular relational dataset it is necessary to map this data to RDF. Relational datasets are based on an entity-relationship (ER) model [15], but the question is whether the RDF model is an entity-relationship model or not? The answer is both yes and no. RDF can be used as a basis for modeling an ER-model, but RDF can do more. RDF is a model of entities (nodes) and relationships. If a developer is used to the ‘ER’ modeling system for data, then the RDF model is basically an opening of the ER model to work on the web. A typical ER model has entity types (e.g. tables), and for each entity type there is a set of relationships (the links between the tables). The RDF model is the same, except that relationships are first class objects: they are identified by a URI, and so anyone can create one [16]. This means that it is not easy to convert a relational database to RDF automatically.

Current research is tackling this problem by converting or mapping relational databases to RDF. An example of such a project is D2R Map [17]. D2R Map is a declarative, XML-based language to describe such mappings [17]. The main goal of this project is to design and create a mapping between a database model and RDF without changing the original database model. This is done by incorporating SQL queries directly in the mapping and aggregating the results. This approach has the ability to map relationships and content to valid RDF triples.

Triplify on the other hand is a simplistic but effective approach to publish Linked Data from relational databases [18]. Triplify is based on mapping HTTP-URI requests onto relational database queries. Triplify transforms the resulting relations into RDF triples and also publishes this data on the Web as Linked Data.

3.3 Interlinking

To enrich your dataset with external information, the interlinking process with other datasets is very important within Linked Data applications. Several aspects need to be incorporated in this process. How can access to the data be achieved? What constraints or conditions are there when a matching is made? This section will discuss different techniques to achieve this.

For small datasets or RDF-files, published manually (such as an individual’s FOAF file\(^5\)), it is possible to create links manually [19]. There are only a few links to be created and it is easy to manually find out whether parts in other datasets are relevant to link against. However, doing so for large datasets is impractical; there should be a way to automatically detect the overlapping parts of heterogeneous datasets. Because within Linked Data datasets the knowledge is represented in a structure (RDF), it is possible to retrieve relevant data items from this structure by following the defined predicates (such as: writtenBy).

\(^5\)Example of Tim Berners-Lee’s FOAF profile: [www.w3.org/People/Berners-Lee/card](http://www.w3.org/People/Berners-Lee/card)
By investigating these relations, it is possible to create sophisticated discoveries [20].

### 3.3.1 Retrieving data

To start the process of discovery, it is important to first address the problem of accessing data. It is easy to understand that if the data is kept in a database with public access, machines are able to retrieve this data. This is, however, mostly not the case on the web. The following subsections will describe three different ways of getting access to data.

- Retrieve content via APIs
- Scraping content
- Use Linked Data

**Retrieve content via APIs** An API (Application Programming Interface) is a language and message format used by an application program to communicate with the operating system or some other control program such as a database management system (DBMS) or communications protocol [21]. APIs are implemented by writing function calls in the program, which provide the linkage to the required subroutine for execution. More and more APIs appear as a web service. When used in the context of web development, an API is typically a defined set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in XML or JavaScript Object Notation (JSON) format. While ‘web API’ is virtually a synonym for web service, the recent trend (so-called Web 2.0) has been moving away from Simple Object Access Protocol (SOAP) based services towards more direct Representational State Transfer (REST) style communications.

Web APIs allow the combination of multiple services into new applications known as mashups. It becomes possible to query external databases and use this information. An example of the use of a Web API is the GeoNames API to search for locations in GeoNames dataset, using a free text search engine. In this way it becomes possible to automatically find different locations in the GeoNames dataset (e.g. Berlin, Germany or Paris, France) and create connections between these locations in a dataset on the one hand and GeoNames on the other hand.

**Scraping content** As discussed in chapter 2, the structure of Web content is currently focused on human processing. A website consists of human-understandable images and data. This means that the content is not delivered to machines in a very structured manner. With a technique called scraping it is possible to convert this human-understandable data to machine-understandable data by finding the structure in the web page and retrieve the important parts. Scraping can be done automatically by scraping tools or manually by a developer constructing his own scraper. The scraping tools can be divided into four categories [22, 23]:

23
Languages for Scraping Development: one of the first initiatives for addressing the problem of scraper generation was the creation of a scraper language. These languages are proposed as alternatives for general-purpose languages (like Python or C). Different kinds of expressions to scrape content are embedded within the language and can be used in scraping web content (e.g., a call to a function called ‘scrapebold()’ (implemented in a scraping language) scrapes all the content within HTML-bold tags).

HTML-aware extraction tools: these tools understand the structure of HTML-documents. They build up a parse tree from the HTML-structure and perform extraction rules on this tree. These extraction tools are configurable.

NLP-based tools: natural language processing tools are often used in extraction tools. These tools apply mechanisms like filtering or semantic tagging to build relationships between phrases and sentences. These tools are more useful if a webpage consists full text pages.

Induction tools: these tools look like the NLP-based tools apart from the fact that they do not look at linguistic equivalences, but at structural equivalences. This means that these tools are more useful within HTML-pages.

It depends on the situation which of the above techniques is the most appropriate one to choose. It is most sufficient to explore the structure of the webpage and base the specific technique on that analysis [22].

Use Linked Data If datasets are published as Linked Data (e.g., DBpedia), it is fairly easy to create links to other datasets. An example is a page with information about the city of Paris (capital of France). The URI of this particular city is: dbpedia.org/resource/Paris. This page consists of a lot of information about the city. The RDF data of this resource can directly be accessed through the following URI: http://dbpedia.org/data/Paris. This page returns the RDF version of the data.

Because DBpedia is a Linked Data application, it also provides a SPARQL Endpoint (http://dbpedia.org/sparql). Via this endpoint it is possible to directly input SPARQL queries and retrieve RDF content. Developers are thus capable of querying an external dataset (like DBpedia) and asking intelligent questions to perform data retrieving.

3.3.2 Interlinking techniques

If a machine can access both the local data and the external data it wants to interlink against (using the described retrieving data techniques), it is now capable of creating the links. There are different automatic interlinking techniques to perform this task. This section will discuss how they work and what can be achieved with them.
**Manually restricted interlinking** If a search is performed on a dataset and multiple results are returned, it is hard to decide which result is the (most) appropriate one. One of the solutions to this problem is to define an extended search. In order to do this extended search, constraints on the resulting output can be created. For example, if a search for a person is performed, the type ‘Person’ can be a restriction. There are a lot of these types denoted in the Yago ontology [24]. Yago has the ability to restrict resources to be of a specific Yago type. However, even with restrictions on the search input, a string may not be discriminating enough. This approach is a manual approach, because it needs definitions for every individual type, and does require a lot of work.

**Automatic naive interlinking** The naive approach means that the first result of a query is the (most) appropriate result. The process is easy and the same method can be applied for different classes (e.g. papers or authors). For example, looking for ‘Persons’ with a SPARQL query, the approach works as follows:

1. Choose a person.
2. Use the SPARQL endpoint to find the same person in a different dataset (e.g. SELECT ?x WHERE { ?x foaf:name 'Person’s name' }).
3. The query returns the URI (if there is a URI, where ‘Person’s name’ is the name of the author).

A problem with this approach is that every misspelling or language ambiguity results in a miss-hit, because these queries only select the exact matches. Another problem with this approach is that it only works with the assumption that the record which is found, is indeed the correct record. This is not always the case because different types (e.g. Persons) can have the same name.

**Automatic graph matching** With automatic graph matching the structure of RDF is being used. This technique tries to disambiguate two artifacts by exploring relations. The RDF structure has the ability to explore relations (e.g. the books written by a person or the country a person lives in). The strength of this technique is that multiple relations can be used in order to disambiguate a search result and find the correct interlinking.

### 3.4 Conclusion

The Linking Open Data project (Linked Data) is a project started by the W3C and builds upon the Semantic Web ideas. This chapter showed what particular features a Linked Data application needs (also referred to as the Linked Data principles). It discussed real-world examples like Revy and the BBC, to show the benefits and its potential as well as technological issues needed to be solved in order to create a Linked Data application. A machine needs to have access to both the local and the external data in order to perform an interlinking.
The interlinking can be created with different approaches. With all this knowledge it should be possible to create a Linked Data application out of a legacy application.
Part II
Creating Linked Data
4 Different steps in transforming a (legacy) dataset into Linked Data

This chapter will discuss the following research question: what steps are needed in transforming a (legacy) dataset into Linked Data? In order to answer this question an example dataset has been used. The Software Technology Department of the TU Delft has a collection of over 1200 publications stored in a database. This information is used as reference material and stored in a local datastore. Bringing this information to the Web of Data as Linked Data would be beneficial. It would enhance the information structure and it can enrich the dataset, if links against other datasets like DBLP (Computer Science Bibliography)\(^6\) or DBpedia (the Linked Data version of Wikipedia)\(^7\) can be created. This chapter proposes a way to transform such a dataset to a Linked Data dataset and will discuss this step by step.

4.1 Introduction to the dataset

The Software Technology department has collected its publications over the years and created a web-based application in order to trace all these publications. The current application\(^8\)) contains publications from now back till 1978. In order to work with this dataset we created a dump (a snapshot of the data) which collects 1241 entries of different kinds of publications. The final goal was to create RDF out of the publications data, add links against other datasets to enrich the current data and then publish the data as Linked Data. This dataset will serve as a running example throughout this chapter.

4.2 Reverse engineer a database schema

The first step in converting relational data to RDF is to define a schema of the data. A collection of data does not say anything about the schema designed to work with the data. In order to decide what information is needed to be converted to RDF and how this data relates, a schema needs to be designed (and in the case of existing data: reverse engineered). The first step is, thus, to create a model, like an ER (Entity Relationship) model, which explains the relations between the data in a structured manner. In the ST Publications dataset case, we reversely engineered the database dump and created a diagram out of it. The schema mainly consists of two parts. The first part (Figure 3) describes all the administrative information which is needed to add the publications to ST Publications dataset. For example, it shows which group this person belongs to together with other administrative information.

\(^6\)http://dblp.l3s.de
\(^7\)http://dbpedia.org
\(^8\)http://publications.st.ewi.tudelft.nl/
Figure 3: The first part of the database schema: administrative information

The second part of the database (Figure 4) defines the core publications data.

Figure 4: The second part of the database schema: core publications data
The different types of publications saved in the dataset are the following:

- Master’s thesis
- PhD thesis
- InBook, a part of a book (a chapter or a range of pages)
- InCollection, a part of a book with its own title
- Booklet, a work that is printed and bound, but without a named publisher or sponsoring institution
- Book, a book with an explicit publisher
- Article, an article from a journal or magazine
- Proceedings, the proceedings of a conference
- InProceedings, an article in the proceedings of a conference
- Unpublished, a document with an author and title, but not formally published
- Manual, a technical documentation
- TechReport, a report published by a school or other institution, usually numbered within a series
- Miscellaneous, used when nothing else seems appropriate

The relations between the data were not explicitly mentioned in the database dump. This means that we needed to define these relations manually. An example of such a relation is a user (table ‘user_account’) adding a publication (table ‘entry’). The relations are being showed with black lines (see Figure 3).

4.3 Exploring corresponding schemas in the Linked Data cloud

After the structure of the data has been defined, a structure of the RDF (an ontology) can be defined. It is a wise decision to look at the Linked Data cloud in order to find related schemas. If schemas are being reused they become more trustworthy and complete, besides people will understand them quicker, because they are acquainted with the schemas [2]. In order to find schemas for our test dataset, we explored the Linked Data cloud for research related datasets. This is a short overview of relevant ontologies for the domain of scientific research communities:
- **Semantic Web for Research Communities (SWRC):** The SWRC ontology is an ontology for modeling entities of research communities such as persons, organizations, publications (bibliographic metadata) and their relationships. The SWRC ontology is used by DBLP. [http://swrc.ontoware.org/ontology#](http://swrc.ontoware.org/ontology#).

- **The Semantic Web Conference Ontology (SWC):** The SWC ontology is an ontology for describing academic conferences. It was initially designed to support the European Semantic Web Conference, ESWC2007, and later extended for both the following conferences in the ESWC series, as well as in the ISWC series. Used by Semantic Web Dog Food. [http://data.semanticweb.org/ns/swc/swc_2009-05-09.html](http://data.semanticweb.org/ns/swc/swc_2009-05-09.html).

- **Citation Oriented Bibliographic Ontology:** An ontology to describe scholarly citations. It covers three primary classes: events, agents, and bibliographic reference types. It is designed to offer a solid general relational model for citation metadata, and also to provide a specific superset of reference types in standard formats like BibTex, RIS, and Refer/Endnote. [http://vocab.org/biblio/schema](http://vocab.org/biblio/schema).

We took the decision to work with the SWRC ontology. The ontology is well established, has been used by a related and trusted dataset (DBLP) and defines the classes and properties suiting our data. With the ontology we can describe both the administrative data and the core publications data. An overview of the different classes and properties we used to describe our dataset can be found in appendix A.

### 4.4 Mapping the dataset on a schema to create RDF

At this point in the transformation process it is decided which ontology is going to be used to define the RDF data in. Now it is time to fit this data into the defined schema (create RDF) and see whether tools can do these things automatically or whether things should be done by hand. As discussed in section 3.2, there are some tools aiming at this specific task. For our ST publications dataset we worked and tested both tools described in section 3.2 (Triplify and D2R). There are, however, also domain specific tools which can create RDF. Because the current ST publications application is able to create BibTex files of the publications, we also looked at BibTex to RDF converters. As shown in Figure 4, there are 13 different types of publications stored in the dataset. All these types are suited for BibTex creation. An overview of some explored Bibtex to RDF is shown in Table 1:

---

9[http://dblp.l3s.de](http://dblp.l3s.de)
Table 1: Overview of different BibTex to RDF tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Link</th>
<th>Ontology used</th>
</tr>
</thead>
<tbody>
<tr>
<td>BibTex-2-RDF</td>
<td><a href="http://www.cs.vu.nl/~mcaklein/bib2rdf/">http://www.cs.vu.nl/~mcaklein/bib2rdf/</a></td>
<td>SWRC Ontology</td>
</tr>
<tr>
<td>Bibtex2rdf</td>
<td><a href="http://www.l3s.de/~siberski/bibtex2rdf/">http://www.l3s.de/~siberski/bibtex2rdf/</a></td>
<td>DC, DCT, VCard</td>
</tr>
<tr>
<td>Java BibTex-</td>
<td><a href="http://www.aifb.uni-karlsruhe.de/WBS/pha/bib/index.html">http://www.aifb.uni-karlsruhe.de/WBS/pha/bib/index.html</a></td>
<td>SWRC Ontology</td>
</tr>
<tr>
<td>To-RDF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We tested every tool, but did not choose any of them. The problem with the Bibtex2RDF converters is their lack of configuration. The tools work as follows:

1. Add a BibTex file (with 1 or more publications included).
2. The ontologies these tools work with are predefined (see Table 1).
3. The BibTex file is converted into this ontology. All the URIs are predefined. This means that every publication itself gets a URI, the authors get a URI and the publisher gets one.

There is nothing we can adapt or configure in this process. Because it only converts the BibTex file, we will lose other information which was available from the dataset (the research group publishing the article, the person adding the article in the system, the project from which the paper originated and some extra help fields available in the database).

We therefore chose to work with the mapping tools discussed in section 3.2. The Triplify tool took a lot of work, because every entity had to be defined manually. This means, for example, that we manually had to define 13 different mappings for 13 different types of publications. If you have a dataset with a small amount of entities, the Triplify tool may suit well.

Because of the amount of manual labor, we chose for a semi-automatic mapping, using D2R. With this mapping we are able to create the relations between the tables (based on the ER-diagram we reversed-engineered). Because D2R does a lot of work in advance, we only had to adapt the generated mapping, which is a great advantage opposed to Triplify. With D2R we are also able to assign our own URIs to ‘things’ (and specify these ‘things’ ourselves). Overall this approach is more flexible and easy to use. Appendix B shows a sample of our mapping with an explanation of the way it works.

### 4.4.1 Decisions during the mapping process

As explained, we used the SWRC ontology to describe the ST Publications data and we chose the D2R tool to convert the dataset into RDF. Because D2R creates its own ‘fictional’ ontology, we had to define the predicates of the chosen ontology manually. There are a lot of difficulties when converting a relational
database to RDF if the automatic mapping does not fit your exact needs. One of the decisions important to make is which database field you choose to function as a label for a specific class. Every class in RDF needs a label to define what the title of that certain class is. Fields in the database that are not always filled, may not be the right ones to choose as labels. It is, however, possible to add conditions to your mapping which say that a specific class can only be created if the label field is not empty.

Another decision we had to make was which parts of our data we were going to give a URI. As explained in section 3.1, URIs are needed for people to find and locate things. They are also very important in the interlinking process. We can only create a link to an entity, if that entity has a URI. In our dataset we wanted the authors to have their own URIs. The original dataset, however, bundled the authors in one field (e.g. ‘A. Bos and A. van Gemund and C. Witteveen’), which means that we were only able to assign a URI to this bundled author field. This means that this field (‘A. Bos and A. van Gemund and C. Witteveen’) consists of three potential URIs. If we are able to create three different URIs, we are able to distinguish the different authors and to create one uniform method as a pointer to these authors. We then also have the ability to create useful data browsing. By following the URIs we can show all the papers written by this specific person or show all other documents he or she was involved in. The D2R mapping tool however does not have the ability to split this database field and create three separate URIs. We therefore added some extra information to the database and created a separate authors table. A script splits the authors string and adds the authors to a new authors table which solved our problem.

Because we created a mapping to individual authors instead of author sets, we are able to visualize an example of a new problem within the ST database, referred to as the ambiguity problem. Because the names of the authors are added by different people, spelling mistakes or different forms of spellings often occur. Figure 5 shows an example of 5 different URIs for the same author (‘Arjan J.C. van Gemund’). This shows that our first attempts in creating URIs are not worth the name unique.

In order to do better than this we had to do an extra analysis on this situation. We created a way to enrich the dataset by merging the same authors and create a single URI per author. We will elaborate in detail on this technique.
in chapter 5. After this step we have a mapping in RDF and we are at the point where we can enrich this dataset through interlinking.

4.5 Interlinking entities

This step is about interlinking entities. If there is RDF data available (which is the case after the mapping step performed before), the RDF data can be linked against other (RDF) datasets. It is important to know what kind of data is needed in order to enrich a dataset. It is also important to see which types of links are useful to create (examples of these links are: ‘same as’, ‘see also’). For our ST Publications dataset we looked at both the Linked Data cloud and at ordinary datasets, to see whether they had interesting and useful information for us. Some examples of these datasets are:

- Within Linked Data cloud:
  - DBLP Hanover, up-to-date version of DBLP Linked Data edition [http://dblp.l3s.de/d2r/](http://dblp.l3s.de/d2r/), useful to link authors or publications against, because the great amount of trustworthy and relevant data in the DBLP dataset.
  - CiteSeer, IEEE and ACM collection of citations [http://www.rkbexplorer.com/](http://www.rkbexplorer.com/), has lots of material available with more information about the publications than in our own dataset.
  - GeoNames [http://www.geonames.org/](http://www.geonames.org/), a lot of information about geographical data. Interesting to link the locations of our publications against.

- Ordinary datasets (websites) to link against:
  - Bibsonomy [http://www.bibsonomy.org/](http://www.bibsonomy.org/), a social bookmark and publication system with a lot of data.
  - Twitter [http://www.twitter.com/](http://www.twitter.com/), has a lot of real-time information available about the researchers.

4.5.1 Start interlinking

Before the interlinking process starts, it has to be decided whether or not the interlinking process can be done manually. As an example, in the case of our ST Publications dataset, we wanted to interlink the authors against the authors in DBLP and the locations entities against locations within GeoNames. Within the ST Publications dataset there are 4000 different author entities and over 500 different locations of publications. These amounts of entities are hardly possible to interlink by hand.
To find corresponding entities in other Linked Data datasets, it is obvious to use SPARQL queries. An example of such a SPARQL query is shown below in Listing 4.

**Listing 4: SPARQL query to find same author entity in DBLP**

```sparql
SELECT ?x WHERE {
  ?x foaf:name "Geert-Jan Houben"
}
```

The query tries to match every author (in the DBLP dataset) named ‘Geert-Jan Houben’ and returns its URI. This query can be processed for every individual author to find all corresponding URIs. Problem with this query is that it will only find authors with exactly the same name.

To overcome the problem of differently spelled author names, SPARQL has the capabilities to search in parts of strings. The SPARQL query in Listing 5 returns the URIs of every author with the string ‘houben’ in it.

**Listing 5: SPARQL query to find same author entity in DBLP with a filter**

```sparql
SELECT ?x WHERE {
  ?x foaf:name ?name FILTER regex(?name, "houben", "i")
}
```

The results can then be further elaborated to decide which author is the one you were looking for. Chapter 5 will show methods to do this.

Another method to perform interlinking is by using scraping techniques. We used scraping to retrieve the authors found by the DBLP search engine. The query denoted in the footnote\(^{10}\) retrieves all the authors with the keyword ‘houben’ in it. Because DBLP does not deliver an API to retrieve this content in a structured format, we created a scraper to extract this information. The HTML version of this page shows the exact information needed to perform this action. Because every author is mentioned in an HTML list (<li>, list items) and there is only one list in the page, we are able to retrieve all the authors from this list. With this method we are able to use the DBLP search engine in our own application, by showing the results belonging to this smart keyword search engine.

The third method we used, was an API. In order to interlink the locations of the publications in our dataset against the locations in GeoNames, we used the GeoNames API\(^{11}\) in our advantage. The locations can be passed through a REST API which delivers the corresponding results in XML format. Every location found by GeoNames has a corresponding GeoNames URI, which we could use to create links.

\(^{10}\)http://dblp.uni-trier.de/search/author?author=houben

\(^{11}\)http://www.geonames.org/export/
4.6 Publish the RDF dataset

Recapitulating this chapter the following steps were performed: we reversely engineered a schema of the ST Publications dataset and we then found an ontology fitting our data. Next we mapped our data to RDF, using the schema we defined and using mapping tools. We then found datasets to link our data against and created the links we were interested in. The Linked Data principles define the following rules:

1. Use URIs as a name for things.
2. Use HTTP URIs so that people can look up those things.
3. Provide useful information, using the standards (SPARQL, RDF), when someone looks up a URI.
4. Include links to other URIs, so that people can discover more things.

This means that the last step we need to do is the publishing part. Within Linked Data applications the accepted standard is to publish the data both as HTML (in order to create a view for users with a HTML-browser) and as RDF (to access the data and in order to browse the data with a Semantic Web browser). The D2R tool, as explained in section 3.2, is automatically capable to present both these HTML and RDF views.

4.7 Related decisions

There are some decisions we had to make during the process, we did not yet introduce. When it is clear what the structure of the data is and which ontology is being used for the RDF, it becomes necessary to decide which parts of the data are being converted to RDF and how this will be done. At the moment, the SPARQL language is not as powerful in terms of speed as known database languages (like SQL). This is because databases use smart indexing techniques to speed up the search which are not applied in most SPARQL stores. Because of this lack of speed, it could be smart to convert only parts of the data to RDF and keep other parts in relations (the creators of LinkedGeoData used this idea to achieve better performance [25]). With our publications example we chose to convert everything, because the total amount of publications is not overloading a SPARQL endpoint.

Another decision needed to be taken is whether the data is converted beforehand or on request (e.g. create the corresponding RDF when someone tries to request an author or a paper). When there is a lot of data to convert or when API data is being converted (RDF Book Mashup is an example of such a Linked Data application [26]), it might be smart to convert the data on request. When a user requests a page the RDF is being created on-the-fly. In our case the decision we took, was to convert the data beforehand. Because we have a copy of the data in a database, we are able to create real-time RDF versions. Because the adding and editing frequency is not that high, we are able to maintain a one-on-one copy of the database in RDF.
4.8 Conclusion

This chapter proposed a way to transform a (legacy) relational dataset into a Linked Data dataset and create a Linked Data application with it. It showed what steps are necessary and which decisions need to be made in this process. The first steps are creating a model of the data and turn this data into an ontology. The second step is to map the data in this chosen ontology and create RDF. After this step the interlinking process starts. The interlinking process has a lot of difficulties which will be explained in chapter 5. After the interlinking process is finished, the publishing step is the last step in the creation of a Linked Data application. There are some tools available which automatically publish the data for you.
5 Optimize the automatic interlinking process

This chapter will answer the question how automatic interlinking between entity URIs in Linked Data datasets can be optimized. An often occurring problem within a dataset is the fact that the same ‘thing’ is represented as different ‘things’. This means that it is not possible to decide whether or not these things are the same or unique. In order to optimize the automatic interlinking process it is necessary to first create URIs. In the ST Publications dataset for example, we have this ambiguity problem with different authors. We need to know which author is a unique person and which authors are in fact the same person. After the URIs have been created we can perform the interlinking. This chapter will explain a sophisticated technique we created and used with the interlinking between authors in the ST Publications dataset and authors within the DBLP dataset.

5.1 Disambiguating authors to create URIs

In order to create URIs for the authors in our dataset we had to identify the problem. We illustrate the problem with the following collection of publications:

1. C. Roos and Arjon van Gemund 'Interior-Point Methods for Linear Optimization'

2. N. Roos and Dick den Hertog: 'A survey of search directions in interior point methods for linear programming'

3. A. van Gemund, P. Zoetewij and R. Abreu 'A New Bayesian Approach to Multiple Intermittent Fault Diagnosis'

4. Arjan van Gemund and Rui Abreu 'A Low-Cost Approximate Minimal Hitting Set Algorithm and its Application to Model-Based Diagnosis'

Identifying the problems in this specific case, we see two different problems occurring:

- **Identification problem**: for example, the real-world entity ‘Arjan van Gemund’ shows up in different forms: ‘A. van Gemund’ and ‘Arjan van Gemund’. There can also be misspellings in the dataset: e.g. Arjon van Gemund.

- **Disambiguation problem**: if there are very similar representations, these representations need to be distinguished. Can it automatically be distinguished whether or not ‘C. Roos’ of the first paper is the same person as ‘N. Roos’ from the second paper?

In order to resolve the entities (authors in this case) the relations between the entities can be explored. If we look at the co-authors of the specific authors we see that both ‘A. van Gemund’ and ‘Arjan van Gemund’ worked with ‘Rui Abreu’. This makes it more likely that these two persons are indeed the same.
The problem is that we need to know that this ‘Rui Abreu’ is one and the same person (which created a cycle). Heuristics are needed to start this process. So let’s state that ‘Rui Abreu’ is indeed one person (because the name ‘Abreu’ is rare, we could say this with higher certainty than with an often occurring Chinese name like ‘Wang’ or ‘Li’).

String matching algorithms are needed in order to define a measure of similarity between strings and solve the identification problem. We will then use these string similarity measures and present two ways to resolve the entities and work on disambiguation: attribute-based and relational-based entity resolution.

5.1.1 Approximate string matching

The similarity of two strings can be measured by approximate string matching algorithms. This is needed to solve the identification problem. These algorithms assign values to pairs of strings, expressing the amount of similarity. For instance, the words ‘similarity’ and ‘dissimilarity’ are likely to get a score indicating that they are very similar since they differ just three characters. The following pair of words ‘similarity’ and ‘distance’ will probably get a bad score. This score depends on the algorithm that is applied. There are some basic concepts in string matching. Each concept has its own variations. The variations and basic concepts will briefly be discussed.

Character-based methods One of the character based methods is the Levenshtein distance (or edit-distance) [27]. This method measures the minimal number of insertions, deletions or substitutions that are needed to transform a string ‘str1’ into a string ‘str2’.

The Jaro distance is expressed in the number of matching characters and the number of swapped letters. The Jaro-Winkler distance is a variant and has a higher weighing factor for prefixes. There are even more variants possible. Take, for instance, matchers that assign penalties to mismatched characters or bonus points to matched characters. Each variant favors some kind of type errors or performs best on a particular range of length. One variant gives less penalty for type errors that occur often (like typing an ‘m’ instead of an ‘n’ and vice versa) and less penalty for the lack or presence of accents on letters (as for ‘c’ and ‘ć’). So depending on the kind of dataset, a different method can be chosen.

Character-based methods can also be applied to strings that consist of more than one token. This string will then be treated as if it is one token, e.g. a space character is just seen as a normal character. This comes with some problems that token-based methods can solve (e.g. word order).

Token-based methods Token-based methods measure the number of matching tokens between two sets of tokens [27]. The Jaccard measure is the simplest example. It measures the ratio of equal tokens in the union of tokens of both strings. A disadvantage of this method is that every word has an equal weight.

TFIDF (Term Frequency / Inverted Document Frequency) is a method that comes from the information retrieval world. This method measures the fre-
quency of a term but also corrects this with the importance of the token. This means that common tokens like ‘a’, ‘the’ and ‘but’ get a lower score because they are not discriminative enough.

The cosine similarity expresses the different strings as term vectors, with each word being a dimension in the vector, counting the frequency of this word. The cosine similarity then measures the angle between the vectors, which is a measure for the similarity between the strings. Unfortunately, the cosine method does not take misspellings into account, which means that misspellings can decrease the similarity score.

Hybrid methods  Bigrams, or, more general, Q-grams can overcome the problem of ignored misspellings by dividing each token in every possible sequence of characters of length q. Now, only the tokens containing the misspellings are being rejected as match. On this new set of tokens, other token-based methods can be applied. There is another hybrid method. Soft-TFIDF (hybrid TFIDF) with Jaro-Winkler performs best on different kinds of datasets [27].

The token-based method will calculate a score based on the number of similar tokens and the character-based method determines if two tokens are similar. We will now present two ways to resolve entities: attribute-based entity resolution and relational-based entity resolution.

5.1.2 Attribute-based entity resolution

Now we know how to measure the string similarity, we are able to work on the disambiguation process. In this approach a similarity \( \text{sim}(r_i, r_j) \) is computed for each pair of entities \( r_i, r_j \) based on their attributes (e.g. \( r_i = \text{Arjan van Gemund} \) and \( r_j = \text{A. van Gemund} \)). The similarity function defines a similarity between the two entities based on a string matching algorithm. Only the pairs with a similarity above some instantiated threshold will occur. For different attributes there are different string similarity metrics available. A weighted combination of the similarities over the different attributes can be used to compute the total similarity. This method is based on syntactical similarity. This means that we can decide that ‘Arjan van Gemund’ and ‘A. van Gemund’ are the same person and with even more certainty we can state that ‘C. Roos’ and ‘N. Roos’ are the same. This, however, does not say anything about the semantic similarity.

5.1.3 Relational-based entity resolution

Related references can also be used as additional attributes for matching. In our case we can look at the co-authors of the specific authors. We have two main ideas. The first is based on string matching, without any reasoning. The second idea is a collective approach. With this approach the resolve step is also done for the co-authors. This last approach we used in our own implementation, which will be discussed in section 5.3.
5.2 Interlinking with tools

As explained, a lot of knowledge within a dataset could be connected to the Linked Data cloud in order to enrich the dataset. To do this, data instances, such as authors, publishers and locations, need to be linked to external resources. In this process a semantic disambiguation problem surfaced: a lot of the data instances in the original relational database are ambiguous and need to be disambiguated in order to perform the interlinking. We looked at two different interlinking tools to see whether they are helpful in solving our problem.

5.2.1 Different tools to interlink

In a Linked Data context there are only few tools which actually focus on interlinking entities and create relations between them. At the moment there are two tools focusing on exactly this problem: LinQuer [28] and Silk [29]. Both tools match different data instances according to syntactical similarities, based on a string distance.

Silk The Silk tool is a Linked Data-focused solution that works over SPARQL endpoints. Silk features a declarative language for specifying which types of RDF links should be discovered between data sources as well as which conditions data instances must fulfill in order to be interlinked. Link conditions apply similarity metrics, like methods based on string, numeric, data, URI, and set comparison, to data properties. Metrics evaluate to similarity scores, which can be weighted and combined using aggregation functions. Silk accesses data sources over the SPARQL protocol and thus can be used without having to replicate datasets locally.

The user specifies the type of resources to be linked and the comparison technique to be used. Silk uses many string comparison techniques, numerical and date similarity measures to compare dates, concept distances in a taxonomy and sets similarity. Transformations can be specified in order to prepare the dataset before the matching process, in order to increase the process efficiency. Silk takes as input two Web datasets accessible behind a SPARQL endpoint. It can output owl:sameAs links or any other RDF predicate specified by the user. The tool is implemented in Python.

LinQl The Linkage Query Writer (LinQuer) is a system for generating SQL queries for semantic link discovery over relational data. It works by translating LinQl\(^{12}\) queries to SQL queries via a Web interface or API. A link specification defines the conditions that two given values must satisfy before a link can be established between them. To create such links, the framework provides several different methods like ones based on synonyms, hyponyms, and a variety of string matching methods. The native string matching methods are based on string similarity predicates that can be implemented in SQL (at this moment they support Weighted Jaccard and the Token Intersect method). In order to

\(^{12}\)http://linquer.linkedct.org:8080/linq1/doc.html
use LinQuer, you need the data in a relational format. The tool was used to interlink the Linked Internet Movie Database\textsuperscript{13} to DBpedia [30].

5.2.2 Comparison between the tools

This section will compare Silk and LinQuer. Both tools are offering a somehow related feature: they have the ability to define a global model to interlink entities. There are, however, a lot of differences between the two tools. We created comparison metrics and analyzed the two tools according to these metrics.

Comparison metrics In order to decide which tool we could use best, we created the following different comparison metrics to compare the different tools.

1. Degree of automation
   - Is the tool fully automated or does it work semi-automatic?
   - If it is semi-automatic, how should the user parametrize the tool and how efficiently does that work?
   - Which tasks does it perform automatically?

2. Different matching techniques
   - Does it use string distance metrics and which?
   - Does it use semantic metrics (explore WordNet\textsuperscript{14} for example)?
   - Does it use external functions (date comparison)?

3. Ease of use
   - Is it easy to use the tool?
   - Is it a lot of effort to learn how to use the tool?

4. Input
   - What underlying technique is used as input?
   - Is there a way to use other types of input?

5. Output
   - What kind of links does the tool generate?
   - Can these types of links be specified?

6. History and usage
   - Which datasets are merges via which tool?

\textsuperscript{13}http://www.linkedmdb.org/
\textsuperscript{14}http://wordnet.princeton.edu/
7. Extra features and differences

With help of these 7 criteria a comparison was created, every criterion will now be discussed.

**Degree of automation** Both tools can be defined as semi-automatic. They use their own specification language in order to work and create the links. Silk uses the Silk-LSL (Language Specification Language), which is an XML-based language. The creators of Silk worked on an easy to read manual\(^\text{15}\) which makes it easy to understand the Silk-LSL language for programmers. The LinQuer tool, calls its specification language: LinQl (Linquer Query language). This language is based on SQL and has explicit mentions for the different relations and link types. The developers of the tool created a wizard for LinQl queries which makes it easier to start creating queries. However, creating sophisticated queries is harder with LinQuer.

**Matching techniques** Silk defines several different matching techniques. The syntax similarity metrics are: Jaro, Jaro-Winkler, Q-grams (Q=2) and string equality. Silk also supports external functions like: numeric similarity, date similarity, URI similarity and set similarity. With these external functions it is not necessary to rewrite XML dates (for instance to an obvious format), because Silk can do this for you. The LinQuer tool offers way less similarity metrics. Because it works over SQL, the metrics need to be written in this language. The creators of the tool offer the following syntactical similarity metrics: weighted Jaccard, token intersect and cosine. The creators of LinQuer also claim to support semantic matching techniques like synonym and hyponym (by expanding WordNet). However, these metrics are not working at the moment.

**Ease of use** Both tools are easy to use. The Silk tool is harder to use, because it needs some manual install labor in order to use it. Because the package offered on the website uses different libraries, the libraries need to be installed manually. A lot of these libraries only work well with a Unix system, which means that Windows users have to do extra work in order to start the tool. The Silk-LSL language itself is understandable and easy to read. The LinQuer tool is somewhat easier to understand, because of the query wizard the programmers created. Because it works over relational data it is straightforward: just mention the columns you want to interlink, choose your matching technique and create a LinQl query. The SQL query created by LinQl can be used to discover the relations. The RDF needs to be created manually.

**Input** Silk totally focuses on datasets in RDF. People can only use it via SPARQL endpoints. This makes it easy to use in a Linked Data context. More and more datasets are being offered via SPARQL and can thus be accessed with Silk. The LinQuer tool uses relational database management systems and currently supports: MySQL and IBM DB2. The problem with this is that external datasets are way less available in a relational format than RDF dumps or SPARQL endpoints.

**Output** Both tools can create any kind of links. We have to imagine, however, that in most of the cases we are looking for owl:sameAs relations. We are

\(^{15}\)http://www4.wiwiss.fu-berlin.de/bizer/silk/spec/
working with string similarity metrics in order to find equivalence and therefore create `owl:sameAs` relations. It might also be the case that the links are working in a broader sense (`skos:broader` or `skos:narrow`) or in a related sense (`rdfs:seeAlso`).

**History and usage** Silk’s paper discusses the possibility of creating links between several different kind of datasets. It shows examples of the creation of links between DBpedia and the Linked Movie DB, Drugbank and GeoNames. On the other hand, the LinQuer tool is currently used for the creation of links between the Linked Movie DB and DBpedia.

**Extra differences** With Silk it is possible to aggregate your comparison results, which makes your results better. You can choose to maximize or average your search. Silk is also able to generate RDF files with the end results. You can specify different types (N-Triples, RDF/XML or N3) and Silk is implemented in open-source Python. LinQuer is on the other hand very fast (because of the indexing techniques used by the RDMSs). Its biggest problem is that the result needs to be mapped to RDF and the source code is not open.

A summary of the results is shown in Table 2:

<table>
<thead>
<tr>
<th></th>
<th>Silk</th>
<th>LinQuer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree of automation</td>
<td>Semi-Automatic</td>
<td>Semi-Automatic</td>
</tr>
<tr>
<td>Matching techniques</td>
<td>A lot</td>
<td>Few</td>
</tr>
<tr>
<td>Ease of use</td>
<td>Fairly easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Input</td>
<td>SPARQL</td>
<td>RDBMS</td>
</tr>
<tr>
<td>Output</td>
<td>Any type</td>
<td>Any type</td>
</tr>
<tr>
<td>History and usage</td>
<td>Few</td>
<td>Few</td>
</tr>
</tbody>
</table>

Table 2: Comparison between interlinking tools

In a Linked Data context it should be obvious that Silk is the best tool to use. Not only because of the fact that it is open-source, but also because it works over SPARQL endpoints (which are way more trivial than open relational databases) and has a large diversity of string similarity metrics. All these advantages compensate the speed benefits of LinQuer.

### 5.3 Our optimized interlinking method

Now we get back to the optimization of the interlinking process with help of the Silk tool. After the first mapping step in our transformation process, the D2R tool generates URIs for every author automatically. In the second step we enter the disambiguation process to overcome the problem of multiple URIs for a single author. Here you have to detect the different author names that represent the same author in the real world. It enables you to attribute one unique URI for the same author that was spelled differently in the original dataset. For example ‘Arjan van Gemund’ can have the following URIs (generated by the D2R mapping) after the first step: `<http://publications.com/authors/Arjan_van_Gemund>`
or <http://publications.com/authors/A._van_Gemund>, while both URIs represent the same author.

In order to reach a clean solution it is necessary to detect that these two URIs represent the same author and convert the existent URIs into one single URI. To match the same authors we need to compare authors’ names against each other, using string similarity algorithms, such as JaroWinkler. Performing these comparisons leaves us with syntactically related author names. However, this information is not enough. It occurs that different authors have syntactically related names while they do not belong to each other. An example from the publications database shows that ‘C. Roos’ and ‘N. Roos’ are syntactically closely related, while these are two different authors with even different working fields. Fortunately, by exploiting the semantic relation among authors it becomes possible to resolve the data instances reaching a more precise matching among them. For example, if two different authors, having very similar names, share the same co-authors, then there is a very high chance that they refer to the same person. This paper describes a tool, which is able to exploit the semantic correlation among the data in order to consolidate similar instances.

5.3.1 Architecture overview

The choice is made to extend the Silk framework, because it is open-source and uses SPARQL endpoints as its data access mechanism, which makes it easy to use in a Linked Data environment. The Silk framework enables the user to perform string similarity algorithms over RDF data and outputs RDF files with the matched results. Silks metrics are used to perform similarity comparisons, while another extra layer has been added to do the semantically related comparison. This layer is called the semantic layer\(^ {16}\).

Basically, Silk queries two SPARQL endpoints and compares, by applying string matching, resources from one source endpoint against resources from the other target endpoint. As result it produces a set of triples that connect one resource from one endpoint to another. In most of the cases it uses the property owl:sameAs for representing these links. In order to increase the precision of this matching process, the extension proposed in this paper adds a second level of comparison, where sub-properties of the resources matched in the first string matching can be used to disambiguate these resources. For example, suppose that you are looking for all co-authors of a specific author, you are able to get all of them with a SPARQL query. After you have collected all the co-authors you can use the similarity algorithms defined in Silk to find out whether these co-authors agree with the co-authors of the original author in the source dataset. In this way you are able to check and verify your results and be certain about the links you create between them.

In our example it is the case that both the source dataset as well as the target dataset is the same, because we are disambiguating our own dataset. The tool is of course well suited to link data instances to another dataset. After the first

\(^{16}\)http://linkedentities.googlecode.com - source code of the approach, together with other code discussed in the thesis
syntactical phase, Silk will leave us with the following results according to the example of Figure 6:

1. Arjan van Gemund
2. A. van Gemund
3. Arjan Gemund
4. Ariaan van Gemond
5. Arjan de Goed

Within these results we have both correct matches and incorrect matches, which should be solved by the semantic matching phase. As suggested a SPARQL query can be defined to find the co-authors of these authors in order to find this semantically related information. These SPARQL queries return lists with co-authors that are automatically compared against each other using string similarity algorithms. The user is able to specify the number of co-authors that should be matched in order to approve the overall match result. The last step is about converting the different URIs for the same person to a single URI. Figure 6 shows an example.

Figure 6: An example of the disambiguation of three different URIs that can be mapped to the same person.

This step is fairly easy, because we already know which authors are the same. As a heuristic the longest author name is chosen for forming the author’s URI that will be used in all instance that refer to the same author.

5.3.2 Link Language Specification extension

To use Silk in the proposed way, its link specification language (Silk-LSL) should be extended. The Silk-LSL language is an XML-based language with a coherent structure, described in a well-documented manual. The semantic matching functionality arises after the ordinary linking phase and has a similar structure. In the code fragment below (Figure 7) a new Silk-LSL file is defined which matches authors from a local publications database against each other in order to find the authors that are the same.
Figure 7: Code fragment of the Silk-LSL file with the semantic matching extension
Silk starts looking for all the authors in publications dataset, index all the author names (Figure 7, line 22-31), and match them against each other with a sorted Winkler similarity algorithm (line 38-43). It returns all the authors with a matching percentage above 80% (thresholds values on line 46). All these results are verified, in an additional step, by the semantic matching layer (starting from line 48). It performs two SPARQL queries which find all the co-authors related to the authors found by Silk. If at least two of these co-authors are the same (according to again a sorted Winkler similarity algorithm with a threshold of 85%, see line 48) then the matching result will be accepted. All these thresholds are found by a trial and error phase together with the test dataset discussed in the following section.

5.3.3 Analyzing the results

The original Silk tool allows us to compare strings and solves our scenario partially. However, these results are based on syntax only and do not have the precision we would like them to have. We analyzed the semantic matching phase of our proposed solution and compared our matching results to the matching results we obtained using Silk, to see which performed best in terms of precision. This section shows that the results are good and beneficial in solving the problem. We use the term match as a pair of authors which are the same in real-life.

Experiment setup In order to test the precision of our matching tool, we have put together a manual matching process that included all the correct data instance matches. This means we created an RDF file with all the owl:sameAs relations between all the authors in our local publications dataset. These relations are relevant to express the matching authors. We manually compared 715 authors in the dataset and added the correct matching authors to our RDF file. With this handcrafted RDF file we are able to compare the precision and recall of both Silk and our semantic matching extension.

The datasets As explained, we used a handcrafted RDF file to compare the matching results and check whether the extension has a higher precision than the original Silk tool. The ST Publications dataset consists of 1200 authors divided over 10 different BibTex entry categories (inproceedings, proceedings, book, inbook, booklet, manual, mastersthesis, phdthesis, techreport and unpublished). We used the authors from the inproceedings papers, which are 715 authors in total. This set is a combination of several different strings, where duplicates can exist. An example of this dataset looks as follows:

Authors = 
{‘Arjan van Gemund’, ‘Geert-Jan Houben’, ‘Jan Hid-
Jan’ }

We then manually compared all these authors and looked at their name equivalence, the work they created and their co-authorship. From these comparisons we derived a second dataset in which we set all the author matches
manually. All the authors which are the same are bundled together in a subset. This set, \text{CorrectMatches}, looks as follows:


This second dataset enabled us to compare the set created with Silk and our extension of Silk to the correct results. So we had three different datasets to compare: the handcrafted dataset (\text{CorrectMatches}), the results of Silk (\text{SilkMatches}) and the results of our extension (\text{Silk+Matches}). We were then able to derive the total amount of correct links or total amount of false links of both Silk and our extension of Silk. Because we could count the total amount of author matches generated (correct and false matches) we were able to determine a precision and recall of our extension. We make use of the following formulas to determine this:

\begin{align*}
\text{Precision} & = \frac{|\text{TotalCorrectLinks}| \cap |\text{TotalGeneratedLinks}|}{|\text{TotalGeneratedLinks}|} \\
\text{Recall} & = \frac{|\text{TotalCorrectLinks}| \cap |\text{TotalGeneratedLinks}|}{|\text{TotalCorrectLinks}|}
\end{align*}

The total precision and recall scores show the quality of the tool in percentage of completeness (amount of matches found) and correctness (amount of correct matches found). Note that these formulas do not incorporate the total amount of incorrect links.

5.3.4 Determine threshold values

To be able to work with our extension we will have to determine precise threshold values. These threshold values determine whether the system will detect author names as a correct match. Together with these threshold values it is important to determine which string distance metric is appropriate to use.

In our experiment we are comparing author names (names of people). There is a lot of literature about the problem of determining equivalence between author names and disambiguation of author names [31]. It showed that the JaroWinkler algorithm and the TF/IDF extraction algorithm work very well. This, however, depends on the language and terminology used for the different author names. If abbreviations are used a lot (‘A.J.C. van Gemund‘), JaroWinkler works well. If the order of the names differs (‘Gemund, Arjan van’ against ‘Arjan van Gemund’) a tokenized algorithm is more appropriate.

After a lot of tests and trial-and-error we chose an hybrid version of the JaroWinkler algorithm: sorted JaroWinkler. This algorithm uses the JaroWinkler algorithm as native string comparison method, but it also looks at the same string in a different order. This algorithm turned out to work well, because the names in the ST Publications dataset often start with either the first name or the last name of an author.
The threshold value is a value between 0 and 1. It determines the precision of the algorithm and it will acknowledge the matching process when it meets this threshold value. Determining the correct and perfect threshold depends on different aspects. We created an automatic trial-and-error script which checks the outputting results of a small dataset against the perfect matching process of this small dataset. This yielded the threshold value we are working with in our experiment.

5.3.5 Experiment results

The handcrafted dataset (CorrectMatches) showed that 414 authors of the original 715 authors matched. Using a sorted JaroWinkler comparison with a threshold value of 0.8, Silk was able to find 120 correct matches and 77 incorrect matches, which means a precision of $\frac{120}{197} = 61\%$ and a recall of only $\frac{120}{414} = 29\%$. However, the semantic matching extension reached the precision of 100% and a recall of 82% (shown in Table 3).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total matches</th>
<th>Correct matches</th>
<th>Incorrect matches</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>CorrectLinks</td>
<td>414</td>
<td>414</td>
<td>0</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>SilkMatches</td>
<td>197</td>
<td>120</td>
<td>77</td>
<td>61%</td>
<td>29%</td>
</tr>
<tr>
<td>Silk+Matches</td>
<td>340</td>
<td>340</td>
<td>0</td>
<td>100%</td>
<td>82%</td>
</tr>
</tbody>
</table>

Table 3: Precision and recall scores of Silk and the extension of Silk

There are some matches missing in our extension. This is due to the fact that authors do not need to have co-authors. This leaves us with an empty co-author dataset and therefore our Silk extension was unable to discover the matches between these authors. If we alter the Silk extension and accept matches without co-authors (based on syntax), the extension will find more matches, but the correctness will decrease. To overcome this problem it would be beneficial to choose another metric, for example looking at the genre of the specific paper or compare the dates on which these authors were publishing their material.

In order to perform these semantic matching tests it is necessary that the semantic knowledge you want to use (in this case co-authors or like we proposed genre) needs be located in both the source and target dataset.

5.3.6 Merging entities

Our extension of the Silk framework delivers a set of triples containing authors which are specified as owl:sameAs relations. This means that the matching authors have to be merged and one specific URI for all of them needs to be chosen. In order to do this work correctly, we followed different steps to merge the URIs, find the most suitable one and finally rewrite the old URIs in the newly chosen one.

The file created by Silk looks like this:
It can be concluded from the example that there is a transitive relation between the different entities. In this example it is the case that Geert-Jan_Houben is the same as Houben%2c_G.J. and the same as Houben. This also yields the fact that Houben%2c_G.J. is the same as Houben. In order to select an appropriate URI for these entities we have to find all equalities and look for an appropriate URI in this set as the URI for all entities.

We decided to parse the previous file and move the triples into a relational database to improve speed. We then had to select all the equal triples and put them together in an array. This work had to be done iteratively, because a table join would not work. We could have cycles in our transitive relation and we did not know when the combinations would stop. With a first iteration we selected the first object and subject which are the same and worked with them to select all other objects and subjects which are the same. At the same time we collected all values in an array and we forgot the duplicate ones. After all authors were found, we moved on to the next author. After this step we had a 2-dimensional array with in each array field a collection of matching authors.

The last step was to write a method which is able to select the most appropriate URI for all related authors. We used simple heuristics to choose the most suitable one. These heuristics consisted of the longest author name in the set, which is most of the time the most complete one. It also applied some rewrite rules in order to correct some misspellings (it removed spaces (_) or dots (.) at the end). One simple query to update the database table was then able to rewrite all the matching authors to this newly created or chosen URI. Because the D2R mapping selected the author names from our database table, we did not need to change our mapping. All the authors were bundled and had one single URI.

5.4 Conclusion

This chapter showed how we tackled the disambiguation problem in order to perform automatic interlinking. We first identified a dual problem: an identification and a disambiguation problem. We then showed that string similarity metrics solve the identification process. We discussed the different methods to tackle the disambiguation process and chose a relational-based resolution method to resolve our entities.
In order to lower the total amount of work, we looked at two different tools to perform the automatic interlinking (Silk and LinQuer). We explained the differences between the tools and the reason we chose Silk to work with. In order to do relational-based entity resolution we needed to alter the Silk tool. We showed what the extension looks like and how it works. The extension improved the results.
6 A Linked Data application

As explained in this thesis document we worked with the Software Technology Department’s publications database to validate our general approach for transforming a (legacy) dataset into RDF and create a Linked Data application out of it. Chapter 4 explained that we reused the SWRC ontology and converted all the relevant data to RDF. We were able to interlink our entities to entities in the Linked Data cloud and stored all the ST Publications data in an RDF triple-store \(^\text{17}\). We created a Linked Data application out of it and added different datasources in order to show the added value. In this chapter we will explain what the benefits of the tool are and where we enriched the ST Publications dataset.

6.1 Application overview

The application is located at: http://wisserver.st.ewi.tudelft.nl and delivers two different views of the application. It servers both an HTML as well as an RDF view of the data. This means that the application can be viewed with both normal webbrowsers, as well as with semantic browsers. Every resource has its own URI, which is dereferencable to two different URIs. An example:

- http://wisserver.st.ewi.tudelft.nl/resource/author/Geert-Jan+Houben, this is the URI (as it is saved in the datastore) for an author named: Geert-Jan Houben.

- http://wisserver.st.ewi.tudelft.nl/page/author/Geert-Jan+Houben, this is the URI for ordinary webbrowsers and serves an HTML view. If a webbrowser tries to locate the first URI, it will automatically be send to this one.

- http://wisserver.st.ewi.tudelft.nl/data/author/Geert-Jan+Houben, this is the URI for an RDF view. Semantic browsers (which accept RDF headers) will automatically be send to this URI.

The backend only uses an RDF triple store. This means that the frontend programming language is independent of the backend, as long as it can connect with an RDF triple store. The main RDF classes it describes are authors and their publications.

6.2 Enrichment of the dataset

The ST publications dataset has been interlinked with three different datasets (DBLP\(^\text{18}\), DBpedia\(^\text{19}\) and GeoNames\(^\text{20}\)). Because of this interlinking we were able to enrich the existing dataset (see Figure 8).

\(^{17}\)The store can be found here: http://wisserver.st.ewi.tudelft.nl:8893/sparql
\(^{18}\)http://dblp.l3s.de/d2r/
\(^{19}\)http://dbpedia.org/
\(^{20}\)http://geonames.org/
All the enrichments have been categorized (depicted with different colors in Figure 8):

- The author names have been merged, which means that every author now has a single URI. Because of this single URI the application is, for example, able to show aggregated views of authors (discussed in section 6.2.1).

- The locations of the publications are linked against locations in GeoNames. Because of this interlinking, the application is now able to retrieve information about the real-world location and can, thus, show the location on a map or print information about the specific city (discussed in section 6.2.2).

- The publishers of the publications are linked against DBpedia. This means that a lot of information about the type of publisher or location of the publisher can be showed in the application (discussed in section 6.2.3).

- The authors are linked against authors in DBLP. Because of this interlinking the application is able to show more information about the authors and their publications (discussed in section 6.2.4).

All of these extensions will be discussed in the following sections.

### 6.2.1 Merge authors and publish

The first enrichment is based on the fact that every author now has its own URI. This extension, the merging of the authors, is based on the work we
explained in chapter 5. Because we performed this merging, we are able to create visualizations of the different authors we were not able to create before. An example of such a visualization is the total amount of papers written by different authors by year. Figure 9 shows an example of all the different publications of a single author.

![Figure 9: Aggregation of publications per author](image)

Before this enrichment, all these publications were listed under different names (e.g. ‘A. van Gemund’ or ‘Arjan van Gemund’). An even bigger enrichment is the exploitation of the fact that every author now has a single URI. Because this is the case, we were able to create entity links between the authors in the ST Publications dataset and, for instance, DBLP.

### 6.2.2 Location links against GeoNames

The GeoNames dataset covers all countries and contains over eight million places with related information (e.g. population, capital etc.). We linked the locations of the publications in the ST Publications dataset against the publications in DBLP. We used the GeoNames API for this purpose and wrote a scraper script to extract the data\(^\text{21}\). Because we created these links, we resolved our location strings into real world places (a string ‘Berlin, Germany’ becomes a city ‘Berlin in Germany’). With these links we had a lot of extra data we could use. Some examples are:

- Aggregation: show all publications based on the continent they were published.
- Visualization: plot the location of a publication on a map.

\(^{21}\)Located over here together with other code discussed: [linkedentities.googlecode.com](linkedentities.googlecode.com)
• Visualization: create a world map for every author to see in which countries her or she published a publication (see Figure 10).

![World map of publications locations per author](image)

**Figure 10: World map of publications locations per author**

Because of the connections already available in the Linked Data cloud, the GeoNames dataset is already connected to the DBpedia dataset. This means that all information about a location in DBpedia is also available. We used this to show an image of the location, print the abstract information and give some statistical data (see Figure 11).
6.2.3 Publishers links against DBpedia

We also enriched the knowledge of publishers in the ST Publications dataset. Because we created links against publishers in DBpedia, we had a lot of extra information. An example can be given with: ‘IEEE publishers’. This ordinary string will be resolved in a real publisher after the connection with DBpedia. We are able to show a lot of information about this publisher, while we were not able to do this before we created the links. Examples of information about publishers we added to our dataset are: location of a publisher, hyperlinks to the websites of the specific publisher and information about other activities or services a publisher can deliver.
6.2.4 Author links against DBLP

In chapter 5 we explained how we merged the authors in our dataset and how we created a clean dataset with URIs for every individual author. Because we were able to create these URIs, we could connect them to DBLP in the cloud. This gives us extra information (e.g. information about publications which are not in the ST Publications database, but do exist in the DBLP dataset). An example of this extra information is the venue a publication was published. Because DBLP denotes all the venues of publications in their dataset it is possible to collect all the other publications of a certain venue and offer the information about these publications to the user. Another great benefit of these links is the fact that developers can create their own mash-ups with these links in order to add information they would like to see.

6.3 Example of topic retrieving due to entity relations

The DBLP-via-SOAP application\textsuperscript{22}, created within the NeOn project\textsuperscript{23} and based on records in the DBLP dataset, collects a lot of extra information about publications. The DBLP-via-SOAP application extracts topics and co-authorship relations of authors from the DBLP dataset. Because we created links against DBLP, we were able to retrieve and exploit this information. The information retrieval process will be explained in this section. The process can be seen in Figure 12.

![Diagram of the DBLP-via-SOAP application](image)

Figure 12: Using Linked Data to retrieve topics per author

The DBLP-via-SOAP application delivers an HTML-page for every author in the DBLP dataset. This data needed to be extracted in order to understand where the topics of the author and the co-authors are located in the HTML-file. This extracted data needed to be added to our local store in order to use it.

\textsuperscript{22}http://neon-project.org/aspl-v2/
\textsuperscript{23}http://www.neon-project.org/
in the end-application. In order to do this, different steps were taken. The different steps shown in Figure 12 will be discussed now.

1. Each author needs to be requested from the triple store.

2. The link to DBLP (existing in the triple store) is followed to retrieve the DBLP URI.

3. The DBLP URI is passed to a Yahoo-pipes extraction process which extracts the information from the page in the Neon-Project application and creates a JSON object (an object in JavaScript) from it.

4. The JSON object is extracted and MySQL queries are adding the data to the MySQL database.

If this process has been finished for every individual author:

1. The D2R server maps the MySQL data and creates RDF.

2. For speed purposes, the data is fetched into another datastore (Virtuoso), which is being used by the final application.

With this extraction process we had the topics and co-authors of every author available in our datastore. We used them to visualize the topics per author (see Figure 13) and show co-author graphs (see Figure 14).

![Figure 13: Topics of an author](image)
6.4 Conclusion

In this chapter we showed what the final Linked Data application of the ST Publications dataset looked like. We created links with the DBpedia, GeoNames and DBLP datasets and enriched our dataset with this external information. We showed what benefits we had from the authors disambiguation process and how we were able to access and use data because of the links we created with other datasets. This application shows that a lot of valuable information can be added to a dataset in a quick and efficient way. It also shows that because the entity links have been created it becomes possible to look further than only Linked Data datasets. The extraction method we used to retrieve topics and co-authors via a link against DBLP (explained in section 6.3) illustrates this extra insight.
Part III
Conclusions and future work
7 Conclusions

In this chapter we will review our research question and final goal in order to understand whether or not the goal is reached and whether the research questions were answered.

The research goal we defined was:

Develop an approach to create Semantic Web applications by transforming a legacy (relational) dataset into RDF, enriching this RDF by (automatically) interlinking entity URIs of several different datasets from the Linked Data cloud, and publishing the resulting RDF as Linked Data.

In order to develop an approach it was necessary to understand the different Semantic Web technologies needed to create a Semantic Web application. Techniques like ontologies and RDF to define and create structured data as well as SPARQL to query this structured data must be understood. In our approach we showed that it is good to first create a schema for the dataset to be converted to RDF. With help of this schema an ontology can be defined. It is efficient to work with an existing ontology in order to keep ontologies consistent and reduce redundancy. If an ontology is defined, the data can then be mapped onto this ontology (in RDF data) using different tools. These tools will introduce new problems so that decisions have to be taken dependent on the situation and specific needs. After the RDF is created, the interlinking process can be started. In order to choose which datasets you want to link against it is important to decide what information is relevant to enrich your dataset with and whether this information is available. It is essential to keep in mind that datasets can be enriched by other developers as well, as long as the links and RDF data is exposed as Linked Data. In our approach we created a semantic aware tool to perform automatic interlinking between entities. The publishing part can be done manually or by using the mapping tool. Throughout this document we worked with an example test dataset, the ST Publications dataset. We converted this dataset to RDF and created a Semantic Web application out of it (to be seen here: http://wisserver.st.ewi.tudelft.nl). We enriched our dataset with information including geographical and bibliographical data. Examples of these enrichments are: location-based information about publications, world-maps of every publication of an individual author and an overview of the topics and co-authors of individual authors. Because of the links created in the interlinking phase, the application becomes part of the Linked Data cloud and more ideas and enrichments can be implemented (e.g. show the publications published in a city with a population greater than 100,000 people). And because of the enrichments, datasets become of greater use and value.

7.1 Summary per research question

This thesis project addressed the research goal by first answering the research questions. We will discuss the five research questions below.
1. What is the Semantic Web and which techniques are used in developing applications in the Semantic Web?

In chapter 2 we described that the Semantic Web is a self-adaptable and machine-readable web according to the ideas of the inventor of the web, Tim Berners-Lee. We showed the Semantic Web technology stack and explained the most important techniques (ontologies, XML, RDF and SPARQL) in order to create a Semantic Web application.

2. What is Linked Data and what can Linked Data be used for in the Semantic Web?

In chapter 3 we defined what Linked Data is and discussed an application (Revyu) and a real-world example by the BBC. We also explained the basic principles of data interlinking and database to RDF mapping. The knowledge of the first and second research question is the basic knowledge for creating a Linked Data application.

3. What steps are needed in transforming a (legacy) dataset into Linked Data?

In chapter 4 we explained the steps needed for transforming a (legacy) dataset into Linked Data. We discussed this question with the ST Publications dataset as our running example. We showed some problems we needed to tackle and some decisions we had to make in order to define our application.

4. How can the automatic interlinking process between entity URIs in Linked Data be optimized?

The interlinking procedure needed to be optimized in order to create complex relations between datasets. We discussed this in chapter 5. Using our ST Publications dataset example, we explained that the links between authors are harder than they appear, because there is a difference between syntactical and semantic similarity. In this chapter we explained the basic problems with interlinking: identifying and disambiguating. Both can be solved with tools. We extended the Silk tool in order to create the best results.

5. Demonstrate the approach for application on the ST Publications data. What are the results if we apply the steps to the ST Publication data?

In chapter 6 we showed the results we achieved after connecting the ST Publications dataset to other datasets. Because we connected the dataset to GeoNames, DBpedia and DBLP, we were able to retrieve different sorts of information relevant for enriching the ST Publications dataset.

Going back to our research goal we can state that the goal has been achieved. All the research questions to reach our goal have been answered in a satisfactory way. We showed that transforming a (legacy) dataset to Linked Data is a hard process. We looked at the problem in a global way, although we sometimes had to make some specific decisions. However, it is impossible to overcome this. We solved real-world problems which were necessary to tackle in order to achieve the global goal: enrichment of a dataset.
7.2 Final remark

It is the author’s belief that Linked Data applications will grow. There is a big opportunity for companies to turn their (legacy) datasets into Linked Data and connect their knowledge to knowledge out on the web. It is clear that companies benefit from exposing their data, because a lot of websites could use this data in order to deliver more information about products to their clients. It is very interesting to see the improving research results within the Semantic Web and how Linked Data is finding its way to become mainstream.
8 Future work

Following the investigations described in this thesis, a number of projects can be started in order to solve some open problems. We will deal with these different problems to show what future work can be done.

8.1 Mapping relational databases to RDF

One of the biggest challenges within the Semantic Web is the conversion of (relational) databases to RDF. Within this research area there has been done a lot of work already, but the problem is not yet solved in a satisfactory way. The ontologies are hard to generate automatically from a (legacy) dataset and there is, at the moment, a lot of manual work necessary to achieve a nice mapping. In order to enhance this process it is important to look at existing ontologies and try to map databases to these existing ontologies. Tools can be useful to create a bridge between automatic and manual ontology mapping.

8.2 Defining automatic interlinking with ontologies

In chapter 5, we showed that current tools like Silk and LinQuer allow a user to define an interlinking between two different datasets. These datasets, however, are very hard to interlink if the structure becomes complex. We introduced an extension to Silk in order to achieve a better result. It is very interesting to see whether or not the definition of the interlinking can be done based on ontology alignment. If a machine can try to find relations between two ontologies, it will decrease the manual labor a lot. The idea is to first find correspondences between the ontology describing datasets (when they exist) before looking for links at the instance level. It may be possible to induce from the alignment a link specification which is a synthetic description for a set of links.

8.3 Model the enrichments

Enrichments are very specific and different for all kinds of datasets. Different people will create different enrichments suiting their specific needs. At the moment it is up to developers whether or not to create enrichments. If it becomes possible to approach enrichments at a more abstract level, ordinary users can create and add their enrichments as well. An approach to this problem can be that content and structure of datasets is exposed in a clear user interface, where users can drag and drop datasets in order to connect them. If ordinary people can enrich datasets, the reuse of Linked Data will be improved.
References


Appendix

A Description of the used ontologies and vocabularies

<table>
<thead>
<tr>
<th>Name</th>
<th>Location of vocabulary</th>
<th>Description of vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>swrc</td>
<td><a href="http://swrc.ontoware.org/ontology#">http://swrc.ontoware.org/ontology#</a></td>
<td>The SWRC (Semantic Web for Research Communities) is an ontology for modeling entities of research communities such as persons, organisations, publications (bibliographic metadata) and their relationships.</td>
</tr>
<tr>
<td>dcterms</td>
<td><a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a></td>
<td>DCMI (Dublin Core) Namespace for metadata terms in the dcterms namespace</td>
</tr>
<tr>
<td>foaf</td>
<td><a href="http://xmlns.com/foaf/0.1/">http://xmlns.com/foaf/0.1/</a></td>
<td>Friend of a Friend (FOAF) vocabulary</td>
</tr>
<tr>
<td>dc</td>
<td><a href="http://purl.org/dc/elements/1.1/">http://purl.org/dc/elements/1.1/</a></td>
<td>See dcterms</td>
</tr>
<tr>
<td>rdfs</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
<td>The RDF Schema vocabulary</td>
</tr>
<tr>
<td>rdf</td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a></td>
<td>The RDF Vocabulary</td>
</tr>
<tr>
<td>owl</td>
<td><a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a></td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>dbpedia-owl</td>
<td><a href="http://dbpedia.org/ontology/">http://dbpedia.org/ontology/</a></td>
<td>The DBpedia ontology</td>
</tr>
</tbody>
</table>

Table A.1: Vocabularies used within the application

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>swrc:Conference</td>
<td>A group of articles of a conference</td>
</tr>
<tr>
<td>foaf:Person</td>
<td>A person</td>
</tr>
<tr>
<td>foaf:Document</td>
<td>A document</td>
</tr>
</tbody>
</table>

Table A.2: The different classes describing our entities
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc:creator</td>
<td>An entity primarily responsible for making the resource</td>
</tr>
<tr>
<td>dc:identifier</td>
<td>An unambiguous reference to the resource within a given context</td>
</tr>
<tr>
<td>dcterms:publisher</td>
<td>An entity responsible for making the resource available</td>
</tr>
<tr>
<td>dc:subject</td>
<td>The topic of the resource</td>
</tr>
<tr>
<td>dc:title</td>
<td>A name given to the resource</td>
</tr>
<tr>
<td>dc:type</td>
<td>The nature or genre of the resource</td>
</tr>
<tr>
<td>swrc:month</td>
<td>The specific month of creation</td>
</tr>
<tr>
<td>dcterms:issued</td>
<td>Date of formal issuance</td>
</tr>
<tr>
<td>swrc:editor</td>
<td>Editor of the resource</td>
</tr>
<tr>
<td>swrc:pages</td>
<td>Amount of pages of the resource</td>
</tr>
<tr>
<td>swrc:series</td>
<td>Series in which the resource was published</td>
</tr>
<tr>
<td>swrc:volume</td>
<td>Volume nr in which the resource was published</td>
</tr>
<tr>
<td>rdf:type</td>
<td>The subject is an instance of a class.</td>
</tr>
<tr>
<td>rdfs:label /dc:title</td>
<td>A human-readable name for the subject.</td>
</tr>
<tr>
<td>owl:sameAs</td>
<td>The property that determines that two given individuals are equal.</td>
</tr>
<tr>
<td>foaf:maker / dc:creator</td>
<td>Relates something to a foaf:Agent that foaf:made it</td>
</tr>
<tr>
<td>foaf:name</td>
<td>A name for some thing.</td>
</tr>
<tr>
<td>swrc:isbn</td>
<td>The ISBN number of a thing</td>
</tr>
<tr>
<td>dbpedia-owl:place</td>
<td>A specific place in the world</td>
</tr>
<tr>
<td>swrc:topic</td>
<td>A specific research topic</td>
</tr>
</tbody>
</table>

Table A.3: The different properties used to describe our entities
B Example of the D2R mapping

```plaintext
@prefix do: <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dmpedia-owl: <http://dmpedia.org/ontology/> .
@prefix d2r: <http://sites.wiwiss.fu-berlin.de/uhl/bizer/d2r-server/config.rdf#> .

c a d2r:Server;
  d2r:label "D2R Server - ST publications database";
  d2r:baseURI <http://127.0.0.1/>;
  d2r:port 80;
  d2r:documentMetadata {
    rdfs:comment "This document describes information from the ST publications database.";
  };
  d2r:vocabularyIncludeInstances true;
.
.
map:database a d2r:Database;
  d2r:jdbcDriver "com.mysql.jdbc.Driver";
  d2r:jdbcURL "jdbc:mysql://dev.iva.uni-heidelberg.de/publications?";
  d2r:username "sooc";
  d2r:password "9noc00-iv0di";
  jdbc:autoreconnect "true";
  jdbc:zeroDateTimeBehavior "convertToNull";
.
# Table proceedings
map:proceedings a d2r:ClassMap;
  d2r:belongsToToClassMap map:database;
  d2r:uriPattern "@proceedings|@title|@urlenode\$\$";
  d2r:class foaf:Document;
  d2r:column Proceedings;
  d2r:propertyNameLabel "proceedings";
.
map:proceedings_label a d2r:PropertyBridge;
  d2r:belongsToToClassMap map:proceedings;
  d2r:property rdfs:label;
  d2r:property d2r:title;
  d2r:pattern "@proceedings|@title\$\$";
.
map:proceedings_year a d2r:PropertyBridge;
  d2r:belongsToToClassMap map:proceedings;
  d2r:property rdfs:label;
  d2r:property d2r:issueld;
  d2r:propertyDefinitionLabel "proceedings year";
  d2r:column "proceedings_year";
```

Figure A.1: Example of D2R mapping

The above Figure A.1 shows the mapping file used to map the relational data to RDF. The mapping starts with the prefixes of the ontologies used in the mapping (line 1-5). After these prefixes the basic configuration details are provided (lines 7-15). The lines 17-23 represent the database connection which is needed to tell D2R where the relational data is located. Every RDF class starts with a `d2rq:ClassMap` in the mapping file (line 26). The URI for the class can be set and the class can be defined using the prefixes (line 29 and 30). After this definition every property of the class is defined using a `d2rq:PropertyBridge` (line 33-37 and 39-43). These steps are performed for every class in the RDF data.